TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Telemetrický systém pro studentskou formuli FS TUL Racing

Bakalářská práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 2612R011 – Elektronické informační a řídicí systémy

Autor práce: **Ondřej Vacek**
Vedoucí práce: Ing. Jan Koprnický, Ph.D.

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Telemetrický systém pro studentskou formuli FS TUL Racing

Jméno a příjmení: **Ondřej Vacek**
Osobní číslo: M18000201
Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídicí systémy
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2020/2021**

Zásady pro vypracování:

1. Seznamte se s pravidly závodů Formula Student.
2. Navrhněte telemetrický systém pro získávání dat z řídicí sběrnice závodního vozu.
3. Realizujte navržený systém.
4. Otestujte realizovaný systém na vozu v podmínkách závodu.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] DIETSCHE, Karl-Heinz a Konrad REIF, ed. *Automotive Handbook*. 10th edition, revised and extended. Karlsruhe: Robert Bosch, 2018, 1750 s. ISBN 978-111-9530-817.

[2] Formula SAE [online]. USA: SAE, 2019 [cit. 2020-10-06]. Dostupné z: http://www.fsaeonline.com

[3] Formula Student Germany [online]. Germany: FSG, 2019 [cit. 2020-10-06]. Dostupné z: https://www.formulastudent.de

[4] CARDEN, Frank, Russell P. JEDLICKA a Robert HENRY. *Telemetry Systems Engineering*. 2. Boston: Artech House, c2002. Artech House telecommunications library. ISBN 1-58053-257-8.

Vedoucí práce:	Ing. Jan Koprnický, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	9. října 2020
Předpokládaný termín odevzdání:	17. května 2021

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 9. října 2020

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

16. 5. 2021 Ondřej Vacek

Telemetrický systém pro studentskou formuli FS TUL Racing

Abstrakt

Tato bakalářská práce je rozdělena na dvě hlavní témata. První z nich je návrh a realizace telemetrického zařízení. Druhé téma se zabývá zpracováním získaných dat a jejich vizualizací. Výsledný produkt je implementován do monopostu týmu FS TUL Racing, který je zaštítěn projektem Formula Student.

Klíčová slova: Telemetrie, Formula Student, Python, C++, Qt, CAN bus, ESP32

Telemetry system for student formula FS TUL Racing

Abstract

This bachelor thesis is divided into two main topics. The first is the design and implementation of the telemetry device. The second topic deals with the processing of acquired data and their visualization. The resulting product is implemented in the monopost of the FS TUL Racing team, which is backed by the Formula Student project.

Keywords: Telemetry, Formula Student, Python, C++, Qt, CAN bus, ESP32

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Janu Koprnickému, Ph.D. za poskytnuté rady. Dále bych rád poděkoval celému týmu FS TUL Racing za možnost být součástí tohoto projektu a Jonáši Závěrkovi za navržení designu krabičky. Speciální poděkování patří Ing. Michalovi Hudci, za předání všech potřebných vědomostí a zkušeností.

Obsah

Seznam zkratek

ADU	Advanced Display Unit
AP	Access point
API	Application Programming Interface
CAN	Controller Area Network
CI	Continuous Integration
CV	Combustion Vehicles
DV	Driverless Vehicles
ECU	Electronic Control Unit
FS	Formula Student
FSG	Formula Student Germany
FSPL	Free Space Path Loss
GUI	Graphical user interface
IP	Internet Protocol
IPxx	Ingress Protection
ISO	International Organization for Standardization
IT	Information Technology
LoRa	Long Range
PMU	Power Management Unit
SAD	Seriously Approximate Deviation
SAE	Society of Automotive Engineers
SoC	System on a Chip
TCP	Transmission Control Protocol
TDD	Test-driven development
TUL	Technická univerzita v Liberci
UDP	User Datagram Protocol
USB	Universal Serial Bus
Wi-Fi	Wireless Fidelity

Seznam obrázků

Seznam grafů

Seznam tabulek

Seznam zdrojových kódů

Úvod

V první části bakalářské práce se seznámíme s projektem Formule Student jakožto celkem a představíme si tým FS TUL Racing, který funguje pod záštitou Technické univerzity v Liberci. Kromě představení celého konceptu Formule Student, si zde také uvedeme pravidla, která souvisejí se zadaným tématem, tedy s telemetrickým systémem.

Druhá část bakalářské práce bude věnována představení konceptu telemetrie. Zároveň budou specifikovány určité požadavky, které by měl výsledný systém splňovat. Požadavky stanovíme jak pro fyzickou tak i pro aplikační část.

Ve třetí části bakalářské práce se budeme zabývat fyzickou části telemetrického systému. Jako první si představíme sběrnici CAN, ze které budeme získávat všechny data. Poté si porovnáme několik způsobů přenosu dat za využití bezdrátových technologií. Pro vybranou bezdrátovou technologii si vypočítáme její limity z pohledu dosahu signálu. Po absolvování této teoretické části přejdeme již k samotnému návrhu. Zde si nejprve navrhneme elektrické schéma, ke kterému následně vytvoříme desku plošných spojů. Nakonec se pustíme do tvorby firmwaru, který si vytvoříme ve dvou verzích. Jeden pro zařízení umístěné ve formuli a druhý pro případný mezičlánek na stranu uživatele.

V poslední části se budeme zabývat vývojem aplikace pro zpracování, sběr a vizualizaci získaných dat v reálném čase. Jelikož aplikace musí být dlouhodobě udržitelná, tak si zde představíme koncepty jako jsou správa verzí, testování, dokumentace aj. Abychom udržitelnost kódu neprezentovali pouze teoreticky, tak si zde tyto koncepty předvedeme přímo na aplikaci. Následně si představíme již samotnou aplikaci. Ta bude mimo jiné obsahovat zobrazení dat v několika režimech, umožní uživateli nastavit parametry sledovaných dat a dovolí průběžně ukládat přijatá data. Nakonec si podrobně probereme jednotlivé části aplikace jako například GUI a nebo zpracování dat.

Vytvořený telemetrický systém bude implementován do nejnovějšího monopostu týmu FS TUL Racing.

1 Formula student

Formula student je celosvětová studentská soutěž v konstrukci vozů formule. Tato soutěž vznikla roku 1981 v USA pod záštitou SAE. Od té doby se rozšířila a nyní existují stovky studentských týmu po celém světě. V této soutěži musí každý tým navrhnout a postavit své vlastní závodní vozidlo. S tímto postaveným vozem se následně účastní závodů po celém světě. Nejenom, že týmy porovnávají síly s ostatními na závodní trati, ale také dostávají zpětnou vazbu na své návrhy od odborníků v dané oblasti, kteří se závodů účastní. Hlavním záměrem tohoto projektu bylo a stále je rozvíjet znalosti a dovednosti mladých inženýrů, naučit je týmové spolupráci a v neposlední řadě je připravit na práci pod tlakem. Každý tým v této soutěži by se dal přirovnat k firmě, která si musí sama obstarat vše od financování a marketingu přes konstrukci a vývoj až po finální produkt. Díky těmto náročným úkolům mají lidé podílející se na tomto projektu více zkušeností, jež dokáží firmy velice ocenit [1].

V posledních letech se stále více dostávají do popředí elektrické a nyní i autonomní automobily. Projekt Formula Student na to reagoval tím, že vytvořil dvě nové kategorie, které se zabývají přesně těmito tématy. Momentálně tedy existují 3 kategorie, ve kterých mohou týmy soutěžit:

1. `Combustion vehicles` – vozidla se spalovacími motory
2. `Electric vehicles` – vozidla s elektrickými motory
3. `Driverless vehicles` – vozidla, která jsou schopna jezdit sama bez přítomnosti řidiče, pouze za využití senzorů a výpočetní techniky

V následujících letech liberecký tým FS TUL Racing plánuje přejít z kategorie Combustion vehicles do kategorie Driverless vehicles a to při zachování spalovacího motoru.

1.1 Závody

Jakmile tým postaví svůj monopost, má možnost se zúčastnit závodů, které se v průběhu léta konají po celém světě. Než však bude moci se svým monopostem nastoupit na trať, musí projít celkem 4 testy:

- `Mechanical Inspection` – kontrola dodržení pravidel při konstrukci vozu
- `Tilt Test` – při naklonění vozu o 60° nesmí dojít k úniku žádné kapaliny a vozidlo musí zůstat všemi koly v dotyku s nakloněnou rovinou
- `Noise Test` – vozidlo nesmí přesáhnout danou úroveň hlasitosti při daných otáčkách
- `Brake Test` – vozidlo musí z určité rychlosti zastavit tak, aby zůstalo v rovině vůči trati

Pokud vůz projde všemi výše zmíněnými technickými přejímkami, je způsobilý k závodům. Jak již bylo zmíněno, soutěž není jenom o závodění, ale také o získávání zkušeností. Z tohoto důvodu mají soutěže dva typy disciplín. První z nich jsou *statické disciplíny*, kde se tým setká s odborníky z různých oblastí a ty je hodnotí v několika oblastech. Statických disciplín se může tým zúčastnit i v případě, že neprojde technickými přejímkami nebo dokonce i pokud na závody přijde bez monopostu. Druhým typem disciplín jsou *dynamické disciplíny*, kde týmy porovnají své vozy v několika druzích závodů.

Statické disciplíny:

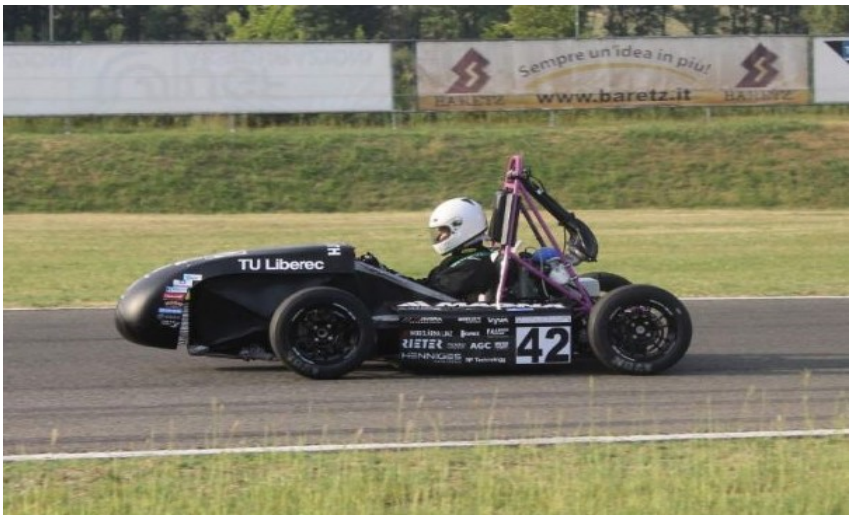
- `Business Plan` – fiktivně vytvořený podnikatelský záměr představený potencionálním investorům
- `Cost and Manufacturing Report` – ocenění celého vozu včetně výrobních nákladů, pronájmů za prostory, atd.
- `Engineering Design Report` – prezentace návrhu vybraných částí vozu od výpočtů přes simulace až po validaci

Dynamické disciplíny:

- `Skidpad Event` – technická trať ve tvaru osmičky
- `Acceleration Event` – rovinka dlouhá 75 m
- `Autocross Event` – 1 km dlouhá, úzká trať s velkým množstvím zatáček
- `Endurance and Efficiency Event` – struktura trati stejná jako u autokrosu pouze s tím rozdílem, že délka trati je 22 km a kromě času se hodnotí i množství spotřebovaného paliva

1.2 FS TUL Racing

FS TUL Racing je tým studentské formule pod záštitou Technické univerzity v Liberci. První formule se jménem Eliška byla představena v roce 2017, avšak její vývoj započal už o dva roky dříve. Po Elišce tým postavil ještě další dva vozy, Markétu a Annu. Tým od té doby ušel veliký kus cesty, což je na první pohled patrné z obrázků 1.1 a 1.2. O pokroku týmu nevypovídá jenom zřejmá změna v konstrukci, ale také úspěchy. Momentálně největším úspěchem je 3. místo v dynamické disciplíně skidpad na závodech v českém Mostu z roku 2019.



Obr. 1.1: První formule, Eliška

V současnosti se na stavbě a vývoji formule podílí přibližně 10 studentů. Studenti jsou převážně z Fakulty strojní a z Fakulty mechatroniky. V týmu je i pár studentů z Fakulty ekonomické a nebo Fakulty textilní.

Obr. 1.2: Třetí formule, Anička

1.3 Pravidla týkající se telemetrie

Každý tým musí při návrhu a stavbě formule dodržovat pravidla soutěže Formula Student, které definují určitě hranice při stavbě vozu a také zajišťují bezpečnost postaveného vozu. Tyto pravidla jsou každý rok vydávány některými z předních pořadatelů soutěží. Náš tým se řídí pravidly vydávanými německými pořadateli, konkrétně se jedná o nejnovější vydání z roku 2020. Nová pravidla pro rok 2021 nebyla vydána. Důvodem je zrušení závodů v roce 2020 kvůli koronavirové krizi.

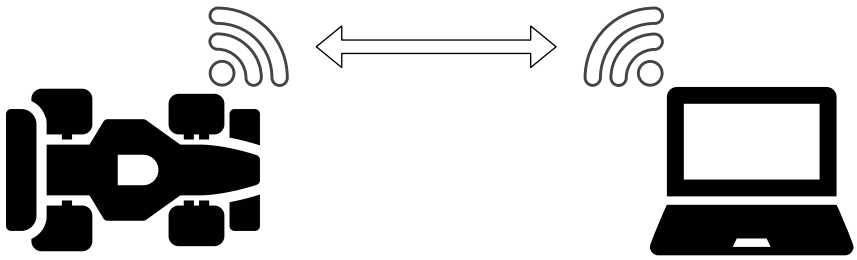
Ohledně telemetrických zařízení jsou pravidla velmi stručná. Dokonce tak, že zmínka o telemetrických zařízení nebo obecně o bezdrátových zařízeních je pouze v sekci definující pravidla pro autonomní vozy. I přesto, že náš vůz není autonomní, tak se tímto pravidlem budeme také řídit, abychom předešli zbytečným sporům při technické přejímce. Konkrétně se tedy jedná o pravidlo **DV 1.2.1**.

- `DV 1.2.1` – Je zakázáno měnit parametry, odesílat příkazy nebo jakkoliv zasahovat do softwaru pomocí bezdrátové komunikace. Příjem informací z vozidla prostřednictvím jednosměrné telemetrie je povolen. Během dynamických událostí může být bezdrátová komunikace omezena. Nerušené a spolehlivé bezdrátové připojení není pořadateli zaručeno [2].

2 Požadavky na telemetrický systém

Před návrhem jakéhokoli systému je velice důležité správně definovat požadavky na systém. K tomu, aby se mohly určit požadavky na telemetrický systém, tak se nejdříve musí definovat, co to vlastně telemetrický systém je a jak funguje.

Telemetrický systém je systém, který slouží k automatickému zaznamenávání a přenosu dat, do vzdáleného IT systému. Přenesená data jsou následně dále zpracována a použita pro monitorování a analýzu sledovaného systému [3]. Obecné schéma telemetrického systému je na obrázku 2.1.



Obr. 2.1: Telemetrický systém

Nyní, když byla uvedena obecná definice telemetrického systému, tak je potřeba vytvořit konkrétní specifikaci pro náš systém. Aby byla specifikace přehlednější, tak se rozdělí na dvě části. První specifikace definuje *fyzickou část*, neboli fyzické zařízení, které bude umístěno ve formuli. Druhá specifikace definuje *aplikační část*, neboli software, který poběží na straně uživatele (příjemce dat).

2.1 Fyzická část

Pro fyzickou část systému bylo stanoveno celkem 5 požadavků. Seznam požadavků, seřazený podle priority, je následující:

- `Vysoká odolnost proti zarušení dat` – Data, která přijdou do zařízení musí být nezkreslená, abychom měřili skutečné hodnoty veličin. Dosáhnout takového ideálního stavu, může být v reálných podmínkách občas problém. Konkretně se jedná o zkreslení dat elektromagnetickým signálem, který by

mohl do měření vnést značnou chybu. Tento elektromagnetický signál je možné indukovat v blízkosti motoru při startování a následně při zapalování palivové směsi v jednotlivých válcích. Je tedy velice důležité vybrat místo, kam se výsledné zařízení umístí.

- `Středně daleký dosah signálu` – Při podmínkách závodu se formule v závislosti na trati a poloze členů týmu může dostat, až na vzdálenosti přibližně 300 m. Systém musí být schopný tuto vzdálenost bez problémů překonat. Stanovená minimální teoreticky vypočtená přenosová vzdálenost systému musí být alespoň 300 m při SAD faktoru minimálně 30 %. SAD faktor vysvětlen v sekci 3.3.4.

- `Minimální ztráta přenesených dat` – Spojení mezi formulí a koncovým uživatelem musí být stabilní. V ideálním případě chceme, aby všechna data odeslaná z formule byly na druhé straně správně přijata. Stanovené maximální ztráty pro tento systém jsou 3 % z celkového objemu přenesených dat.

- `Odolnost vůči vlivům prostředí` – Systém bude implementován do závodního vozu, který je otevřený a tudíž není nijak chráněn proti vlivům prostředí. Systém musí být chráněný zejména před vodou a mechanickým poškozením. Je tedy vhodné navrhnout kromě funkčního zařízení, také krabičku, která zajistí alespoň částečnou ochranu proti vlivům prostředí.

2.2 Aplikační část

U aplikační části je celkem 5 požadavků. Na rozdíl od požadavků na fyzickou část, požadavky na aplikační část nemají priority. Požadavky jsou následující:

- `Zobrazení aktuálního stavu` – V aplikaci musí být možnost zobrazit aktuální stav vozu. Tato funkcionalita by měla zastupovat přístrojovou desku. Zobrazená by měla být například teplota vody, teplota oleje, tlak oleje, otáčky motoru, zařazený stupeň, apod.

- `Zobrazení průběžného stavu` – Tento požadavek je obdobný s požadavkem na zobrazení aktuálního stavu, s tím rozdílem, že zde nebudou zobrazovány pouze aktuální hodnoty, ale všechny historicky zaznamenané hodnoty. Tyto hodnoty budou zobrazovány v grafech.

- `Konfigurace CAN zpráv` – Uživatel musí mít u každé zaznamenávané veličiny možnost nastavit její parametry. V nastavení pro jednotlivé CAN zprávy musí být možnost nastavit CAN ID, počáteční bit, délku, násobitel, offset a endian.

- `Konfigurace připojení` – Uživatel musí mít možnost nastavit parametry komunikace. Parametry mohou být IP adresa, port, přenosová rychlost a jiné v závislosti na zvoleném typu komunikace.

