VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NÁVRH ŘÍDICÍ JEDNOTKY AUTONOMNÍHO VOZIDLA

DESIGN OF CONTROL UNIT FOR AN AUTONOMOUS VEHICLE

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Martin Vobruba

AUTHOR

VEDOUCÍ PRÁCE — Ing. Roman Adámek

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Martin Vobruba**
Studijní program:	Mechatronika
Studijní obor:	bez specializace
Vedoucí práce:	**Ing. Roman Adámek**
Akademický rok:	2023/24

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Návrh řídicí jednotky autonomního vozidla

Stručná charakteristika problematiky úkolu:

Řídicí systém autonomních vozidel je složen z řady prvků, počínaje senzory snímajícími okolí, výpočetními a řídicími jednotkami a konče motory zajišťující pohyb vozidla. Klíčovou částí je řídicí jednotka, která přijímá povely od navigační části systému ohledně žádané rychlosti a směru pohybu vozidla a na základě nich určí potřebné rychlosti a natočení jednotlivých kol s využitím znalosti jeho kinematického a popřípadě i dynamického modelu. Tvorba právě takové řídicí jednotky, která nahradí již existující řešení s uzavřeným zdrojovým kódem, pro šestikolové vozidlo se samostatně natáčivými a řízenými nápravami je cílem této bakalářské práce.

Cíle bakalářské práce:

1) Analyzujte současnou řídicí jednotku vozidla, její propojení s ostatními komponenty, vstupní i výstupní signály a komunikační sběrnice.

2) Na základě provedené analýzy navrhněte vlastní řídicí jednotku, která bude zpětně kompatibilní se současným řešením.

3) Vytvořte firmware pro navrženou řídicí jednotku, který bude přijímat řídicí signály přes sériovou linku a na základě modelu vozidla a zvoleného jízdního režimu určí akční zásahy pro elektromotory zajišťující zatáčení a pohyb vozidla. Odpovědí řídicí jednotky bude zpráva o aktuálním stavu zpět do nadřazené řídicí jednotky. Tato zpráva bude obsahovat informace o aktuální rychlosti, natočení kol, údaje z enkodérů atd. Implementujte kinematické modely pro zatáčení první a třetí nápravou, krabí režim, kdy se všechny nápravy natáčí stejně a případně i další jízdní režimy. Implementujte i bezpečnostní funkce pro zastavení vozidla v případě chyby na motorech a ztráty komunikace s nadřazeným řídicím systémem.

4) Ověřte funkčnost vytvořené řídicí jednotky a její zpětnou kompatibilitu s původním řešení.

Seznam doporučené literatury:

[1] ZÁHLAVA, Vít. Návrh a konstrukce desek plošných spojů: principy a pravidla praktického návrhu. Praha: BEN - technická literatura, 2010. ISBN 9788073002664.

[2] VALÁŠEK, Michael. Mechatronika. Dot. 1. vyd. Praha: České vysoké učení technické, 1996. ISBN 80-010-1276-X.

[3] NOSKIEVIČ, Petr. Modelování a identifikace systémů. Ostrava: Montanex, 1999. ISBN 80-722-50-0-2.

[4] GREPL, Robert. Kinematika a dynamika mechatronických systémů. Brno: Akademické nakladatelství CERM, 2007. ISBN 978-80-214-3530-8.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2023/24

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Tato práce se zabývá návrhem řídící jednotky autonomního vozidla.

Nová jednotka nahrazuje předešlou, ke které nebyl dostupný zdrojový kód a nedala se tedy nijak upravovat její funkcionalita. Úkolem řídící jednotky je přijmout zprávu od nadřazeného počítače, zpracovat ji, dle příkazu ovládat pohony a informovat řídící počítač o jejich současném stavu.

Nejprve byla provedena rešerše dané problematiky (kinematika vozidla, CAN sběrnice). Praktická část práce se zabývá analýzou předešlé jednotky, která je klíčová k jejímu nahrazení. Následuje tvorba firmwaru a navržení hardwaru. Poslední praktickou částí je otestování funkčnosti jednotky.

Klíčová slova

Autonomní vozidlo, Řídící jednotka, Aktuátor, Servopohon, Sběrnice CAN, TAROS

ABSTRACT

The topic of this bachelor thesis is designing control unit for an autonomous vehicle.

The new ECU replaces the old one which was without access to the source code, so its functionality couldn’t be modified. The purpose of the ECU is to receive a message from the control computer, process it, control the actuators according to the orders and inform the computer about their current state.

The first part of the thesis is research of the vehicle kinematics and CAN bus. The practical part focuses on the analysis of the previous ECU which is crucial to its replacement. It is followed by the firmware and hardware design. The last practical part of the thesis is the testing of the new ECU’s functionality.

Key words

Autonomous vehicle, Control unit, Linear actuator, Servo drive, CAN bus, TAROS

BIBLIOGRAFICKÁ CITACE

VOBRUBA, Martin. *Návrh řídicí jednotky autonomního vozidla.* Brno, 2024. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/157780. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Roman Adámek.

VOBRUBA, Martin. *Návrh řídicí jednotky autonomního vozidla* [online]. Brno, 2024 [cit. 2024-05-14]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/157780. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Roman Adámek.