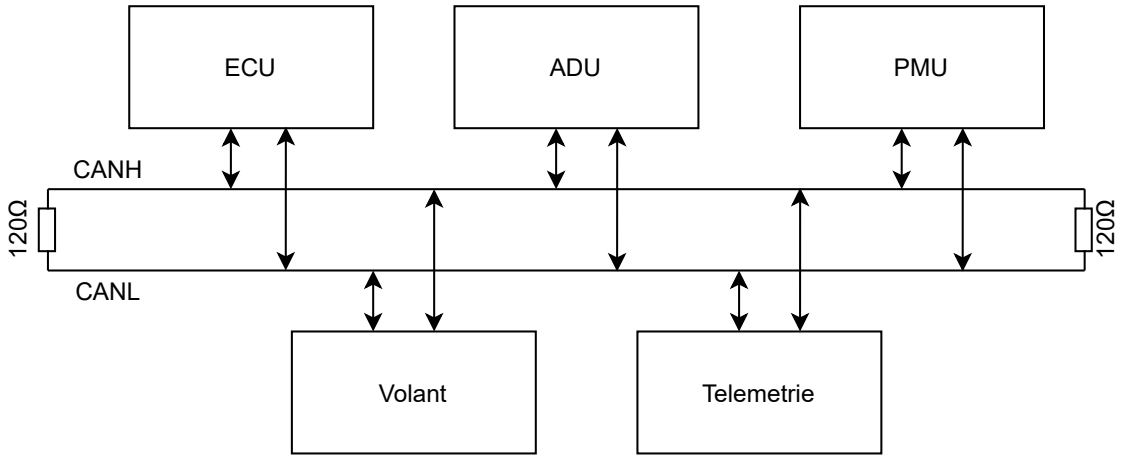
- `Zobrazení aktuálního stavu spojení` – V aplikaci musí být na viditelném místě zobrazení aktuálního stavu spojení s formulí. Stavy jsou definovány 3. První stav značí odpojeno, druhý značí pokus o připojení a třetí značí připojeno.
- `Ukládaní dat a jejich vhodná komprese` – Všechna přijatá data se budou zaznamenávat do souboru. Kvůli úspoře místa se na ukládaná data aplikuje vhodný kompresní algoritmus.

3 Fyzická část

Fyzickou částí se myslí návrh zařízení, jeho naprogramování, a vytvoření krabičky. A to včetně rešerše dané problematiky, kterou se při návrhu jakéhokoliv zařízení musí začít.

3.1 Sběrnice CAN

Formule disponuje sběrnicí CAN, která zajišťuje komunikace mezi všemi jednotkami ve formuli. Konkrétně se jedná o ECU, ADU, PMU a telemetrii, kterou se zabývá tato práce. ECU, ADU a PMU jsou všechny zakoupeny od firmy ECUMASTER. Na obrázku 3.1 je grafické znázornění CAN sběrnice na formuli.



Obr. 3.1: Aktuální zapojení CAN sběrnice

CAN sběrnice je robustní sběrnice navržena pro komunikaci elektronických zařízení v automobilovém průmyslu. Postupem času se začala používat i jako průmyslová sběrnice pro komunikaci se senzory nebo akčními členy. Oficiální představení této sběrnice bylo v roce 1986 firmou Bosch na konferenci SAE v Detroitu, Michigan. Hlavním záměrem při návrhu této sběrnice bylo snížení použitého materiálu při tvorbě elektrických svazků v automobilech. Postupem času byla sběrnice normalizována do sady ISO norem. Konkrétně nás bude zajímat norma ISO 11898-1 popisující spojovou vrstvu sběrnice CAN a norma ISO 11898-2 popisující fyzickou vrstvu sběrnice CAN [4].

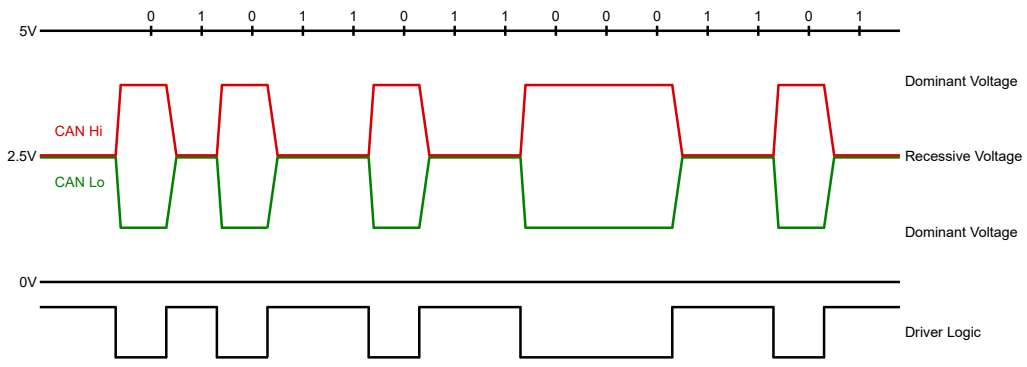
3.1.1 Fyzická vrstva

Fyzická vrstva je nejnižší vrstvou síťového modelu ISO/OSI. Zajišťuje fyzický přenos, tedy převod bitů na signál a pak zase zpět. Nejběžněji se využívá elektrický signál. Protokol fyzické vrstvy definuje kompletní rozhraní. Tedy napěťové úrovně, kmitočty, průběhy ale i tvar a velikost konektorů [5].

Fyzická vrstva sběrnice CAN je specifikována normou ISO 11898-2, taktéž nazývanou vysokorychlostní CAN. Na formuli se používá CAN 2.0, který dosahuje rychlosti přenosu až 1 Mb/s. CAN se vyznačuje dvěma logickými úrovněmi: dominantní (log 0) a recesivní (log 1). Pokud budou všechna připojené zařízení v recesivním stavu, tak je i celá sběrnice v recesivním stavu. Pokud ovšem bude, alespoň jedno zařízení vysílat, tedy bude v dominantním stavu, pak je i celá sběrnice v dominantním stavu. Podle normy ISO 11898 se pro vedení signálu nejčastěji používá dvouvodičová diferenciální sběrnice, která je na obou koncích zakončena odporem o hodnotě 120 Ω. Tato metoda ve formě kroucené dvojlinky je použita i na formuli. Kroucená dvojlinka se zde používá pro minimalizování vlivu elektromagnetického záření [4].

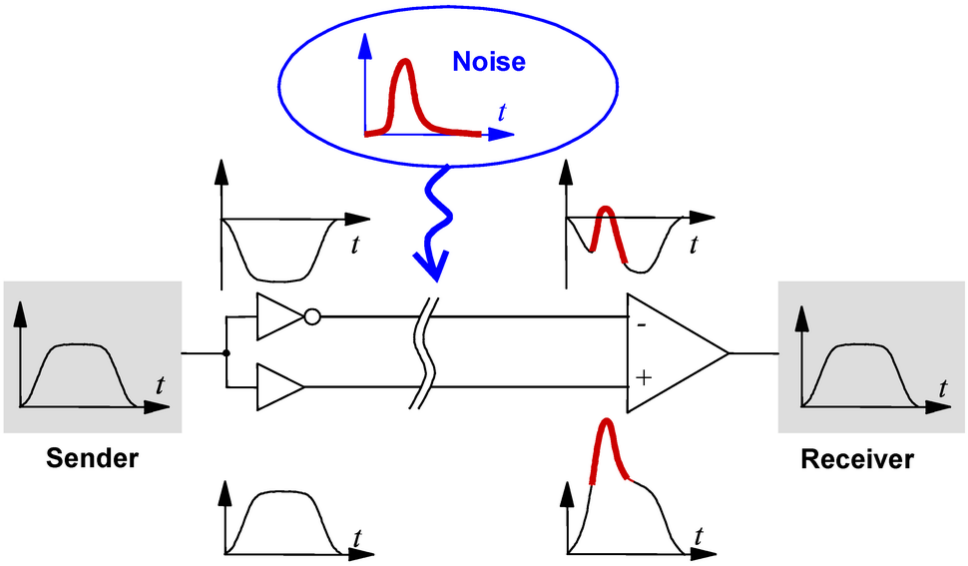
$$V_{diff} = V_{CANH} - V_{CANL} \tag{3.1}$$

Za recesivní stav je považován každý stav, kde je diferenciální napětí (3.1) menší než 0,5 V. Dominantní stav nastává jakmile je diferenciální napětí větší než 0,9 V. Nominální hodnota diferenciálního napětí je stanovena na 2 V. Na obrázku 3.2 je grafické zobrazení výše popisovaného.



Obr. 3.2: Recesivní a dominantní stav sběrnice CAN, ISO 11898-2 [6]

Jak již bylo zmíněno, využívá se zde diferenciální dvojlinky, což znamená, že signál v jedné žíle je invertovaným signálem k signálu v druhé žíle. Díky tomu je velice imunní proti zarušení. Jakmile se v signálu objeví ruch z okolí, tak bude naindukován na obou částech dvojlinky. Díky tomu, zařízení, které signál přijme, bude přesně vědět, která část signálu je informace a která pouze ruch. Zjištění probíhá odečtem napětí mezi žílami, které by v ideálním případě mělo být nulové. Tento princip je graficky znázorněn na obrázku 3.3.



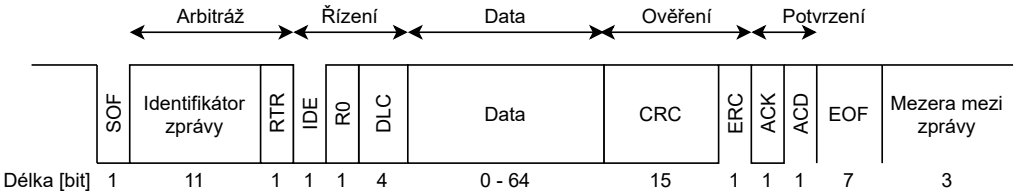
Obr. 3.3: Ukázka ruchu v diferenciálním páru [7]

3.1.2 Spojová vrstva

Spojová vrstva je druhou nejnižší vrstvou síťového modelu ISO/OSI. Zajišťuje komunikaci mezi jednotlivými uzly sítě. Spojová vrstva definuje formát datového rámce. Mimo to provádí kontrolní součet, zahazuje poškozené nebo nechtěné rámce a v případě potřeby zajišťuje jejich opakovaný přenos.

Spojová vrstva sběrnice CAN je definována normou ISO 11898-1. Její 3 základní specifikace jsou CAN 2.0A, CAN 2.0B a nejnovější CAN FD. Na formuli se používá CAN 2.0A, jejíž struktura je zobrazena na obrázku 3.4. Nová zpráva je indikována jedním dominantním bitem SOF (Start of Frame). Následuje 11bitový identifikátor, který udává prioritu přenášené zprávy. Nižší hodnota identifikátoru znamená vyšší prioritu zprávy. Bit RTR (Remote Transmission Request) slouží k rozlišení zprávy zdali jde o datovou zprávu (dominantní) a nebo žádost o přístup ke sběrnici (recesivní). Následuje 6bitové řídicí pole, kde IDE (Identifier Extension) určuje, zdali se jedná o základní nebo rozšířený formát identifikátoru. V případě základního identifikátoru musí být dominantní. R0 je rezervovány pro případné další využití. Zbylé 4 bity DLC (Data Length Code) obsahují informaci o velikosti datového pole. Datové pole může zabírat 0 až 8 bajtů [4].

CRC (Cyclic Redundancy Check) slouží k detekci chyby v kódu. Ukončen je jedním recesivním bitem ERC (End of Redundancy Check). Bity ACK (Acknowledge) a ACD (Acknowledge Delimeter) slouží pro potvrzení. Potvrzení probíhá tak, že všechna zařízení, která přijala zprávu, aniž by zaznamenala chybu, vyšlou dominantní ACK. Jakmile alespoň jedno zařízení obdrží zprávu v pořádku, tak přenos končí. Pokud žádné zařízení nevyšle dominantní ACK, tak se vysílající zařízení po-



Obr. 3.4: Datová struktura specifikace CAN 2.0A

kusí odeslat zprávu znovu. Bit ACD slouží jako oddělovač potvrzení. Celý rámec je ukončen 7 recesivními bity EOF (End of Frame) a 3 recesivními bity sloužící pro oddělení zpráv [4].

Specifikace CAN 2.0B byla zavedena z důvodu nedostačujícího počtu identifikátorů ve specifikace CAN 2.0A. Hlavním rozdílem je tedy maximální počet identifikátorů, který se z 2^{11} zvětšil na 2^{29}.

Specifikace CAN FD byla zavedena z důvodu stále rostoucích požadavků na datovou propustnost v automobilovém průmyslu. Hlavním rozdílem je, že jedna zpráva může místo původních 8 B přenést až 64 B užitečné informace a přenosová rychlost se může navýšit až na 5 Mb/s.

3.2 Bezdrátové technologie

Nyní je jasné, jak funguje přenos dat mezi jednotlivými komponenty uvnitř formule, a jak jednotlivé datové pakety vypadají. Dalším krokem je vybrat technologii, pomocí které se budou data přenášet z formule na vzdálený PC.

Při volbě technologie se omezíme pouze na přenos pomocí rádiových vln. Ostatní přenosové technologie, jako je například bezdrátová optická komunikace, nebudou brány v úvahu. Důvodem je buď velmi malý dosah a nebo potřeba přímé viditelnosti mezi vysílačem a přijímačem. Dále se při výběru omezíme na technologie, které jsou výrobci dodávané jako hotové řešení. Myšleno tak, že z fyzického pohledu bude zařízení kompletní bez nutnosti dodělávat například vysílač a všechny jeho periférie. Stejně tak v softwaru nebude potřeba dodělávat knihovny pro komunikaci nebo jakkoliv jinak zasahovat do přenosu dat.

3.2.1 LoRa

LoRa se vyznačuje malou spotřebou energie a velkým dosahem. Její dosah může být i v řádech km. V Evropě ji lze provozovat na kmitočtu 433 MHz a nebo 863–870 MHz. Může dosáhnout přenosové rychlosti až 50 kb/s. Tato technologie se hodí zejména pro IoT, kde posíláme malé datové pakety v intervalech minut až hodin. Z tohoto důvodu není úplně vhodným kandidátem pro telemetrii, kde požadujeme kontinuální přenos dat a středně velkou přenosovou rychlost [8].

3.2.2 XBee

XBee je rodina modulů firmy Digi International. Předností těchto modulů je bezdrátová komunikace, kde si uživatel může zvolit mezi krátkým dosahem (2,4 GHz) a dlouhým dosahem (900 MHz). Kromě bezdrátové komunikace jsou moduly také klasicky vybaveny i sběrnicemi, jako jsou UART nebo I2C. Sběrnicí CAN vybaveny nejsou. Při výběru je pro nás důležitá především specifikace bezdrátové komunikace. Při využití modulů s kmitočtem 900 MHz se dosahuje ve venkovním prostředí vzdáleností až nižších 10 km při rychlostech maximálně pár 100 kb/s [9]. Moduly komunikující na frekvenci 2,4 GHz využívající specifikaci IEEE 802.15.4 dosahují ve venkovním prostředí vzdálenosti i jednotek kilometrů při rychlostech až 250 kb/s [10]. I přesto, že přenosová rychlost je znatelně nižší než rychlost CAN sběrnice ve voze, tak poměr rychlost/dosah je dostatečně vhodný pro použití v telemetrickém systému.

3.2.3 Wi-Fi

Wi-Fi je souhrnné označení pro skupinu standardů IEEE 802.11, které většinou operují v pásmech 2,4 GHz a 5 GHz. Existují ale i standardy pracující v pásmu 0,9 GHz (802.11ah) nebo naopak v pásmu 60 GHz (802.11ad) [11]. Wi-Fi se převážně používá ve vnitřních prostorech pro připojení velkého množství uživatelů do jedné sítě. Jak již použití napovídá, tak se vyznačuje především velkou přenosovou rychlostí. Například nejnovější standard 802.11ax nabízí přenosovou rychlost až téměř 10 Gb/s [12]. Na druhou stranu běžné zařízení využívající Wi-Fi dokáží komunikovat pouze na vzdálenost nižších stovek metrů. Výjimkou je výše zmíněný standard 802.11ah, který by měl být schopen komunikovat až na vzdálenost 1 km [13]. Velkou výhodou využití Wi-Fi je možnost přímé komunikace s koncovým PC.

3.2.4 Výběr přenosové technologie

Z výše popsaných technologií jsou vhodné pouze XBee a Wi-Fi, kde XBee je lepší v oblasti dosahu a Wi-Fi v oblasti rychlosti přenosu. Další výhodou použití Wi-Fi je možnost přímého propojení koncového zařízení a uživatelského PC, což se dost promítne i na konečné ceně takového zařízení. Nyní by bylo vhodné vybrat jednu z výše uvedených technologií a obhájit její výběr, což by zajisté nebylo těžké. Ovšem rozumnější je přijít s otázkou, jestli opravdu stačí pouze jedna z těchto technologií. Z vlastních zkušeností víme, že při prvotních testováních chceme sbírat mnohem víc dat, než při závodě s vyladěným autem. Stejně tak se při testování formule nedostává dál než 150 m od depa, což se o závodech rozhodně říct nedá. Z toho tedy vyplývá, že nastanou situace, kdy bude stačit relativně krátký dosah, ale bude potřeba vyšší rychlost, což splňuje Wi-Fi. Na druhou stranu stejně tak nastanou situace, kdy budeme vyžadovat větší dosah, ale řidší datový tok nebude problém, neboli XBee a jemu podobné technologie.

Vybraným modulem je ESP32 od firmy Espressif. Důvodem pro výběr tohoto zařízení je možnost bezdrátové komunikace jak za využití klasických Wi-Fi standardů, tak i pomocí speciálního protokolu 802.11LR. Ten stejně jako velká část Wi-Fi využívá kmitočet 2,4 GHz. Při využití LR protokolu se dá komunikovat až na vzdálenost kolem 1 km, při rychlosti 250 Kb/s [14]. Stejně jako u XBee není možné pomocí tohoto protokolu komunikovat s klasickými PC přímo. Z toho důvodu bude třeba při komunikaci pomocí 802.11lr využít mezičlen, kterým bude další ESP32.

3.3 Přenos dat

Po zvolení přenosové technologie je třeba matematicky ověřit, zda je vůbec možné, abychom dosáhli požadované vzdálenosti při využití kmitočtu zvolené technologie. Pracovní kmitočet v naší aplikaci je 2,4 GHz. Stanoveným požadavkem je komunikace na 300 m při SAD faktoru minimálně 30 %.

3.3.1 Ztráty ve volném prostoru

Ztráty ve volném prostoru jsou jedním z důležitých parametrů při návrhu bezdrátové komunikace. Vyznačují poměr přijímaného výkonu ku přenášenému výkonu. Pro zjednodušení budeme uvažovat izotropní antény bez direktivity. Tím pádem se nám ztráty ve volném prostoru omezí na funkci 2 proměnných, vzdálenosti antén a frekvenci přenosu. Je důležité zmínit, že výpočet počítá s ideálním prostředím, kde neexistují překážky a není přítomno žádné rušení od jiných antén [15].

$$FSPL = \left(\frac{4\pi df}{c}\right)^2 \tag{3.2}$$

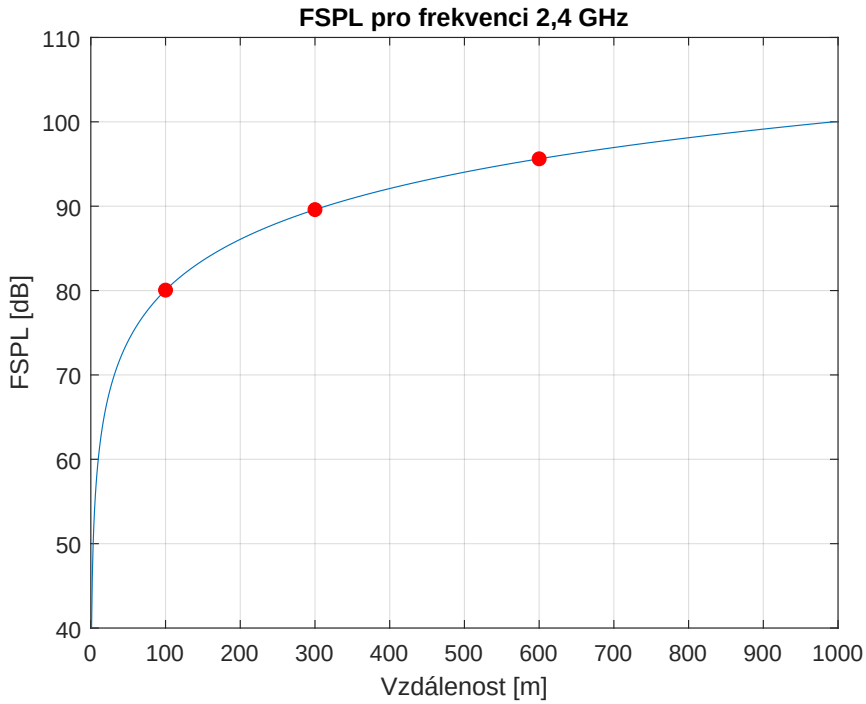
Rovnice (3.2) vyjadřuje ztráty výkonu při šíření volným prostorem, kde f [Hz] je vysílací frekvence, d [m] vzdálenost od vysílače a c [m/s] rychlost světla.

$$FSPL = 20\log_{10}(f) + 20\log_{10}(d) + 20\log_{10}\left(\frac{4\pi}{c}\right) \text{[dB]} \tag{3.3}$$

Rovnice (3.3) vychází z rovnice (3.2) a vyjadřuje ztráty výkonu při šíření vlny ve volném prostředí v decibelech. Vybraný systém bude pracovat s frekvencí 2,4 GHz. Na grafu 3.1 je zobrazen průběh FSPL v závislosti na vzdálenosti. V tabulce 3.1 jsou zobrazeny konkrétní hodnoty pro vyznačené body z grafu.

Tabulka 3.1: Hodnoty FSPL pro vybrané vzdálenosti

Frekvence [MHz]	$FSPL_{100m}$ [dB]	$FSPL_{300m}$ [dB]	$FSPL_{600m}$ [dB]
2400	80	89,6	95,6



Graf. 3.1: FSPL pro 2,4 GHz

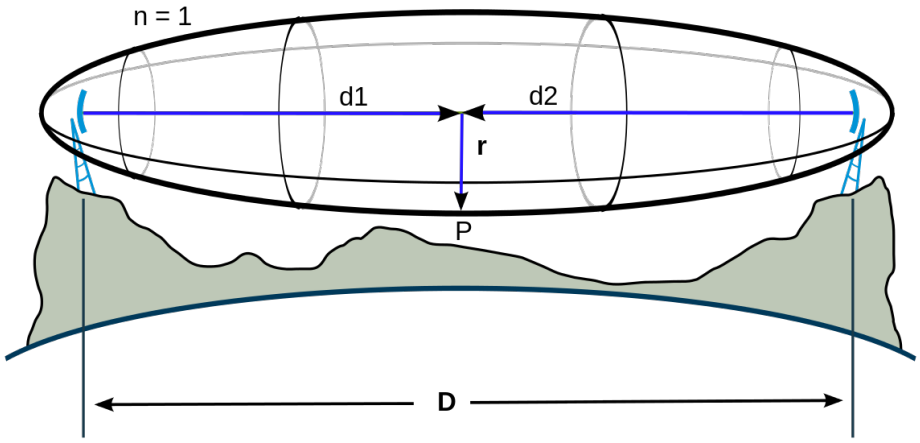
3.3.2 Fresnelova zóna

Při přenosu signálu od vysílače k přijímači se část vln šíří mimo osu přímé viditelnosti. Tyto vlny se pak mohou odrazit od překážet, což může vést ke správnému nasměrování vlny k přijímači. Takto odražená vlna dorazí k vysílači mimo fázi, což vede k destruktivní interferenci, pokud je fázový rozdíl polovinou periody.

Fresnelovy zóny definují množinu bodů v 3D prostoru, takže odraz od libovolného bodu v dané Fresnelově zóně způsobí fázový posun mezi $n-1$ a n polovinou periody oproti fázi vln, které se šíří přímočaře. Hranice těchto zón mají tvar elipsoidu, jehož ohniska jsou na vysílači a přijímači [16]. Tvar zóny je zobrazen na obrázku 3.5.

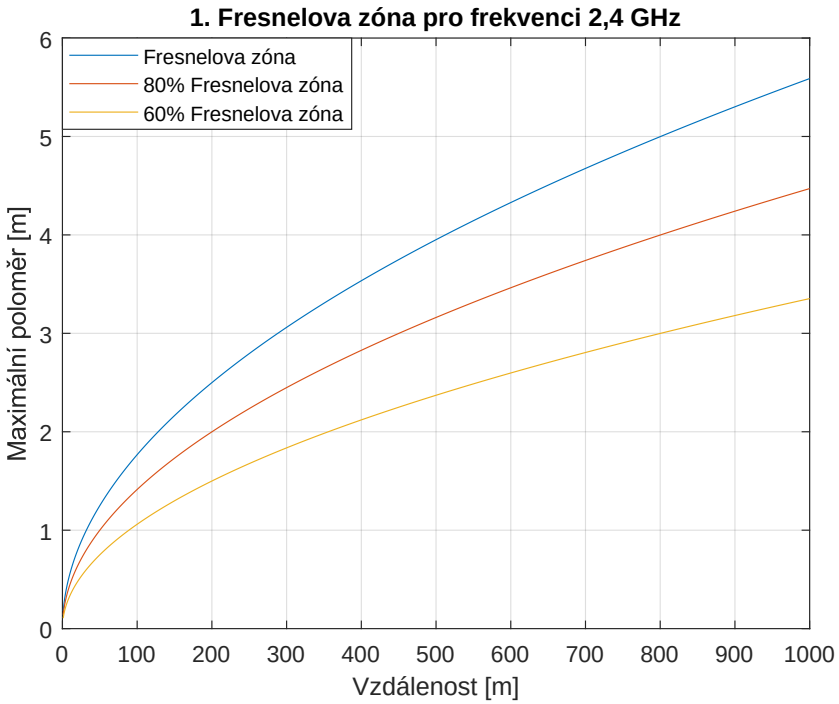
Z praktického hlediska je nejdůležitější první Fresnelova zóna, ve které se nachází většina vysílané energie. Vzorec (3.4) vyjadřuje maximální poloměr primární Fresnelovy zóny, kde $c\,[\mathrm{m/s}]$ je rychlost světla, $D\,[\mathrm{m}]$ je vzdálenost mezi vysílačem a přijímačem a $f\,[\mathrm{Hz}]$ je frekvence.

$$r_{\max} = \frac{1}{2}\sqrt{\frac{cD}{f}} \tag{3.4}$$



Obr. 3.5: Tvar Fresnelovy zóny [17]

Pravidlem je, že první Fresnelova zóna by měla být v ideálním případě bez překážek na 80 %. Úplným minimem je 60 % oblasti bez překážek [18]. V grafu 3.2 jsou zobrazeny průběhy maximálního poloměru první Fresnelovy zóny v závislosti na vzdálenosti vysílačů a právě stanovených pravidel.



Graf. 3.2: Primární Fresnelova zóna pro frekvenci 2,4 GHz

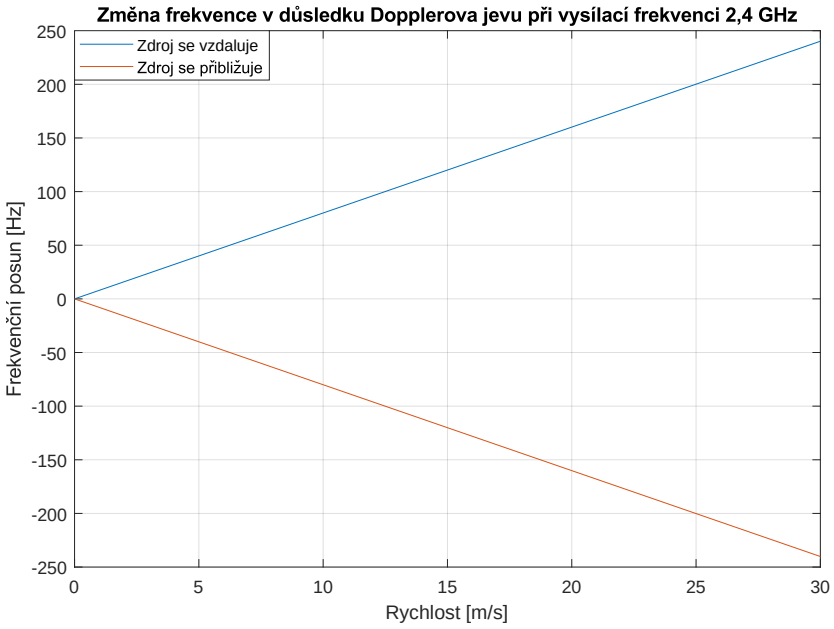
Z grafu vyplývá, že při přenášení signálu o frekvenci 2,4 GHz je maximální poloměr primární Fresnelovy zóny poměrně malý. To pro nás znamená, že bude stačit, když uživatel bude na lehce vyvýšeném místě oproti formuli. K zlepšení přenosu signálu, můžeme pomoci i umístěním antény na nejvyšší místo na formuli. Tím místem je hlavní oblouk rámu. Ovšem umístěním antény do tohoto prostoru se vysílač velmi přiblíží motoru. Bude tedy třeba vyzkoušet, jak velký vliv bude mít motor na signál, a to jak na signál bezdrátově odesílaný, tak na sběrnici CAN, která musí být k vysílači přivedena.

3.3.3 Dopplerův jev

Doplerův jev vyjadřuje míru posunutí frekvence, způsobenou relativním pohybem vysílače a přijímače. Přijatý signál se tedy bude jevit, jako by byl frekvenčně modulován. Tento jev se tedy bude objevovat v naší aplikaci, kde jedna strana bude v pohybu (formule) a druhá strana bude stacionární (uživatel). Při závodu dosahuje naše formule rychlosti až 30 m/s.

$$\Delta f = f_0 \left(1 - \frac{c}{c \pm v}\right) [\mathrm{Hz}] \tag{3.5}$$

Vzorec (3.5) zobrazuje vztah pro výpočet frekvenčního posunu, kde $f_0\,[\mathrm{Hz}]$ je vysílaná frekvence, $v\,[\mathrm{m/s}]$ je relativní rychlost přijímače vůči vysílači a $c\,[\mathrm{m/s}]$ je rychlost šíření vlny (rychlost světla).



Graf. 3.3: Změna frekvence v důsledku Dopplerova jevu

V grafu 3.3 je znázorněn průběh posunu frekvence v závislosti na rychlosti formule. Maximální frekvenční posunutí, kterého bychom měli v reálných podmínkách dosahovat je přibližně ±250 Hz. Klasicky používané krystaly mají stabilitu kmitočtu v jednotkách až desítkách ppm. Námi spočítané posunutí by odpovídalo desetinám ppm. Takto malé frekvenční změny by neměly způsobovat problémy s přijmutím dat na straně přijímače. Z toho důvodu můžeme vliv Dopplerova jevu zanedbat.

3.3.4 Výpočet maximální vzdálenosti přenosu signálu

Jedním z požadavků je, že systém dokáže při teoretickém výpočtu přenést data na vzdálenost 300 m a to při SAD faktoru minimálně 30 %.

SAD (Seriously Approximate Deviation) uvádí o kolik procent má systém větší výkon, než je potřeba. Obecně se požaduje minimálně 30 %. Tento faktor byl zaveden z důvodu velké nepřesnosti výpočtů. Ty totiž nepočítají s rušením a překážkami, kterým se v praxi téměř nedá vyhnout [19].

Hlavním prvkem vysílače je čip ESP32-WROOM-S, který při využití protokolu 802.11b a 802.11LR může mít vysílací výkon až 20 dbm a citlivost −98 dBm [14]. Navíc k němu bude připojena 2,5dB anténa.

Při první variantě bude na uživatelské straně libovolný notebook, který by měl být schopen přijmout malé množství dat i při citlivosti kolem −70 dBm. A jehož vysílací výkon bude přibližně 10 dBm [20]. Zesílen bude všesměrovou anténou se ziskem 6 dB.

Při druhé variantě bude na straně uživatele zařízení se stejnými vlastnostmi jako má zařízení ve formuli.

$$P_{TX} = S_{TX} + G_{TX} - L_{TX}[\text{dB}] \tag{3.6}$$

Rovnice (3.6) uvádí vzorec pro výpočet celkového výkonu vysílací stanice P_{TX} [dB]. Ten se vypočítá jako součet výkonu vysílače S_{TX} [dBm] se ziskem antény vysílače G_{TX} [dB] minus ztráta ve vedení na straně vysílače L_{TX} [dB].

$$P_{RX} = -S_{RX} + G_{RX} - L_{RX}[\text{dB}] \tag{3.7}$$

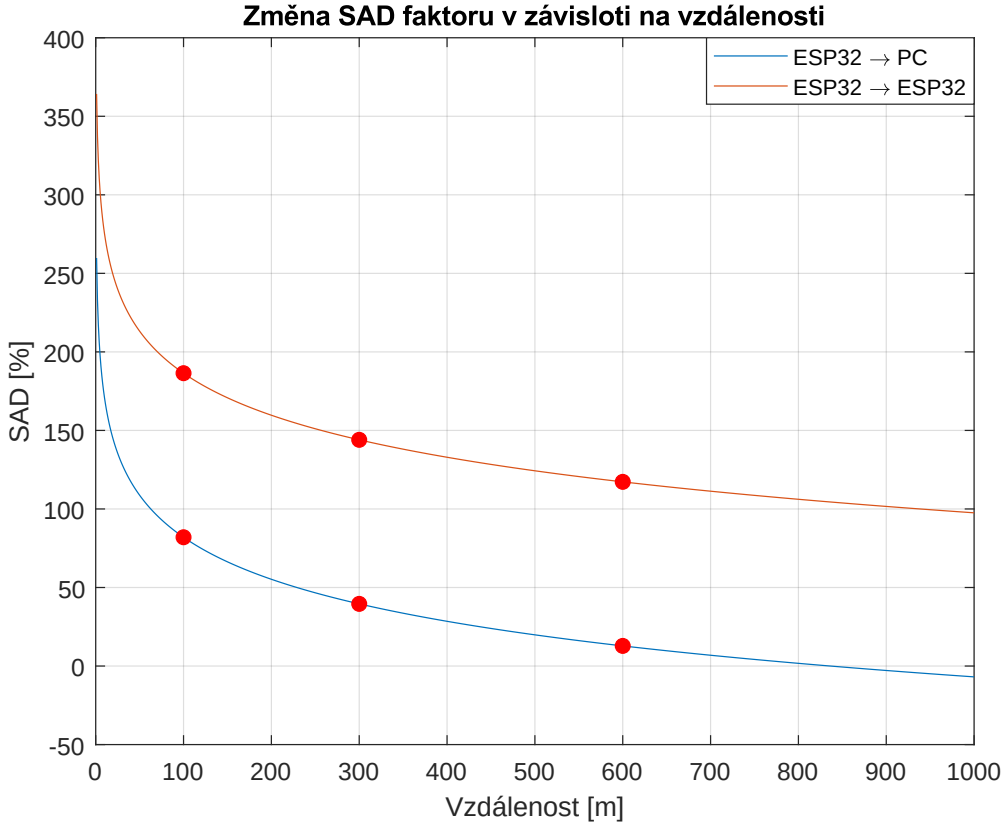
Obdobně k tomu se podle vzorce (3.7) spočítá celkový výkon přijímací stanice P_{RX} [dB]. Vypočítá se obdobně jako výkon vysílací stanice. Za S_{RX} [dBm] se dosadí citlivost přijímače, G_{RX} [dB] je zisk antény přijímače a L_{RX} [dB] jsou ztráty ve vedení na straně přijímače.

Pokud bude zařízení s ESP32 vysílat, tak jeho celkový výkon bude $P_{TX} = 22,5\,\text{dB}$. V opačném případě bude celková výkonová citlivost $P_{RX} = 99,5\,\text{dB}$. Pro notebook pak vychází $P_{TX} = 16\,\text{dB}$ a $P_{RX} = 76\,\text{dB}$.

Při výpočtech byly uvažovány nulové ztráty ve vedení, protože je výrobce neuvedl. Dá se tedy předpokládat, že ztráty byly již zakomponovány do celkového zisku antény.

$$SAD = \frac{P_{TX} + P_{RX} - FSPL}{P_{TX}} \times 100[\%] \tag{3.8}$$

Podle vzorce (3.8) se spočítá SAD faktor, neboli kolika procentní je přebytek energie vysílače. Pro připomenutí $FSPL$ [dB] je ztráta ve volném prostředí. SAD faktor se vypočítá pro dvě varianty. V první variantě je ESP32 vysílačem a přijímačem je notebook s přidanou anténou. Ve druhé variantě je na obou stranách komunikace zařízení založené na ESP32. V grafu 3.4 je zobrazen pokles SAD v závislosti na vzdálenosti vysílače a přijímače.



Graf. 3.4: Změna SAD v závislosti na vzdálenosti

Tabulka 3.2: Hodnoty SAD faktoru pro vybrané vzdálenosti

Varianta	$SAD_{100\,m}$ [%]	$SAD_{300\,m}$ [%]	$SAD_{600\,m}$ [%]
ESP32 → PC	82	39,6	12,8
ESP32 → ESP32	186,4	144	117,3

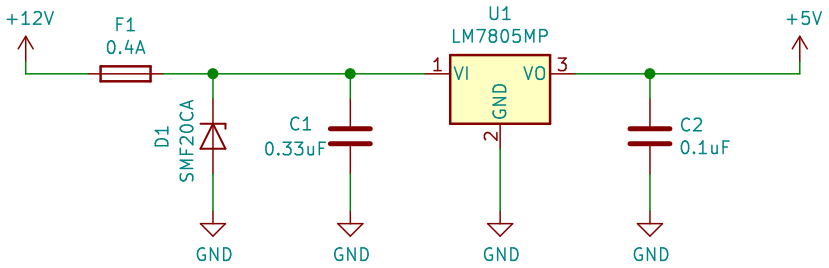
V tabulce 3.2 jsou zobrazeny konkrétní hodnoty pro vyznačené body z grafu. Z tabulky vyplývá, že stanovená hranice 300 m při 30% SAD faktoru je splněna v obou variantách. Při komunikace mezi dvojicí ESP32 se dá dokonce předpokládat stabilní spojení i na 600 a více metrů.

3.4 Hardwarový návrh

Nyní, když je určeno, jaká data se budou přenášet, a zároveň jakým způsobem se budou přenášet, tak je potřeba navrhnout mezičlen, který dokáže pracovat s CAN sběrnicí a zároveň bude schopný komunikovat s koncovým zařízením pomocí Wi-Fi. Právě možnost komunikace pomocí Wi-Fi a zároveň podpora CAN sběrnice je jedním z hlavních důvodů, proč byl vybrán ESP32 od firmy Espressif. V následujících sekcích jsou popsány jednotlivé části navržené desky.

3.4.1 Power Suply 5 V

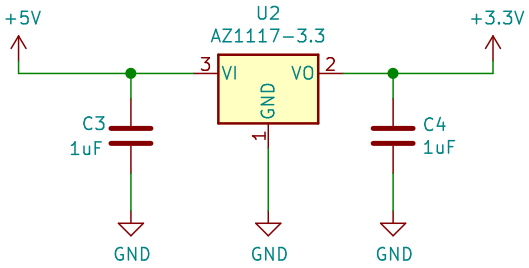
Jakožto primární zdroj napětí je zde zvolen LM7805, který ze vstupního napětí 12 V vytvoří 5 V na výstupu při maximálním odběru 1,5 A. Vstup je chráněn vratnou pojistkou s vypínacím proudem 0,4 A a transilem chránícím před elektrickými výboji až do 30 kV. Velikost kondenzátorů byla zvolena podle hodnot doporučeného zapojení, uvedeného v technické dokumentaci zdroje. Schéma zapojení je na obrázku 3.6.



Obr. 3.6: Zapojení zdroje napájení 5 V

3.4.2 Power Suply 3,3 V

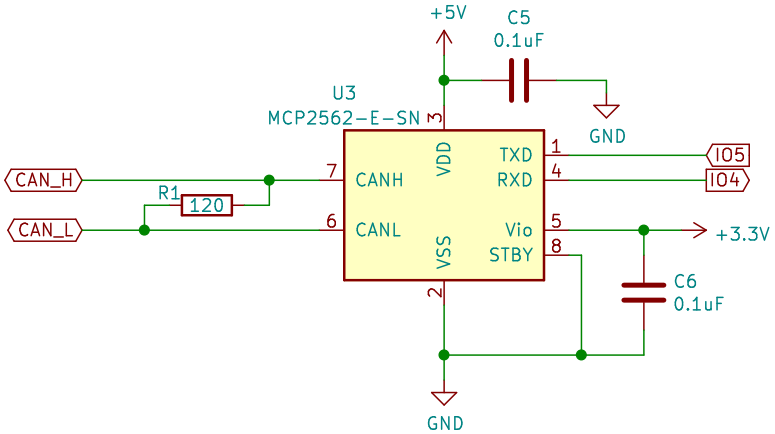
Druhým zdrojem napětí je nízkoúbytkový regulátor AZ1117 s výstupním napětím 3,3 V a maximálním výstupním proudem 1,3 A. Tento zdroj slouží pro napájení ESP32. Velikost kondenzátorů byla opět zvolena podle technické dokumentace zdroje. Schéma zapojení je na obrázku 3.7.



Obr. 3.7: Zapojení zdroje napájení 3,3 V

3.4.3 CAN bus Transceiver

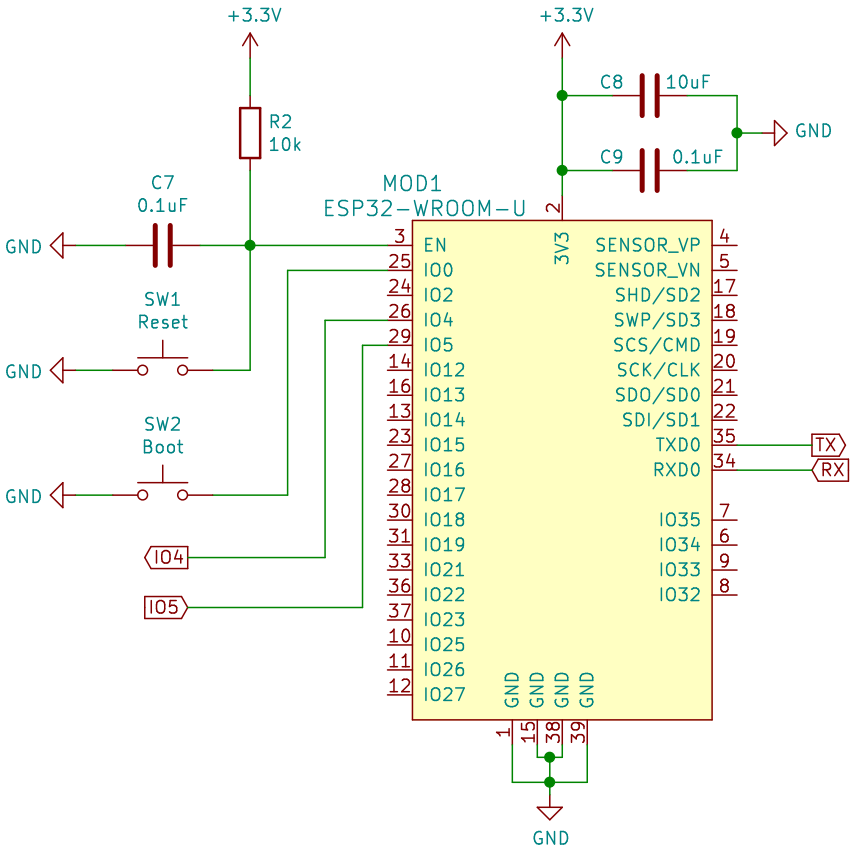
Proto, aby mohlo ESP32 komunikovat s formulí pomocí CAN sběrnice, je nutný mezičlánek. Tím mezičlánkem je čip MCP2562, který funguje jako rozhraní mezi protokolem CAN a fyzickou vrstvou sběrnice CAN. Piny CANH a CANL jsou určeny k přímému zapojení sběrnice CAN z formule. Piny TXD a RXD se naopak připojí na určené piny ESP32, pomocí kterých bude probíhat komunikace podle CAN protokolu. Pin VIO, určuje referenční napětí pro piny TXD, RXD a STBY. Je napájen 3,3 V, protože toto napětí vyžadují piny ESP32. Pin STBY přepíná mezi *Standby mode* a *Normal mode*. Při aktivním STBY je vypnuto odesílání dat a vysokorychlostní příjem, proto je tento pin uzemněn, tedy neaktivní. Schéma zapojení je na obrázku 3.8.



Obr. 3.8: Zapojení vysílače a přijímače CAN sběrnice

3.4.4 ESP32

ESP32-WROOM-U je takzvaný SoC, který kromě mikrokontroléru obsahuje i další klíčové komponenty jako jsou krystal nebo anténa, které umožňují okamžitou a jednoduchou integraci do koncového produktu. Jeho zapojení do obvodu, včetně hodnot součástek, je převzato z technické dokumentace [14]. Čip se bude programovat pomocí externího programátoru přes komunikační piny TXD0 a RXD0. CAN komunikace je uskutečňována pomocí pinů IO4 a IO5, kde IO4 slouží pro příjem dat a IO5 pro odesílání dat. Tyto piny byly vybrány, protože podporují CAN komunikaci. Schéma zapojení je na obrázku 3.9.

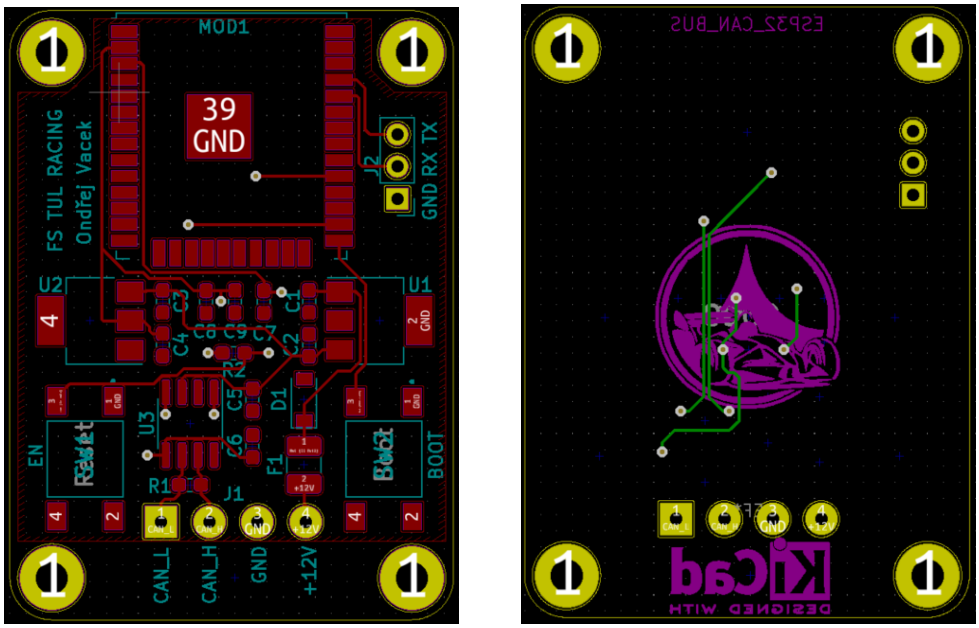


Obr. 3.9: Zapojení ESP32-WROOM-U

3.4.5 Plošný spoj

Na základě schémat výše byla navržena deska plošných spojů. Všechny komponenty se povedlo vměstnat na desku o velikosti 35×48 mm. Kromě součástek obsažených ve schématu se na desku přidaly i 4 díry, které mohou sloužit k uchycení desky do krabičky. Kompletní seznam, včetně pouzder a cen jednotlivých komponent, je v příloze B. Návrh plošného spoje je zobrazen na obrázku 3.10.

Signálové trasy přenáší zanedbatelné proudy, takže u nich není potřeba se zabývat šířkou spoje. Napájecí trasy mohou při běžném provozu přenášet proudy kolem 1 A. Doporučená šířka trasy při 1 oz mědi a teplotním vzrůstu 10 °C pro 1 A je 0,25 mm [21]. Nakonec byly i signálové trasy upraveny na šířku 0,25 mm, kvůli požadavkům výrobce.



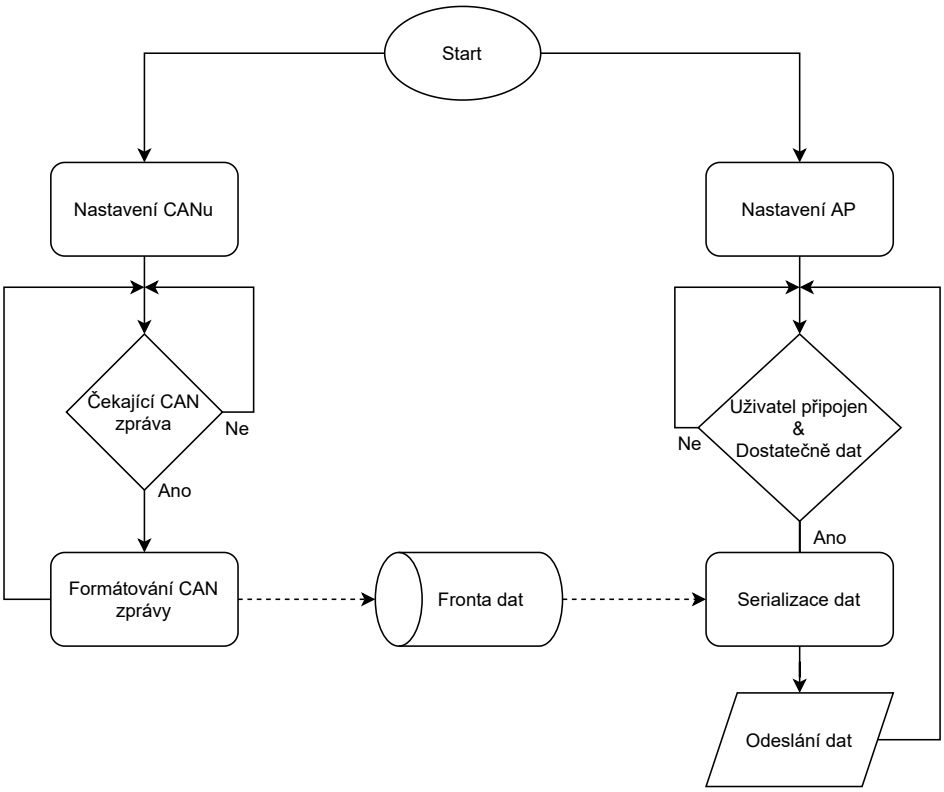
(a) Přední strana (b) Zadní strana

Obr. 3.10: Plošný spoj navržené desky

3.5 Program pro vysílač

Ovládací program vysílače je napsaný v jazyce C++ za podpory arduino frameworku a FreeRTOS. Tento program běží na zařízení, které je umístěno ve formuli. Jeho činnost je rozdělena na dvě úlohy. První z nich je získávání a formátování dat ze sběrnice CAN. Druhá úloha je vytvoření přístupového bodu a odesílání dat uživateli. Jelikož je ESP32 dvoujádrový mikroprocesor, tak se této výhody využilo a výše zmíněné úlohy běží každá na svém jádře. Propojením těchto úloh je zařízení schopné přečíst informaci z formule a následně ji odeslat připojenému uživateli v požadovaném formátu.

Následující podkapitoly jsou věnovány detailnějšímu popisu funkcionalit dílčích částí kódu. Pro názorné pochopení celého programu je níže zobrazený vývojový diagram 3.11, který jasně a jednoduše ukazuje jak na sebe jednotlivé bloky kódu navazují. Kompletní zdrojový kód pro vysílač, včetně náležitých doplňkových souborů je dostupný na GitHubu [22].



Obr. 3.11: Vývojový diagram programu pro vysílač

3.5.1 Nastavení CANu

Po spuštění úlohy obstarávající chod CAN komunikace se jako první provede nastavení. Jako první se nastaví rychlost přenosu. Připojená sběrnice CAN komunikuje na rychlosti 1 Mb/s takže se program nastaví na tuto rychlost. Následně se nastaví na jaké piny je fyzicky připojená komunikace. V případě výše navržené desky to jsou piny 4 a 5. Jako poslední věc před inicializací se ještě vytvoří fronta pro příchozí zprávy. Její velikost je nastavena na 100 záznamů struktury typu CAN_frame_t. Ta obsahuje informace jako je velikost, ID a data přijaté zprávy. Nakonec se spustí inicializační sekvence, kde se podle uživatelských definic nastaví CAN komunikace. V případě selhání se systém automaticky restartuje a pokusí se vše doposud provedené nastavit znovu.

3.5.2 Získání a formátování CAN dat

Po nastavení a inicializaci CANu přejde úloha do nekonečné smyčky. Úlohou této smyčky je získat, formátovat a nakonec přeposlat upravenou CAN zprávu do fronty dat čekajících na odeslání uživateli. Níže je ukázka kódu z této úlohy, která funguje následovně. Pokud je alespoň jeden CAN rámec čekající ve frontě, tak si ho program načte pro další zpracování. Z již načteného CAN rámce program vyčte o jaký typ zprávy se jedná, jestli je to standardní a nebo rozšířený formát. Stejně tak zjišťuje zdali se nejedná o RTR (Remote Transmission Request). Pokud je rámec ve stan-

dardním formátu a zároveň se nejedná o RTR, tak jsou splněny obě dvě podmínky, určující zdali je CAN rámec chtěný, či nikoliv. V tom případě se vyčtený rámec zkonvertuje do formátu odpovídající struktuře *CanMessage* a zařadí se do datové fronty pokud v ní je volný slot.

Při konverzi přijatého CAN rámce do formátu struktury *CanMessage* se data konvertují do formátu, který očekává uživatel. Příprava spočívá v tom, že uživatel očekává zprávu o délce 12 bajtů, kde první 4 bajty jsou ID a zbylých 8 bajtů je pro vlastní data. Zbylé informace obsažené v přijatém CAN rámci se zahodí.

```
1  CAN_frame_t rx_frame;
2  CanMessage canMsg;
3
4  if(xQueueReceive(CAN_cfg.rx_queue, &rx_frame, 10*portTICK_PERIOD_MS) == pdPASS){
5    if(rx_frame.FIR.B.FF == CAN_frame_std && rx_frame.FIR.B.RTR != CAN_RTR){
6      canMsg = convertCanFrameToCanMessage(rx_frame);
7      if(xQueueSend( messageQueue, (void *) &canMsg, 100*portTICK_PERIOD_MS) !=
       ↪  pdPASS){
8        Serial.println("Failed to send Can message to queue");
9      }
10   }
11 }
```

Zdrojový kód 3.1: Ukázka vyčtení dat z CANu

3.5.3 Nastavení AP

Po spuštění úlohy obstarávající chod AP se jako první provede nastavení. V nastavení se nakonfiguruje komunikační protokol, ten může být 802.11b/g/n nebo speciální 802.11lr. Ten použijeme v případě, že nechceme komunikovat přímo s uživatelským PC, ale s mezičlenem, díky čemuž se razantně prodlouží vzdálenost, na kterou jsme schopni komunikovat. Dále se nastaví vysílací výkon. Ve většině případů bude nastaven na maximální hodnotu, kterou ESP32 dovoluje (20 dBm). Případně na maximální hodnotu, kterou dovoluje legislativa země, ve které bude zařízení spuštěno.

Na úrovni transportní vrstvy se používá TCP a to i přes to, že logicky se pro tuto aplikaci hodí spíše UDP vzhledem k tomu, že se snažíme o práci v reálném čase. Důvodem je, že ve formuli je na CANu momentálně definováno pouze pár desítek zpráv, tudíž není zas tak důležité získat větší datovou propustnost na úkor spolehlivosti. V případě potřeby se dá jednoduchým zásahem do kódu změnit transportní metoda na UDP.

Dál se již nastaví parametry tykající se přímo přístupového bodu jako jsou identifikátor, přístupové heslo, lokální IP adresa, gateway IP a maska sítě. Nakonec se spustí nakonfigurovaný přístupový bod. V případě selhání při konfiguraci jakéhokoliv parametru, či při samotném spuštění se zařízení samo restartuje, čímž spustí celou sekvenci znovu.

3.5.4 Komunikace s připojeným zařízením

Po úspěšném nastavení AP přejde úloha do nekonečné smyčky. Úlohou této smyčky je přijmout CAN data, serializovat je a nakonec odeslat paket uživateli. Níže je ukázka kódu z této úlohy, která funguje následovně. Po dobu, kdy je uživatel připojen, se zjišťuje, zdali ve frontě nečeká dostatečný počet zpráv. Pokud je dostatečný počet čekajících zpráv, tak se pro tyto zprávy alokuje paměť. Následně se začnou zprávy z fronty po jedné přijímat. Jejich obsah se překopíruje do alokované paměti tak, že jednotlivé zprávy na sebe v paměti navazují, neboli se serializují. Následně se tento blok paměti obsahující serializoané zprávy odešle připojenému uživateli.

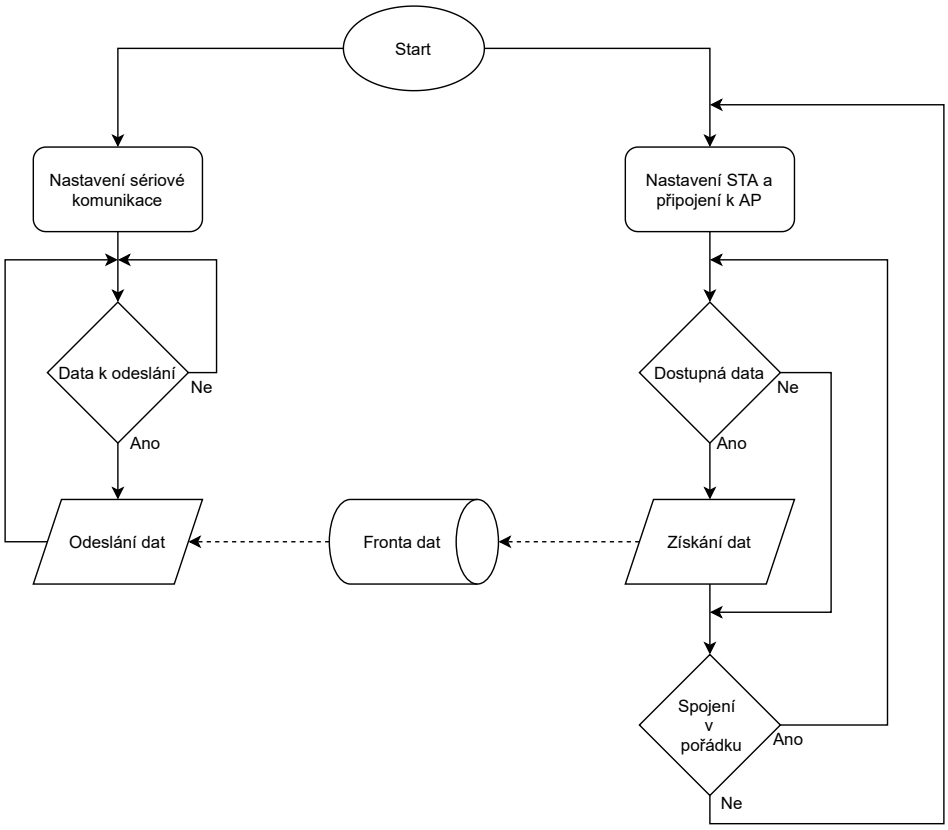
```
1  while (client.connected()) {
2    if( uxQueueMessagesWaiting(messageQueue) > msg_send_count){
3      uint8_t dataToSend[msg_send_count * msg_size];
4      CanMessage canMsg;
5      for(int i = 0; i < msg_send_count; i++){
6        if(xQueueReceive(messageQueue, &(canMsg), (TickType_t) 0) == pdPASS){
7          memcpy(dataToSend + msg_size * i, canMsg.msg, msg_size);
8        }
9      }
10     client.write(dataToSend, msg_send_count * msg_size);
11   }
12 }
```

Zdrojový kód 3.2: Ukázka odeslání dat uživateli pomocí Wi-Fi

3.6 Program pro přijímač

Ovládací program přijímače běží na zařízení, které má uživatel připojené do svého PC pomocí USB. Momentálně je jako přijímací zařízení použita vývojová deska ESP32, která obsahuje všechny potřebné periférie. Toto zařízení je použito v případě, že uživatel chce komunikovat s formulí na větší vzdálenost, než je schopen za použití přímého Wi-Fi připojení k formuli. Činnost zařízení je opět rozdělena na dvě úlohy, kde každá úloha běží na vlastním jádře. Funkce první úlohy je navázat/udržet spojení s formulí a přijímat data. Druhá úloha je velmi jednoduchá, zde se pouze vezmou přijatá data a odešlou se po sériové lince do počítače.

Pro názorné pochopení celého programu je níže zobrazený vývojový diagram 3.12, který jasně a jednoduše ukazuje, jak na sebe navazují jednotlivé bloky kódu. Kompletní zdrojový kód pro přijímač, včetně náležitých doplňkových souborů, je taktéž dostupný na GitHubu [22].



Obr. 3.12: Vývojový diagram programu pro přijímač

3.6.1 Sériová linka

Při spuštění zařízení se provede jednoduché nastavení sériové linky. Jedná se o přenosovou rychlost, počet datových bitů, paritu a počet stop bitů. Konkretně je komunikace nastavena na rychlost 256 kBd s datovým formátem 8N1 (8 datových bitů bez parity, 1 stop bit).

Vlastní odeslání dat je pak již velmi jednoduché. Jakmile je ve frontě dat dostupná zpráva, tak ji program přepošle koncovému zařízení po sériové lince.

3.6.2 Wi-Fi komunikace s formulí

K tomu, aby byla možná komunikace s formulí, je třeba provést potřebné nastavení. Zařízení se nastaví do módu STA, díky čemuž se z něj stane klient schopný komunikace s AP. Dále je třeba nastavit schodný komunikační standard, v tomto případě se

jedná o 802.11 LR. Následně se zařízení připojí k AP pomocí identifikátoru a hesla. Po úspěšném spojení se vytvoří komunikační kanál s konkrétní IP adresou na konkrétním portu. V případě jakéhokoliv selhání při nastavení komunikace se zařízení automaticky restartuje, čímž celý proces spustí znovu.

Po úspěšném vytvoření komunikačního kanálu začne zařízení zjišťovat zdali jsou dostupná data, jejichž velikost odpovídá alespoň jedné CAN zprávě (12 bajtů). Jakmile taková data dostupná jsou, tak je program rovnou přepošle do fronty dat čekající na odeslání po sériové lince.

V rámci zajištění spolehlivosti disponuje zařízení diagnostikou spojení. V případě, že program detekuje přerušení spojení, tak opět přejde do fáze, kde se znovu nastaví celá komunikace a pokusí se toto spojení obnovit. V případě selhání této opravy dojde k restartu celého zařízení.

```
1  xTaskCreatePinnedToCore(
2      wifiHandler,      /* Function to implement the task */
3      "WiFiHandler",    /* Name of the task */
4      10000,            /* Stack size in words */
5      NULL,             /* Task input parameter */
6      0,                /* Priority of the task */
7      &wifiTask,        /* Task handle. */
8      0);               /* Core where task should run */
```

Zdrojový kód 3.3: Ukázka vytvoření úlohy běžící na dedikovaném jádře

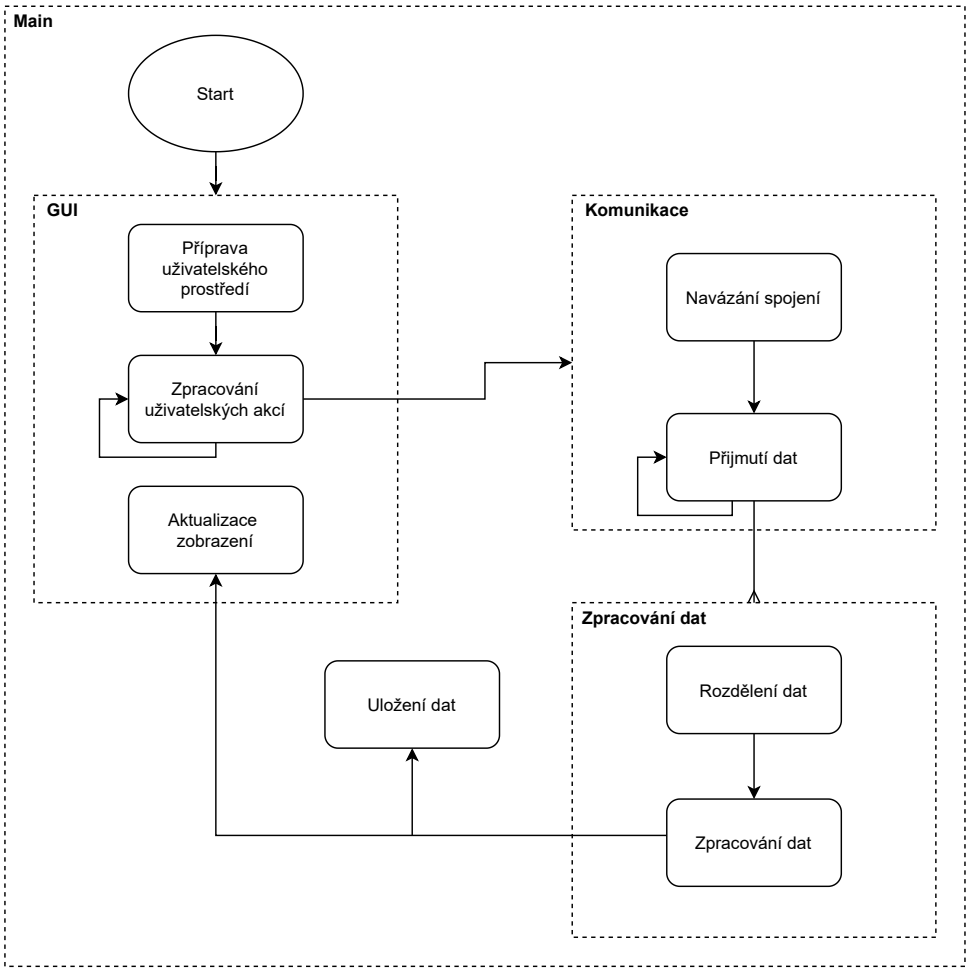
4 Aplikační část

V této části práce se budeme zabývat aplikací pro zpracovaní a vizualizaci získaných dat. Aplikace s názvem „CANReader“ byla vytvořena v jazyce Python 3.x za využití frameworku Qt. Jako API mezi Qt a Pythonem byl využit modul PyQt5.

Při tvorbě aplikace byl brán zřetel na dlouhodobou udržitelnost kódu kvůli potřebě aplikaci předat do správy členům teamu. Pojmem udržitelnost je myšlena aplikace, verzovaná pomocí GITu, ke které jsou vytvořeny sady testů včetně nastaveného CI. Kód má patřičnou dokumentaci a obsahuje logger, který by měl pomoci při identifikaci problému v produkční verzi programu. V sekci 4.1 jsou tyto jednotlivé techniky a nástroje probrány podrobněji.

Nyní už k samotné funkci programu. Na obrázku 4.1 je obecný vývojový diagram funkce programu. Šipky vedoucí z jednoho bloku do druhého značí přímou interakci těchto bloků. Kdežto šipky vedoucí z bloku na hranu oblasti označené čárkovanou čárou značí vyvolání daného podprogramu. Jak již z předchozího popisu vyplývá, tak se jedná o vícevláknový program, který obsahuje 2 stálá vlákna „Main“ a „GUI“. Vlákna „Komunikace“ a „Zpracování dat“ jsou vytvářena situačně na základě uživatelem vyvozené akce a následně přijmutím dat z formule. Na diagramu je vidět, že vazba mezi blokem přijmutí dat a vytvořením vlákna pro zpracování dat je one-to-many. Tato vlastnost je důležitá z důvodu snížení prodlevy mezi odesláním dat na straně formule a konečným zobrazením dat na straně aplikace.

Hlavní program, v diagramu označen „Main“, zajišťuje komunikaci mezi jednotlivými podprogramy, vedle toho má také na starost ukládání příchozích dat. Druhý podprogram, v diagramu označený „GUI“, má na starost kompletní uživatelské prostředí. Tedy zobrazení základního GUI, vypořádání se s uživatelskými požadavky a aktualizaci na základě poskytnutých dat. Komunikační podprogram, v diagramu „Komunikace“, zajišťuje komunikaci s formulí na základě uživatelem stanovených kritérií. A nakonec poslední podprogram, v diagramu uvedený jako „Zpracování dat“, zajišťuje filtraci, zpracování a rozřazení příchozích dat. V následujících podsekcích se budeme jednotlivým podprogramům věnovat blíže.



Obr. 4.1: Zjednodušený model programu pro aplikační část

4.1 Udržitelnost programu

4.1.1 Správa verzí

Správa verzí je systém zaznamenávající změny souborů v čase tak, aby se uživatel mohl kdykoliv navrátit k jedné ze starších verzí. Kromě návratu projektu do předchozího stavu umožní porovnávat změny provedené v průběhu času, zjistit, co nyní možná způsobuje problémy, kdo a kdy vytvořil diskutabilní část a mnoho dalšího. Používáním systému pro správu verzí získáme snadné řešení v případě problému se ztrátou nebo nechtěnou manipulací s daty. Všechny tyto výhody navíc získáme jen při velmi malém zvýšení režií [23].

V našem konkrétním případě se využil Git, což je distribuovaný systém správy verzí. Jako správce se použil GitHub, což je cloudová služba dovolující uživateli přehledně a jednoduše spravovat repozitáře Git. Kromě této služby GitHub také disponuje rozšířenějšími nástroji, díky kterým může uživatel spravovat celý aplikační cyklus od návrhu řešení, přes testování až po nasazení do produkce.

4.1.2 Testování

Jednou z hlavních zásad udržitelnosti kódu je testování. Díky testům jsou vývojáři schopni zajistit správnou funkcionalitu jednotlivých částí aplikace. Toto se hodí zejména u rozsáhlejších programů, kde není jednoduché poznat, jak a jestli nová funkcionalita ovlivní ty staré. Jednou z doporučených metod vývoje aplikací je takzvaný Test-driven Development, neboli TDD. Což je proces vývoje softwaru, který je založen na převedení požadavků na testy ještě před tím než se implementuje samotné řešení. Po celou dobu vývoje se pak sleduje chování softwaru opakovaným testováním proti všem vytvořeným testům.

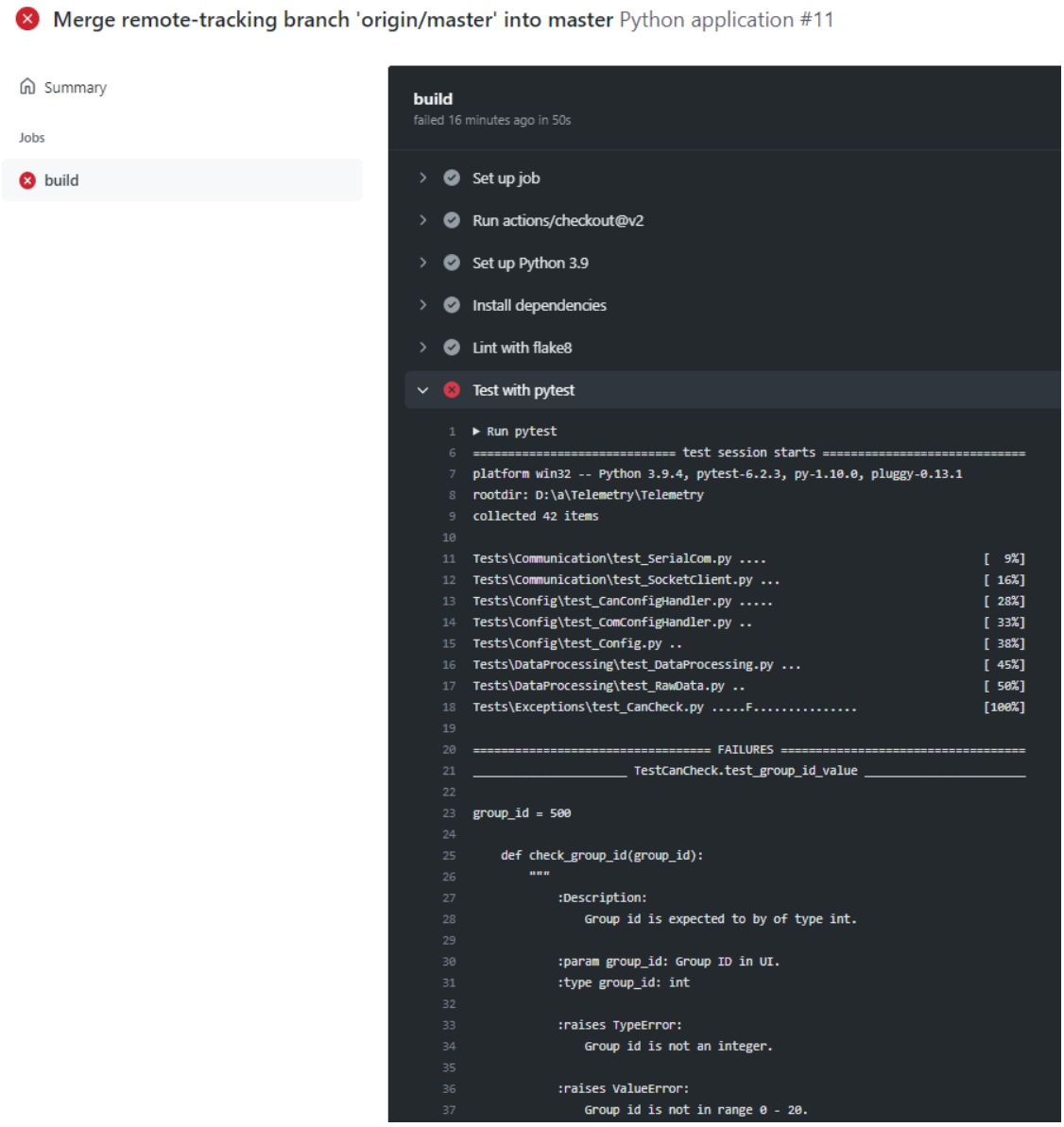
V našem případě není kompletní program zaštítěn testy. Balíček testů obsahuje 42 testovacích scénářů, které mají za úkol ohlídat spolehlivost klíčových částí programu. Především se jedná o testování omezení uživatelských vstupů a testy ověřující správnou funkcionalitu při zpracování dat. Součástí nejsou například testy, které by přímo ověřovaly jednotlivé prvky v GUI. Níže je zobrazen ukázkový testovací případ 4.1, který ověřuje funkcionalitu dekódování na základě vstupních dat a seznamu datových konfigurací.

```
1  def test_data_decode(self):
2      can_id = "603"
3      can_data = "0000000011111111000000001111111100000000111111110000000011111111"
4      config_list = [CanDataConfig(1, 1, 1, 1, "Temperature", "°C", "603", 0, 16,
       ↪  0.5, 0, "B"),
5                     CanDataConfig(2, 2, 2, 2, "Oil temperature", "°C", "603", 16,
                      ↪  16, 0.25, 0, "B"),
6                     CanDataConfig(2, 2, 2, 2, "Speed", "km/hod", "600", 0, 16, 1,
                      ↪  0, "B")]
7      results = [255/2, 255/4]
8
9      data_processor = DataProcessing(can_id, can_data, config_list)
10     data_points = data_processor.data_decode()
11
12     self.assertEqual(len(data_points), 2)
13     for i, data_point in enumerate(data_points):
14         self.assertAlmostEqual(results[i], data_point.value, 1)
```

Zdrojový kód 4.1: Ukázka testování kódu

4.1.3 Kontinuální integrace

Kontinuální integrace, neboli CI, automatizuje vytváření a testování softwaru. Umožňuje nám tedy otestovat naši aplikaci v různých prostředích (Windows, Linux, MacOS, aj.). Kromě vyhodnocení testů nám ale také může například kontrolovat i to, zdali dodržujeme zásady psaní čistého kódu.



Obr. 4.2: Ukázka zachycení chyby v kódu díky kontinuální integraci

GitHub nabízí poskytnutí výpočetního výkonu zdarma za účelem CI pro veřejně vedené repositáře. Díky tomu jsme mohli tuto užitečnou funkcionalitu implementovat. Při každém nahrání dat (push) do repositáře nebo při sloučení (merge) dvou větví se automaticky spustí CI pipeline. Na obrázku 4.2 je ukázka CI pipeline, která skončila neúspěšně z důvodu neprojití všemi testy. Ta v izolovaném prostředí Windows 10 nainstaluje Python i se všemi potřebnými závislostmi. Následně vyhodnotí kvalitu napsaného kódu. A nakonec otestuje program kompletní sadou vytvořených testů.

4.1.4 Dokumentace

Dokumentace je velmi důležitá součást při vývoji programu. Dokumentace vysvětluje, jak daný program funguje nebo jak ho použít. Pro jeden program může existovat i více dokumentací v závislosti podle toho na koho jsou dané dokumentace mířeny. Například dokumentace pro koncového uživatele bude pravděpodobně obsahovat jiné informace než dokumentace určená pro vývojáře.

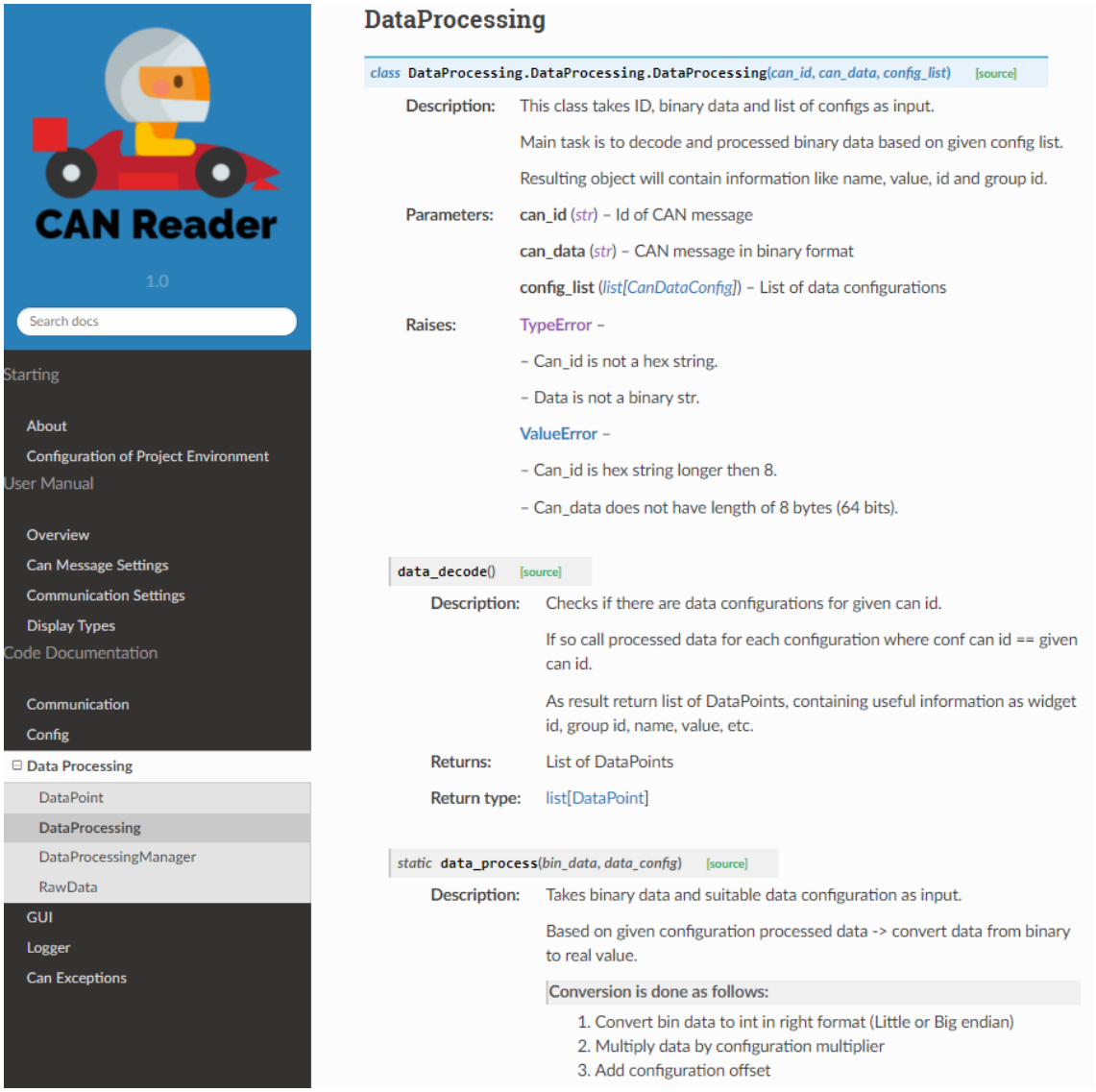
Ke zjednodušení tvorby dokumentace se používají generátory, které na základě šablon vytvoří přehledné webové stránky. V tomto případě používám python generátor Sphinx [24]. Obsah do těchto stránek se může brát automaticky přímo z kódu. Část kódu označující dokumentaci se nazývá „doc string“. V této části se popisuje očekávaná funkcionalita, vstupy, výstupy atd. Části dokumentace nezabývající se přímo popisem kódu se již píší do samostatných souborů. V našem případě se tvoří soubory ve formátu .rst, neboli reStructuredText. Což je jednoduchý značkovací jazyk navržený tak, aby byl jednoduše čitelný a zároveň i zpracovatelný pro programy sloužící k automatickému generování dokumentace.

Vzhledem k tomu, že se v našem případě nejedná o komerčně určenou aplikaci, tak není důvod tvořit více dokumentací. Vytvořená dokumentace má tedy všechny důležité informace na jednom místě a čtenář si musí sám najít tu část, která ho zajímá. Odkaz na online dokumentaci je dostupný v seznamu použité literatury [25]. Pro usnadnění vyhledávání má dokumentace 3 hlavní části.

Sekce „Starting“ obsahuje základní informace o aplikaci, jako pro koho je aplikace určena, jaké jsou její obecné vlastnosti, kdo ji vytvořil, a jak se orientovat v dokumentaci. Vedle toho obsahuje ještě návod na konfiguraci projektu ze zdrojového kódu.

Sekce „User Manual“ je určena především pro koncové uživatele. Obsahuje již podrobnější informace o tom, čeho je aplikace schopná, jak nastavit komunikaci se vzdáleným zařízením a jak nastavit jednotlivé CAN zprávy.

Sekce „Code Documentation“ obsahuje velmi podrobnou dokumentaci celého kódu. Ke každé použité třídě a metodě je vytvořený „doc string“, který obsahuje popis funkcionality, jaké jsou očekávané vstupy a výstupy, jaké chyby umí zachytit a další. Tento popis umožní dalším programátorům pochopit záměr a tím i mnohem snadněji implementovat libovolné modifikace. Na obrázku 4.3 je ukázka vytvořené dokumentace, konkrétně se jedná o popis třídy DataProcessing a jejích metod.



Obr. 4.3: Ukázka z vytvořené dokumentace

4.1.5 Logování

Poslední technika, o které se budeme bavit v rámci udržitelnosti programu je logování. Logování je nástroj zaznamenávající činnosti programu. To se hodí zejména při potřebě identifikovat příčinu problému v již vydané aplikaci, kde není dostupné vývojové prostředí s konzolí, kam by se mohly vypisovat vnitřní stavy. Programátor se může rozhodnout s jakou důležitostí budou jednotlivé zprávy zaznamenávány. Důležitost záznamů může nabývat jednu z následujících úrovní: FATAL, ERROR, WARN, INFO, DEBUG a nebo TRACE.

FATAL Představuje skutečně katastrofické situace. V takovéto situaci se aplikace většinou sama ukončí, aby předešla poškození nebo jiným závažným problémům. V praxi to může znamenat to, že bude třeba, aby někdo, i třeba uprostřed noci, vstal a šel okamžitě daný problém vyřešit.

ERROR Použije se v případě chyby, některé ze zásadních komponent. Na rozdíl od FATAL se aplikace sama nepřeruší. V praxi se může jednat například o výpadek propojení s databází nebo nedostupnost některé z využitých služeb.

WARN Tato úroveň není již tak jasná z pohledu identifikace chyby. Tato úroveň se použije v případech, kdy chceme dát vědět, že se vyskytla neobvyklá situace, která by mohla vyústit v závažnější problém. Může se jednat například o situaci, kdy úspěšnému spojení se serverem předcházelo několik neúspěšných pokusů. Tato situace je neobvyklá, ale k skutečnému problému nedošlo a nevíme, zdali se toto chování bude v budoucnu opakovat.

INFO Nyní se dostáváme mimo oblast problémů. Tento typ zpráv se používá k zaznamenávání klasického chování aplikace. Při běžném chodu aplikace nám na těchto záznamech nebude příliš záležet na rozdíl od těch hlásících chyby, ale poskytují nám obecný přehled toho, co se uvnitř aplikace děje. Může se například jednat o zprávy typu: „Služba XX byla spuštěna“, „Uživatel YY se přihlásil“ a další.

DEBUG Zde se začnou zahrnovat podrobnější diagnostické informace. Tím se dostaneme do oblasti, kdy začneme přijímat větší množství dat než je potřebné při běžných situacích. Poskytuje podrobné diagnostické informace, které se hodí ostatním vývojářům, správcům atd.

TRACE Je ještě podrobnější než DEBUG. Na této úrovni se v zásadě snažíme zachytit všechny možné podrobnosti o chování aplikace. V praxi to znamená, že dostaneme velmi dlouhý a podrobný výpis všech možných činností, které použijeme na důkladnou diagnostiku.

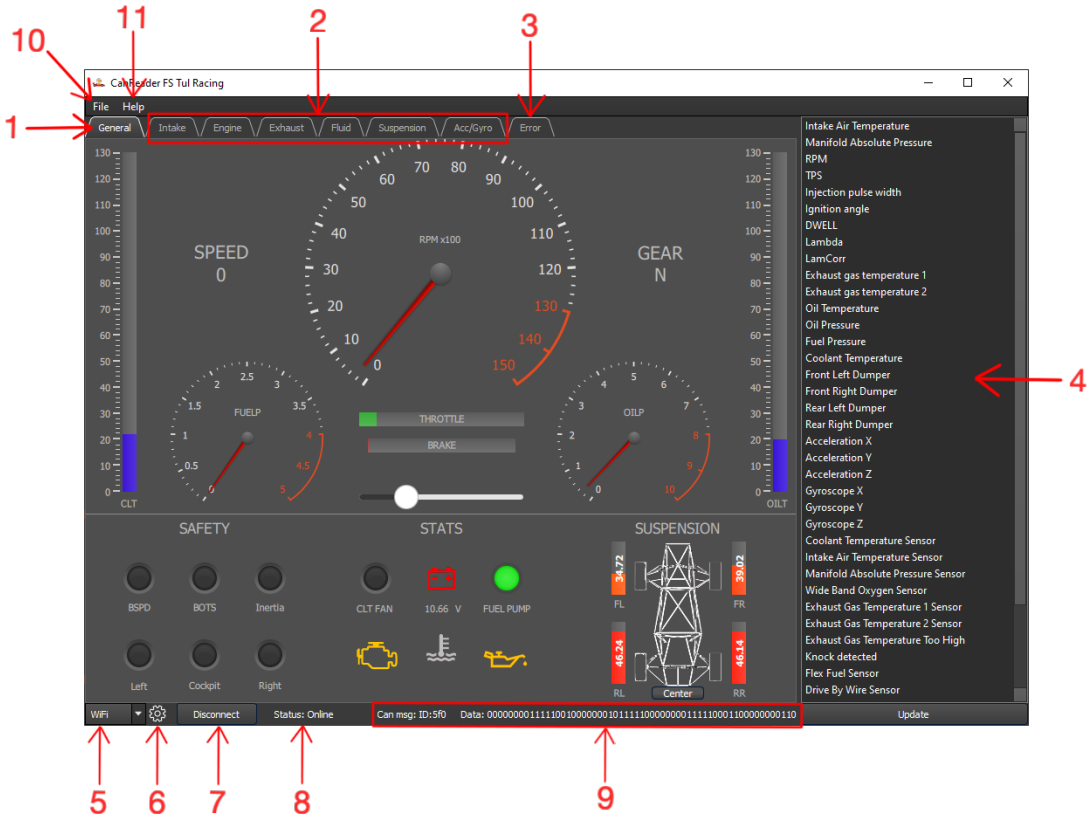
Pro usnadnění logování se používají knihovny, které nám dovolí nakonfigurovat si logger podle potřeby. Můžeme si tak například nastavit kam se budou logy ukládat, od jaké úrovně je chceme ukládat, v jakém formátu se mají ukládat a nebo i například, aby nám přišel email v případě závažné chyby. Níže je ukázka kódu 4.2, který nám v případě pokusu o práci se špatným a nebo neexistujícím sériovým portem zapíše záznam chyby do souboru. Kromě ukázky logování, je zde i ukázka dokumentace kódu. Na základě těchto informací se pak automaticky generují webové stránky obsahující kompletní dokumentaci tak, jak bylo popsáno v podsekci Dokumentace 4.1.4.

```python
@staticmethod
def check_com_port(port):
    """
        :Description:
            Check if port is a str and can be found on the system.

        :param port: Com port name.
        :type port: str

        :raises TypeError:
            Port is not a str.

        :raises OSError:
            Port cannot be found on the system.
    """
    try:
        if type(port) != str:
            raise TypeError
        if port.upper() not in SerialCom.available_com_ports():
            raise OSError
    except OSError:
        error_msg = "Cannot find port {}!".format(port)
        logging.exception(error_msg)
        raise OSError(error_msg)
    except TypeError:
        error_msg = "COM port must be a string!"
        logging.exception(error_msg)
        raise TypeError(error_msg)
```

Zdrojový kód 4.2: Ukázka použití loggeru při zachycení chyby

4.2 GUI

GUI, neboli grafické uživatelské rozhraní, slouží k interakci mezi uživatelem a aplikací. Pro uživatele to je jedna z nejdůležitějších částí aplikace a to obzvláště u aplikace, zaměřující se na grafickou prezentaci dat. Při návrhu aplikace byl na tento fakt brán zřetel a proto byl design tvořen tak, aby byl přehledný a velmi jednoduchý na ovládaní. Na obrázku 4.4 je ukázka hlavního okna aplikace včetně anotace.



Obr. 4.4: Ukázka hlavního okna aplikace

Aplikace se skládá z následujících 11 částí:

1. Přístrojový panel. Popsán v podsekci 4.2.1.
2. Panel grafů. Popsán v podsekci 4.2.2.
3. Panel chyb. Popsán v podsekci 4.2.3.
4. Výčet a nastavení dostupných CAN zpráv. Popsáno v podsekci 4.2.4
5. Volba typu komunikace.
6. Nastavení vybraného typu komunikace. Popsáno v sekci 4.3
7. Navázání/přerušení komunikace.
8. Zobrazení aktuálního stavu spojení.
9. Zobrazení poslední přijaté CAN zprávy.
10. Možnosti práce se soubory.
11. Zobrazení dokumentace, nebo okna „o aplikaci“.

4.2.1 Přístrojový panel

Přístrojový panel má za úkol informovat uživatele o aktuálním stavu formule. Ideálně tak, aby hned na první pohled bylo jasné, co se ve formuli děje. Z toho důvodu je většina hodnot znázorněna graficky. Uživatel, tak například dokáže okamžitě zjistit na základě barvy ikonky reprezentující stav baterie, jestli není vybitá a to aniž by přečetl skutečnou hodnotu napětí.

Ukázka přístrojového panelu je na obrázku 4.4. Vzhled a funkcionalita přístrojového panelu byla navržena v programu Qt Creator. Tento program slouží především pro propojení designu s kódem. Ovšem umožňuje i vytvoření jednodušších grafických návrhů. Pro vytváření komplexnějších grafických návrhů se používají specializované programy. Jedním z takových programů je například Photoshop, s nainstalovaným rozšíření pro podporu Qt. Návrh je interpretován pomocí programovacího jazyku QML, což je zkratka pro „Qt Modeling Language“. Jedná se o značkovací jazyk určený pro vytváření uživatelských rozhraní. Výsledný .qml soubor obsahuje stromovou hierarchii jednotlivých objektů. Jednotlivé QML objekty dokáží využívat JavaScript. Díky tomu má vývojář možnost například dynamicky přizpůsobovat vzhled na základě vnitřních proměnných. Níže je ukázka QML kódu v kombinaci s JavaScriptem 4.3. Konkrétně se jedná o dynamickou změnu ikonky na základě změny teploty chladicí kapaliny.

```
1  Connections{
2      target: cltGauge
3      onValueChanged:{
4          if(cltGauge.value >= 125 || parseInt(CLT_sensor.value) === 1 ){
5              waterTempStatus.source = "water_temp_error.png"
6          }
7          else if((cltGauge.value > 115) && (cltGauge.value < 125)){
8              waterTempStatus.source = "water_temp_warning.png"
9          }else{
10             waterTempStatus.source = "water_temp_normal.png"
11         }
12     }
13 }
```

Zdrojový kód 4.3: Ukázka QML kódu v kombinaci s JS

Navržená přístrojová deska se skládá z následujících 4 částí:

1. Hlavní a také nejprostornější část zobrazuje základní informace o formuli. Po stranách jsou sloupcové ukazatele zobrazující teplotu vody (vlevo) a teplotu oleje (vpravo). Pro grafické znázornění teploty se plynule mění barva ukazatele z modré (0 °C) na červenou (130 °C). Jako další zde jsou 2 ručičkové ukazatele zobrazující tlak paliva (vlevo) a tlak oleje (vpravo). Oba ukazatele mají červeně vyznačenou oblast vysokého tlaku. Poslední a také největší ručičkový ukazatel zobrazuje aktuální otáčky motoru. Dále zde jsou 2 horizontální sloupcové ukazatele indikující mírů sešlápnutí plynu a brzdy. Respektive znázorňují míru otevření škrticí klapky a tlak v brzdném okruhu. Uživatel z nich nezjistí přesnou hodnotu. Slouží spíše jenom pro orientační uvedení, zdali se brzdí nebo přidává plyn. Jako poslední zde jsou dva textové ukazatele. Číslo pod nápisem „SPEED“ zobrazuje aktuální rychlost. Hodnota uvedená pod nápisem „GEAR“ zobrazuje aktuálně zařazený stupeň.

2. Další část signalizuje stav jednotlivých komponent z bezpečnostního okruhu. Na přístrojové desce se nachází v levé spodní části pod nadpisem „SAFETY“. Bezpečností okruh je definován pravidly Formula Student. Pro spalovací vozy se skládá ze 6 sériově zapojených komponent. Součástí bezpečnostního okruhu jsou 3 stop tlačítka, BOTS, BSPD a inerciální spínač. Dvě stop tlačítka jsou umístěna na stranách formule a jedno je v kokpitu. BOTS je spínač umístěný za brzdovým pedálem, který rozepne okruh v případě prudkého brzdění. BSPD je zařízení, které rozepne bezpečnostní okruh v případě, že tlak v brzdném okruhu a otevření škrticí klapky je nad určitou úroveň. Jinými slovy má zapůsobit v případě zaseknutí brzdového pedálu. Poslední prvek bezpečnostního okruhu je inerciální spínač, označený jako „Inertia“. Tento prvek rozepne bezpečností okruh v případě nárazu. V případě přerušení bezpečnostního okruhu v důsledku výpadku kteréhokoliv prvku se přeruší napájení pro vstřiky, zapalování a palivové čerpadlo. Prvek, který způsobil výpadek bezpečnostního okruhu se na přístrojové desce rozsvítí červeně. Díky tomu budeme schopni jednodušeji určit příčinu poruchy.

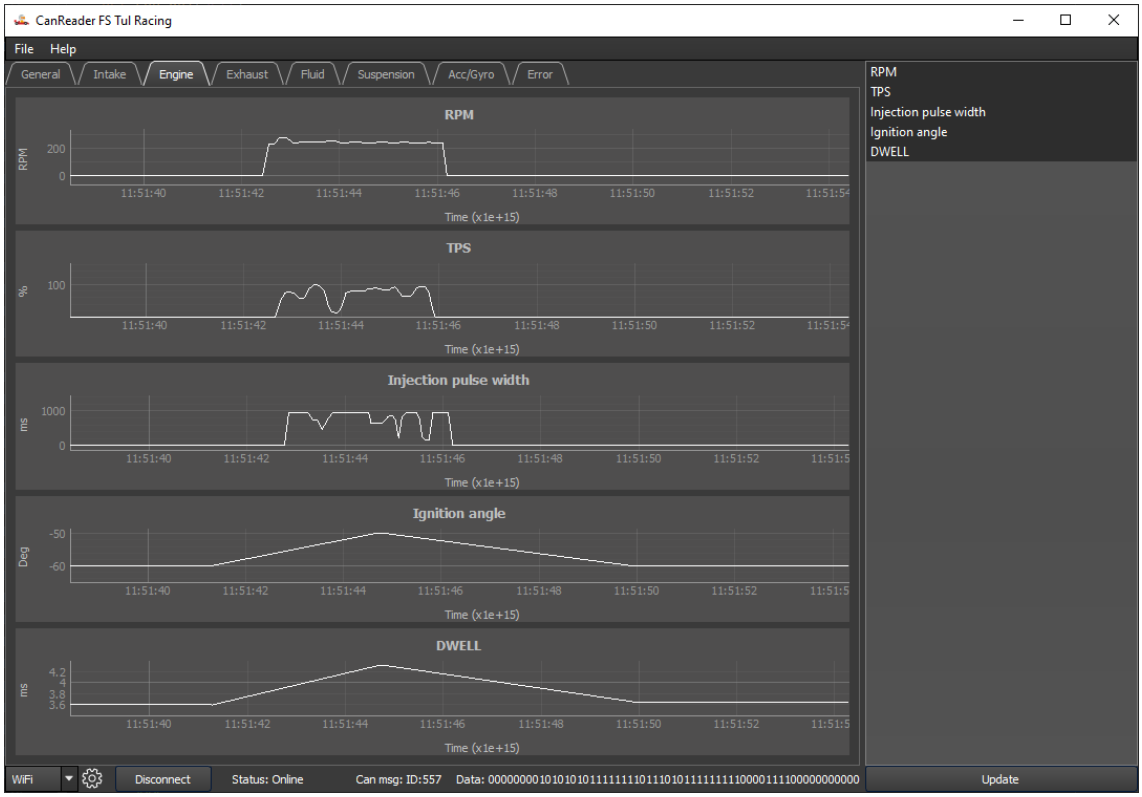
3. Prostřední část s názvem „STATS“ informuje uživatele o důležitých stavech formule. Obsahuje 6 komponent, 4 ikonky a 2 binární indikátory. Levý indikátor se rozsvítí zeleně v případě, že je spuštěn ventilátor chlazení vodního okruhu. Pravý indikátor se rozsvítí zeleně v případě, že je spuštěna palivová pumpa. Mezi nimi je ikonka baterie, která se rozsvítí červeně, klesne-li napětí na baterii pod 11 V. Pod ikonkou je i text zobrazující skutečnou hodnotu napětí na baterii. Levá spodní ikonka značí stav motoru. Žlutě se rozsvítí v případě, že tlak paliva klesne pod 1 bar. Červeně se rozsvítí v případě, že tlak paliva bude větší než 4 bary, vyskytne se chyba se senzorem tlaku paliva, vyskytne se chyba s WBO senzorem a nebo se detekuje klepání na motoru. Prostřední ikonka vespod slouží ke znázornění stavu chladicího okruhu. Žlutě se rozsvítí v případě, že teplota chladicí kapaliny je větší než 115 °C. Červeně se rozsvítí v případě, že teplota chladicí kapaliny je větší než 125 °C a nebo když se vy-

skytne chyba se senzorem teploty chladicí kapaliny. Poslední ikonka, umístěná v pravém spodním rohu, reaguje na tlak oleje. Žlutě se rozsvítí v případě, že tlak oleje klesne pod 1 bar. Červeně se rozsvítí v případě, že tlak oleje překročí 8 barů a nebo nastane chyba se senzorem tlaku oleje.

4. Poslední část s názvem „SUSPENSION“ dává uživateli přehled o stlačení pružin jednotlivých zavěšení. Hodnota stlačení je zde graficky znázorněna pomocí sloupcového indikátoru. Kromě toho je i ve středu každého indikátoru text udávající aktuální hodnotu stlačení. Uživatel má možnost si programově vycentrovat hodnoty stlačení pomocí tlačítka „Center“. Díky tomu pak může pozorovat relativní stlačení pružin vůči sobě.

4.2.2 Grafický panel

Jedním ze způsobů zobrazení dat je zobrazení jednotlivých veličin v samostatných grafech viz obrázek 4.5. Momentálně je v aplikaci dostupných 6 záložek s grafy na obrázku 4.4 označeny číslem 2. Každá záložka tvoří skupinu dat, které spolu logicky souvisí. Například v záložce „Engine“ jsou zobrazena data související s chodem motoru. Uživatel se zde dozví informace jako otáčky motoru, otevření klapky, doba vstřikování paliva, atd.



Obr. 4.5: Ukázka panelu s grafy

Přidat nebo odebrat jednotlivé grafy uživatel momentálně nemůže přímo v aplikaci. V případě, že by chtěl nastavit co, kde a v jakém pořadí se bude zobrazovat, tak má možnost upravit konfigurační soubor, který obsahuje všechny tyto informace.

Každý graf se dá samostatně ovládat. Levým tlačítkem myši se může uživatel libovolně pohybovat v grafu. Kolečkem se ovládá přiblížení nebo oddálení pohledu. Stiskem pravého tlačítka se otevře menu. V menu si uživatel může nastavit rozsah obou os, zobrazení mřížky, způsob vykreslení, aj. Je zde i možnost vyexportovat data v .csv formátu. Případně si uložit celý graf, nebo jen jeho část, jako obrázek ve standardním formátu jako je .png, .jpg a nebo vektorový .svg. V případě, že by uživatel chtěl přejít zpět na nejnovější data, tak může stisknutím klávesy „c“ jako „center“ nebo klávesou „f“ jako „focus“.

4.2.3 Chybový panel

Speciálním panelem je panel zobrazující případné chyby jednotlivých komponent viz obrázek 4.6. Všechny indikátory v tomto panelu jsou napojeny na veličiny, které nabývají pouze hodnoty 0 nebo 1.

Obr. 4.6: Ukázka panelu chyb

Indikátor může nabývat jeden z následujících tří stavů:

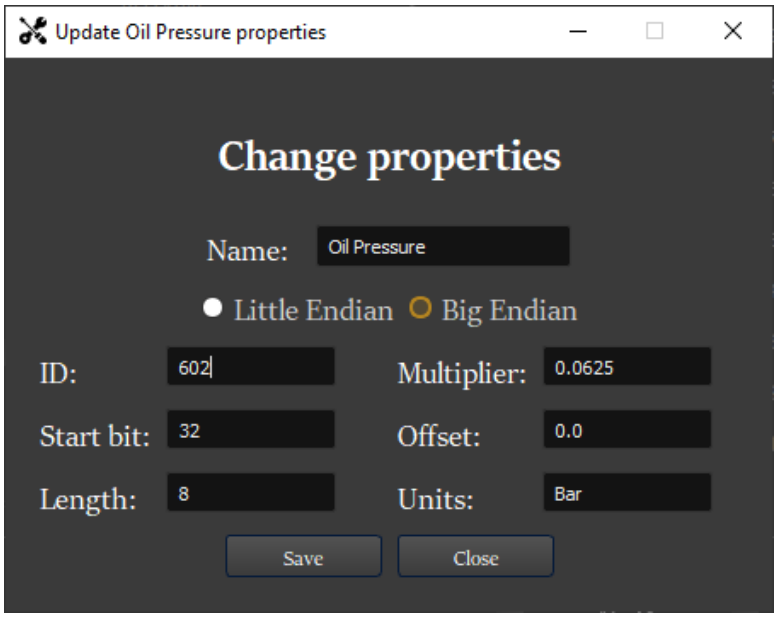
1. **OK** – Tento stav je znázorněn zeleným indikátorem a znamená, že je vše v pořádku.

2. **Error** – Tento stav je znázorněn červeným indikátorem a znamená výskyt chyby. Mimo změny indikátoru se zobrazí i čas, kdy k chybě došlo a tlačítko na restart. Chyba nezmizí dokud ji uživatel manuálně nerestartuje. Tím je zaručeno, že uživatel bude informován o chybě i v případě, že se zrovna nedívá na panel chyb.

3. **No data** – Tento stav je znázorněn žlutým indikátorem a znamená, že od spuštění a nebo od restartování chyby nepřišla informace o aktuálním stavu.

4.2.4 CAN specifikace

Jednou z klíčových vlastností aplikace je možnost nastavení CAN specifikace pro jednotlivé proměnné. Uživatel tak například může jednoduše reagovat na změnu CANu ve formuli. Seznam všech sledovaných proměnných je v případě přístrojové desky zobrazen na pravé straně aplikace viz obrázek 4.4 s označením 4. V případě ostatních typů zobrazení jsou v seznamu pouze ty proměnné, které se vyskytují v daném zobrazení. Uživatel může pro libovolnou proměnnou otevřít okno s nastavením viz obrázek 4.7. A to buď označením jedné z proměnných a následným kliknutím na tlačítko update a nebo dvojkliknutím na libovolnou z nich.



Obr. 4.7: Ukázka nastavení

Uživatel může pro každou CAN informaci nastavit tyto parametry:

- **Endianita** – Uživatel si může vybrat mezi malý a velkým endianem viz 4.4.1.
- **ID** – Identifikátor CAN zprávy. Musí být v HEX formátu s maximální délkou 8 znaků.
- **Start bit** – Bitový offset v datové části CAN zprávy. Musí být v rozsahu 0 – 63.
- **Length** – Délka oblasti zájmu v bitech. Musí být v rozsahu 1 – 63.
- **Multiplier** – Násobitel proměnné.
- **Offset** – Ofset proměnné. Spolu s násobitelem se používají pro získání skutečné hodnoty.

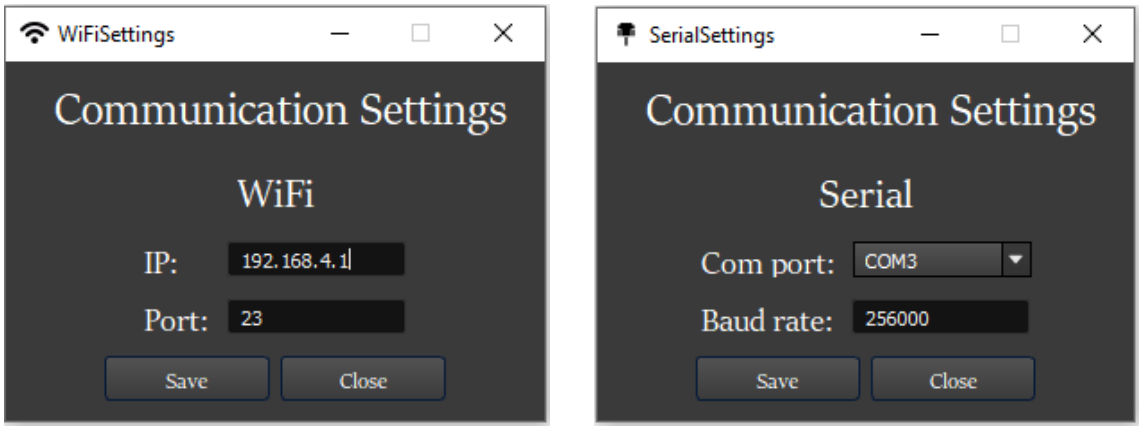
Uživatel nemůže změnit název a jednotku proměnné. Stejně tak zde není možné nastavit v jaké záložce a na jaké pozici se má vygenerovat grafické znázornění dané zprávy. V případě potřeby se dají tyto vlastnosti upravit přímo v konfiguračním souboru aplikace.

4.3 Komunikace

Komunikační podprogram má, jak již z názvu vyplývá, na starost komunikaci mezi aplikací a zařízením, které dodává potřebná data. Schválně zde není uvedeno, že má na starost komunikaci mezi formulí a aplikací, a to i přes to, že je to v podstatě pravda. Důvodem je, že si uživatel může v aplikaci vybrat ze dvou způsobů, jak navázat komunikaci a pouze jeden z nich navazuje přímou komunikaci s formulí.

První způsob, jak může uživatel komunikovat s formulí, je přímo pomocí Wi-Fi za využití protokolů, které zařízení ve formuli podporuje, viz 3.5.3. Uživatel si při tomto druhu komunikace musí nastavit IP adresu, na které očekává, že bude nakonfigurováno zařízení ve formuli. Stejně tak si musí nastavit port, na kterém se otevře komunikační kanál mezi formulí a komunikačním podprogramem. I toto nastavení musí být shodné s tím ve formuli. Ukázka grafického prostředí, přes které si uživatel nastavuje tyto hodnoty, je na obrázku 4.8a.

Druhým způsobem je komunikace po sériové lince. Při tomto druhu komunikace aplikace dostává data od fyzicky připojeného mezičlenu, který přímo komunikuje s formulí. I zde musí uživatel správně nastavit základní parametry, aby mohla aplikace komunikovat s mezičlenem. Jako první musí uživatel vybrat port, ke kterému je mezičlen připojen. Zde si uživatel vybírá pouze z aktivních portů. Díky tomu dokáže uživatel rychle identifikovat, jaký port přísluší pravě připojenému zařízení. Druhým nastavovaným parametrem je baudová rychlost. Ta musí být nastavena na stejnou hodnotu jako je v mezičlenu. Ukázka grafického prostředí pro nastavení sériové komunikace je na obrázku 4.8b.



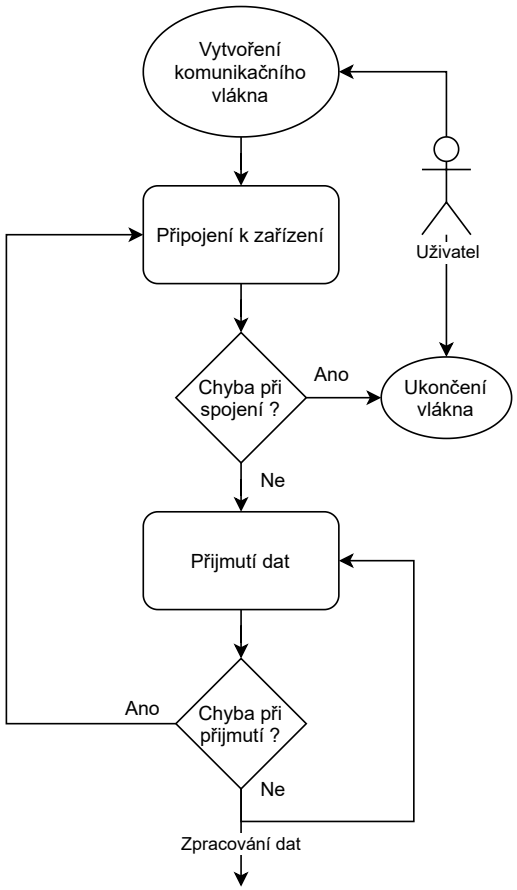
(a) Wi-Fi komunikace (b) Sériová komunikace

Obr. 4.8: Nastavení komunikace

Nyní již k podrobnějšímu popisu funkce pomocí vývojového diagramu viz sekce 4.9. Komunikační vlákno se vytvoří v reakci na uživatelův požadavek na připojení se k formuli. Na základě uživatelova požadavku se tedy inicializuje komunikační vlákno založené na jedné ze dvou komunikačních tříd. Pro unifikaci komunikace obě komunikační třídy vychází z jedné základní třídy. Ta implementuje základní metody a signály, předepisuje jaké metody mají její potomci implementovat a také jaký je základní sled událostí.

Při inicializaci se, kromě nastavení základních parametrů, také kontroluje, zdali načtené parametry mají hodnotu, které mohou skutečně nabýt. Například, zdali parametr určující port sériové komunikace je na systému dostupný a jestli je typu *string* viz kód 4.2 použitý k ukázce logování.

Po úspěšném vytvoření komunikačního vlákna se podprogram pokusí navázat spojení s cílovým zařízením. Stav spojení je signalizován jedním z 5 možných stavů, které jsou prezentovány uživateli. První stav je„Offline“. Tento stav je zobrazen v situaci, kdy není vytvořeno žádné komunikační vlákno, je to také počáteční hodnota při inicializaci. Další dva stavy jsou „Connecting“ a „Reconnecting“, které jsou zobrazeny v době, kdy se podprogram pokouší navázat spojení se zařízením. Jediný rozdíl mezi těmito dvěma je ten, že „Reconnecting“ je zobrazen pouze v případě, že se zařízení pokouší znovu připojit, respektive předcházel mu problém při přijmutí dat. Následně je zde stav „Online“, který signalizuje, že se podařilo úspěšně navázat spojení a komunikace probíhá bez problémů. Posledním stavem je stav „Failed“, který se zobrazí v případě, že došlo k chybě při navazování spojení. Tento stav také znamená, že se podprogram automaticky ukončí a přejde do stavu „Offline“. Typickým případem, kdy dojde na stav „Failed“ je, když se snažíme navázat spojení se zařízením, ale zapomněli jsme se připojit do společné sítě v případě Wi-Fi. A nebo se snažíme otevřít komunikační kanál se zařízením, které již komunikuje s jiným programem v případě sériové komunikace.



Obr. 4.9: Vývojový diagram komunikačního podprogramu

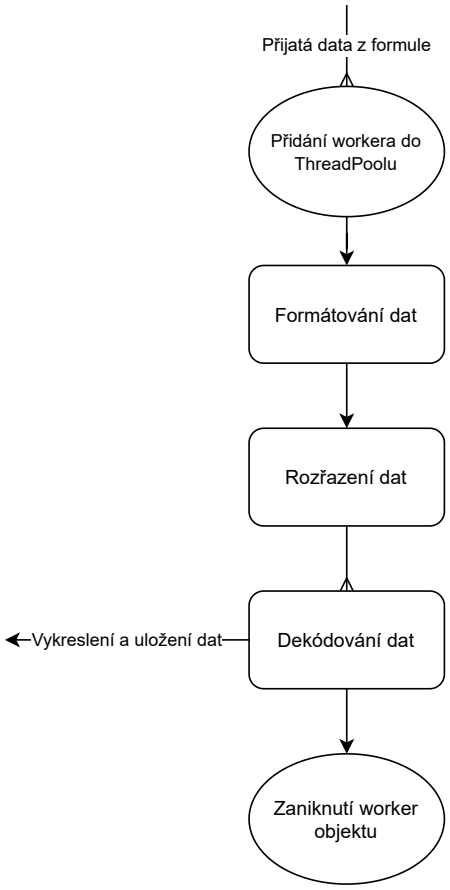
Jakmile je úspěšně navázáno spojení, jsme tedy ve stavu „Online“, tak začneme přijímat data, která nám formule posílá. Přijatá data se odešlou do hlavního programu na zpracování (proces zpracování je popsán v sekci 4.4). V případě, že nastane problém při přijmutí dat, tak program opět přejde do části kódu, která má na starost navázání spojení. Pokusí se celý proces komunikace a s tím spojeného nastavení provést znovu. Nejčastěji je tato chyba způsobena nedodáním dat v určitém časovém limitu od požádání. Takováto situace může nastat například tím, že se formule dostane mimo dosah.

4.4 Zpracování dat

Všechna přijatá data z komunikačního podprogramu jsou odeslána do hlavního programu, kde se předají na další zpracování. Pro zajištění zpracování dat v téměř reálném čase se využívá metoda rozdělení práce do několika vláken, které spravuje jeden *threadpool*. Jak již z předchozí věty vyplývá, tak *threadpool* je tedy takový správce, který spravuje několik vláken a dosazuje do nich práci podle potřeby. Obrovskou výhodou *threadpoolu* je, že všechna spravovaná vlákna jsou předpřipravena a po dokončení práce nezanikají. Tudíž se tím ušetří čas, který by jinak byl potřeb-

ný pro vytvoření a zaniknutí vláken, což u velmi krátkých a frekventovaných úkolů může negativně ovlivnit výkon celé aplikace. Další zajímavou vlastností *threadpoolu* je, že počet spravovaných vláken se může dynamicky měnit například v zavilosti na aktuálním vytížení, neboli počtu čekajících úkolů ve frontě.

Nyní, když je vysvětlen princip fungování *threadpoolu*, tak můžeme přejít k samotnému zpracování dat. I zde pro usnadnění pochopení použijeme vývojový diagram, viz 4.10. Prvním krokem je přidání *worker* objektu do *threadpoolu*. Přidaný objekt má funkci manažera zpracování dat, vyvolává jednotlivé kroky v určitém pořadí a předává mezi nimi potřebná data.



Obr. 4.10: Vývojový diagram pro zpracování dat

Formátování dat v tomto kontextu znamená, že se vezmou přijatá data a rozdělí se na dvě části CAN ID a CAN data. ID se získá z prvních 4 bajtů přijaté zprávy. Po formátování je vráceno v podobě hexadecimálního čísla. Data se získají ze zbylých 8 bajtů. Data se navrací v binárním formátu s fixní délkou 64 bitů, jsou tedy zachovány všechny bity včetně vedoucích nul.

Tyto dvě získané hodnoty se spolu se specifikačním listem CAN zpráv odešlou na rozřazení. Zde se podle získaného CAN ID přiřadí existující specifikace. Pro každou specifikaci se vytvoří samostatný objekt, který se následně používá při směrování zprávy k příslušnému grafickému prvku. Můžeme tak tedy pro jednu CAN zprávu získat až 64 samostatně zpracovaných objektů. Každý takto zpracovaný objekt obsahuje následující informace:

- **overview id** – ID v hlavním zobrazení
- **group id** – ID skupiny v UI.
- **widget id** – ID widgetu v příslušné skupině.
- **name** – Název (teplota oleje, ...)
- **value** – Skutečná hodnota po dekódování.

Skutečnou hodnotu, kterou vytvořené objekty obsahují, získáme dekódováním CAN dat pomocí příslušné specifikace. Podrobná podoba specifikace je popsaná v sekci 4.2.4. Samotný proces dekódování se skládá z následujících kroků:

1. Získáme oblast zájmu na základě informace o pozici počátečního bitu a délce zprávy v bitech. Obě tyto informace se získají z dané specifikace.
2. Ze specifikace zjistíme endianitu a na základě toho přizpůsobíme data, viz 4.4.1.
3. Převedeme data do desítkové soustavy.
4. Vynásobíme data násobitelem, kterého získáme ze specifikace.
5. Přičteme k výsledku offset, který získáme taktéž ze specifikace.

4.4.1 Endianita

Endianita určuje v jakém pořadí jsou uloženy jednotlivé bajty číselného datového typu [26]. Uživatel si při vytváření specifikace pro jednotlivé CAN zprávy může vybrat, jestli se daná hodnota bude ukládat jako *bigenidan* a nebo *littleendian*.

Little endian
: V tomto případě se na paměťové místo s nejnižší adresou uloží nejméně významný bajt (LSB) a za něj se ukládají ostatní bajty až po nejvíce významný bajt (MSB).

Big endian
: V tomto případě se na paměťové místo s nejnižší adresou uloží nejvíce významný bajt (MSB) a za něj se ukládají ostatní bajty až po nejméně významný bajt (LSB).

4.4.2 Ukládání dat

Jednou z vlastností programu je i průběžné ukládání přijatých dat do souboru. Zpracovaná data jsou odeslána do hlavního vlákna, kde se kromě předání na zobrazení předají i na uložení. Objekt starající se o ukládání dat obsahuje až 1000 posledních zpráv. Každý uložený záznam obsahuje časovou značku, CAN ID a nezpracovaná data v binární podobě. V případě dosažení limitu tedy 1000 zpráv, se získaná data uloží do souboru. Tím je zajištěno, že program nebude zbytečně zpomalován neustálým zapisováním do souboru.

Již po pár minutách by soubor obsahoval několik 100 MB dat. Z toho důvodu je potřeba chytře redukovat ukládaná data. Jelikož drtivá většina CAN zpráv je generována periodicky a nikoliv jako odezva na nějaký podnět, tak se velmi často opakuje obsah jednotlivých zpráv. Této vlastnosti můžeme využít k ušetření velkého množství dat a to tak, že budeme ukládat pouze rozdílné zprávy. Musíme si tedy pro každé ID držet poslední hodnotu. V případě, že se příchozí data liší od těch uložených, tak nová data zapíšeme v požadovaném formátu do bufferu a zapíšeme novou poslední hodnotu pro dané ID.

5 Závěr

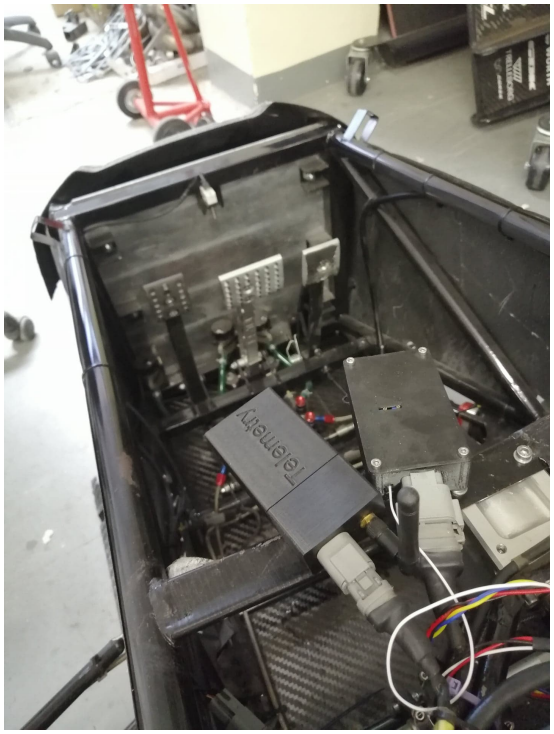
Cílem bakalářské práce bylo navrhnout, realizovat a následně otestovat telemetrický systém pro studentskou formuli. Na základě rešerše byla vybrána přenosová technologie využívající frekvenci 2,4 GHz. Pro vybranou technologii byl zvolen vhodný mikrokontrolér, kterým je ESP32. Pro zajištění většího dosahu byly navrženy dvě varianty řešení. V první variantě probíhá komunikace přímo mezi formulí a počítačem uživatele. Druhá varianta zajišťuje větší dosah, kde probíhá komunikace mezi formulí a mezičlenem a zároveň paralelně mezi mezičlenem a počítačem uživatele. Následně se výpočty zjistilo, zdali je s danými konfiguracemi možné dosáhnout požadovaných vzdáleností. Po potvrzení dosažitelnosti bylo navrženo zařízení, které dokáže sbírat data ze sběrnice CAN a zároveň je odesílat uživateli. Jako případný mezičlen se využil vývojový kit ESP32. Pro obě tato zařízení byl napsán firmware v jazyce C++ za využití prvků FreeRTOS.

V druhé části práce jsem se zabýval vývojem aplikace pro zpracování a vizualizaci přijatých dat. Aplikace s názvem „CAN Reader“ byla vytvořena v jazyce Python 3 za využití frameworku Qt. Výsledná aplikace dokáže komunikovat pomocí obou navržených řešení. Data dokáže vizualizovat ve třech režimech. Prvním režim imituje funkci klasické přístrojové desky. Druhý režim zobrazuje vývoj dat pomocí grafů. A nakonec třetí režim, který je určen pro vizualizaci případných chyb. Aplikace všechna přijatá data ukládá s vhodnou bezztrátovou kompresí, díky které je výsledný soubor poměrně malý a zároveň člověkem čitelný.

Při tvorbě aplikace byl brán zřetel na dlouhodobou udržitelnost kódu, kvůli potřebě předání aplikace do správy členům týmu. Z tohoto důvodu je aplikace verzovaná pomocí GITu, jsou k ní vytvořené sady testů a má nastavené CI. Běh programu je vhodně logován a k aplikaci je vytvořená kompletní dokumentace.

Všechny body zadání byly splněny, avšak z důvodu nepojízdnosti formule, byl systém otestován pouze v dílenských podmínkách, při kterých se choval dle očekávání. Jakmile bude dokončena stavba nové formule, bude systém otestován znovu, tentokrát již v podmínkách závodu. Zde bude třeba zjistit, jaký je skutečný komunikační dosah systému a jaký dopad bude mít použití protokolu TCP na přenosovou rychlost v případě vzdalování se formule od přijímače.

Do budoucna by bylo dobré rozšířit aplikaci o možnost zpětné vizualizace dat ze souboru.



Obr. 5.1: Ukázka provizorní implementace do formule

Literatura

[1] SAE INTERNTIONAL. *Formula SAE* [online]. 2020 [cit. 2020-10-02]. Dostupné z: http://www.fsaeonline.com.

[2] FORMULA STUDENT GERMANY. *FS Rules 2020 v1.0.* [online] [cit. 2020-10-02]. Dostupné z: https://www.formulastudent.de/fileadmin/user_upload/all/2020/rules/FS-Rules_2020_V1.0.pdf.

[3] STACKIFY. *What is Telemetry? Application Monitoring and Performance Tools For Developers* [online] [cit. 2020-12-26]. Dostupné z: https://stackify.com/telemetry-tutorial/.

[4] BOSCH. *CAN Specification 2.0* [online] [cit. 2020-12-28]. Dostupné z: http://esd.cs.ucr.edu/webres/can20.pdf.

[5] WIKIPEDIE. *Referenční model ISO/OSI* [online] [cit. 2020-12-18]. Dostupné z: https://en.wikipedia.org/wiki/OSI_model.

[6] EE JRW. *ISO 11898-2 High Speed CAN Signaling.* 2017. Dostupné také z: https://en.wikipedia.org/wiki/CAN_bus#/media/File:ISO11898-2.svg.

[7] LINEAR77. *Noise reduction using differential signaling.* 2012. Dostupné také z: https://en.wikipedia.org/wiki/Differential_signaling#/media/File:DiffSignaling.png.

[8] LORA ALLIANCE. *A technical overview of LoRa® and LoRaWAN™* [online] [cit. 2021-02-28]. Dostupné z: https://lora-alliance.org/wp-content/uploads/2020/11/what-is-lorawan.pdf.

[9] DIGI INTERNATIONAL. *Digi XBee-PRO 900HP RF Module* [online] [cit. 2021-02-28]. Dostupné z: https://www.digi.com/products/embedded-systems/digi-xbee/rf-modules/sub-1-ghz-rf-modules/xbee-pro-900hp.

[10] DIGI INTERNATIONAL. *Digi XBee 3 802.15.4 RF Module* [online] [cit. 2021-02-28]. Dostupné z: https://www.digi.com/products/embedded-systems/digi-xbee/rf-modules/2-4-ghz-rf-modules/xbee3-802-15-4.

[11] BLACKBOX. *Wi-Fi Standards Explained* [online] [cit. 2021-03-09]. Dostupné z: https://www.bboxservices.com/resources/blog/bbns/2018/04/30/802.11-wireless-standards-explained.

[12] KHOROV, E. et al. A Tutorial on IEEE 802.11ax High Efficiency WLANs. *IEEE Communications Surveys Tutorials.* 2019, roč. 21, č. 1, s. 197–216. Dostupné z DOI: 10.1109/COMST.2018.2871099.

[13] WI-FI ALLIANCE. *Wi-Fi Halow* [online] [cit. 2021-03-09]. Dostupné z: https://www.wi-fi.org/discover-wi-fi/wi-fi-halow.

[14] ESPRESSIF SYSTEMS. *ESP32 Datasheet* [online] [cit. 2021-02-28]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[15] WHITAKER, J.C. *The Electronics Handbook*. Taylor & Francis, 1996. Electrical Engineering Handbook. ISBN 978-0849383458. Dostupné také z: https://books.google.cz/books?id=DSHSqWQXm3oC.

[16] TOMASI, Wayne. *Electronic Communications System: Fundamentals Through Advanced*. Pearson Education, 2009. ISBN 978-8131719534. Dostupné také z: https://books.google.cz/books?id=eX53FhL2DnoC.

[17] JCMCCLURG. *Fresnel zone*. 2012. Dostupné také z: https://en.wikipedia.org/wiki/Fresnel_zone#/media/File:FresnelSVG1.svg.

[18] PLANET3 WIRELESS. *CWNA Certified Wireless Network Administrator Official Study Guide*. McGraw-Hill/Osborne, 2005. Certification Press Series. ISBN 978-0072255386. Dostupné také z: https://books.google.cz/books?id=QnMunBGVDuMC.

[19] ZYTRAX. *Wireless Overview* [online] [cit. 2020-12-28]. Dostupné z: https://www.zytrax.com/tech/wireless/intro.htm.

[20] COMMOTION. *Learn Wireless Basic* [online] [cit. 2021-03-01]. Dostupné z: https://commotionwireless.net/docs/cck/networking/learn-wireless-basics/.

[21] MILLENNIUM CIRCUITS LIMITED. *Guide to PCB Trace Width vs. Current Table* [online] [cit. 2021-03-15]. Dostupné z: https://www.mclpcb.com/pcb-trace-width-vs-current-table.

[22] VACEK, Ondřej. *Telemetry* [online]. GitHub, 2021 [cit. 2021-05-05]. Dostupné z: https://github.com/GKPr0/Formula-Student-Telemetry.

[23] CHACON, Scott; STRAUB, Ben. *Pro Git (Second Edition)*. Apress, 2014. ISBN 978-1484200773. Dostupné také z: https://git-scm.com/book/en/v2.

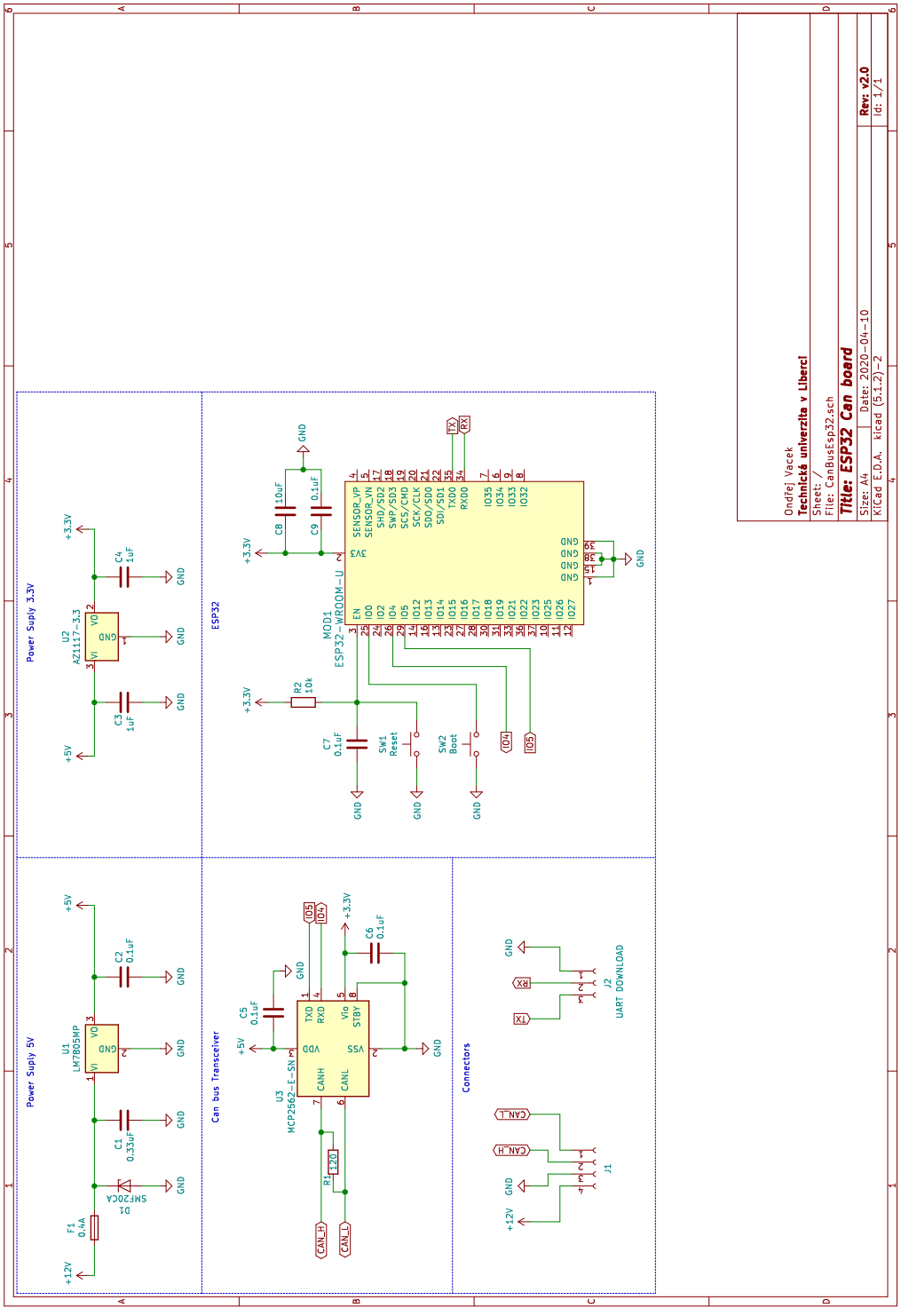
[24] BRANDL, Georg. *Sphinx Python Documentation Generator* [online] [cit. 2021-04-10]. Dostupné z: https://www.sphinx-doc.org/en/master/.

[25] VACEK, Ondřej. *CAN Reader* [online]. GitHub, 2021 [cit. 2021-05-16]. Dostupné z: https://gkpr0.github.io/Formula-Student-Telemetry/about.html.

[26] WIKIPEDIE. *Endianita* [online] [cit. 2021-04-18]. Dostupné z: https://cs.wikipedia.org/wiki/Endianita.

Přílohy

A Elektrické schéma



B Seznam součástek

Název	ks	Objednací číslo	Poznámka	Pouzdro	Obchod	Cena/Ks	Suma
MCP2562-E/SN	1	579-MCP2562-E/SN	CAN/LIN	SOIC-8	Mouser	21.87	21.87
LM7805MP/NOPB	1	595-LM7805MP/NOPB	Lin. Stabilizátor 5V	SOT-223-4	Mouser	31.86	31.86
AZ1117EH-3.3TRG1	1	621-AZ1117EH-3.3TRG1	LDO regulátor 3.3V	SOT-223-3	Mouser	10.69	10.69
ESP32-WROOM-32U	1	356-ESP32-WROOM-32U	ESP32 s ext. antenou	ESP32	Mouser	92.34	92.34
SMF20CA	1	576-SMF20CA	ESD protection - diode	SOD-123FL-2	Mouser	10.76	10.76
CC0603KRX5R8BB105	2	603-CC603KRX5R8BB105	Kondenzátor 1uF 25V 10 %	0603 (1608 metric)	Mouser	2.43	4.86
C1608X5R1E334K080AC	1	810-C1608X5R1E334K	Kondenzátor 0.33uF 25V 10%	0603 (1608 metric)	Mouser	2.34	2.34
C885012206071	5	710-885012206071	Kondenzátor 0.1uF 25V 10%	0603 (1608 metric)	Mouser	1.04	5.2
JMK107ABJ106KA-T	1	963-JMK107ABJ106KA-T	Kondenzátor 10uF 6.3V 10%	0603 (1608 metric)	Mouser	3.65	3.65
CRCW0603310K0JNEBC	1	71-CRCW0603310K0JNEBC	Rezistor 10kOhms 5%	0603 (1608 metric)	Mouser	2.43	2.43
CRGCQ0603F120R	1	279-CRGCQ0603F120R	Rezistor 120Ohms 1%	0603 (1608 metric)	Mouser	2.43	2.43
MF-USMF020-2	1	652-MF-USMF020-2	Vratná pojistka 0.4A	1210 (3225 metric)	Mouser	8.51	8.51
LL3301NF065QG	2	612-LL3301NF065QG	Tlačítko	-	Mouser	18.95	37.9
					Celkem:		234.84