PROHLÁŠENÍ

Prohlašuji, že jsem *bakalářskou* práci na téma **Návrh řídící jednotky autonomního vozidla** a její přílohy vypracoval samostatně s použitím odborné literatury a pramenů, uvedených v seznamu, který je součástí této práce.

Martin Vobruba

Datum	*Jméno a příjmení*

PODĚKOVÁNÍ

Děkuji tímto Ing. Romanu Adámkovi za cenné připomínky a rady, které mi poskytl při vypracování závěrečné práce. Děkuji také své nejbližší rodině za podporu při tvorbě práce.

Martin Vobruba

OBSAH

1 ÚVOD ... 11
2 REŠERŠE ... 12
2.1 TAROS 6x6 ... 12
2.2 Ackermannovo řízení ... 13
2.2.1 Výpočet Ackermannovy kinematiky ... 15
2.3 Diferenciální podvozek ... 18
2.4 CAN bus ... 19
2.4.1 Princip funkce ... 19
2.4.2 Struktura zprávy ... 21
2.4.3 Výhody CAN busu ... 23
3 Analýza předešlé řídící jednotky ... 24
3.1 Součásti jednotky ... 24
3.2 Vstupní a výstupní signály ... 25
3.3 Vstupní a výstupní konektory ... 25
3.4 Určení maximálního úhlu natočení ... 26
4 Firmware ... 28
4.1 Inicializace ... 28
4.2 IDLE režim ... 29
4.3 Zpráva od nadřazeného PC ... 30
4.3.1 Struktura řídící zprávy ... 30
4.3.2 UART Interrupt Callback ... 30
4.4 Odpověď nadřazenému PC ... 31
4.4.1 Chybové byty ... 32
4.4.2 Skutečná a žádaná rychlost kol ... 32
4.4.3 Aktuální poloha servopohonů ... 33
4.4.4 Aktuální a žádaná poloha aktuátorů ... 33
4.4.5 Ostatní byty a kontrolní součet ... 33
4.5 CAN komunikace ... 34
4.5.1 Sestavení zpráv ... 34
4.5.2 Odesílání zpráv ... 37
4.5.3 Odpovědi od servopohonů ... 38
4.5.4 Odpovědi od aktuátorů ... 39
4.5.5 Vyčtení chyb ... 40
4.6 Režim KRAB ... 42
4.6.1 Určení potřebné rychlosti a natočení (KRAB) ... 42
4.6.2 Rampa natočení (KRAB) ... 43
4.6.3 Rychlostní rampa (KRAB) ... 43
4.7 Režim CIRCULAR ... 45
4.7.1 Získání úhlu β ... 45
4.7.2 Rampa natočení (CIRCULAR) ... 45
4.7.3 Rychlostí rampa (CIRCULAR) ... 46

4.8 Bezpečnostní prvky 47
4.8.1 Nouzové zastavení 47
4.8.2 Ochrana proti zacyklení 48
5 HARDWARE 49
5.1 Návrh přídavné desky 49
5.2 Porovnání hardwaru řídících jednotek 51
6 OTESTOVÁNÍ JEDNOTKY 52
6.1 Testování firmwaru 52
6.2 Testování hardwaru 52
7 ZÁVĚR 54
SEZNAM ZKRATEK 55
SEZNAM POUŽITÝCH ZDROJŮ 56
SEZNAM OBRÁZKŮ 58
SEZNAM TABULEK 59
SEZNAM PŘÍLOH 60

1 ÚVOD

Oblast autonomních vozidel zaznamenává velký rozvoj. V současnosti již není neobvyklé setkat se s osobním automobilem, který je, do jisté míry, schopný autonomního řízení. Myšlenka vozidla, které nepotřebuje řidiče je tu ale s námi už dlouho a ve vojenském odvětví průmyslu to platí dvojnásobně. Bezposádkové bojové vozidlo dokáže zamezit ztrátám na životech nebo podpořit pěchotu. Autonomní zařízení se dají označit jako budoucnost, nejen válečných konfliktů.

Právě vojenského průmyslu se týká předmět této práce. Jedná se o autonomní vozidlo TAROS, konkrétně jeho první generaci, jenž vytvořila Univerzita obrany ve spolupráci s firmou VOP CZ. Jedná se o demonstrátor možností bezposádkového bojového vozidla, nikoli o sériově vyráběný model nasazovaný v akci.

Práce se soustředí na nahrazení předchozí jednotky, u níž není přístup ke zdrojovému kódu a nelze tedy měnit funkcionalitu vozidla. Cílem je napsat nový firmware, který replikuje funkcionalitu předešlé jednotky, vylepšit nalezené nedostatky a starý hardware nahradit novým.

Řídící jednotka, kterou se práce zabývá, zprostředkovává rozhraní mezi nadřazeným počítačem plánujícím trasu a akčními členy (lineárními aktuátory, servopohony, manipulátorem a výsuvným sloupem). Jednotka tedy řeší zpracování příkazů a následnou komunikaci s pohony v souladu s nimi. Současně odesílá informace o současném stavu pohonů zpět do řídícího počítače.

Vozidlo s novou jednotkou může sloužit k výuce mobilní robotiky a k dalšímu výzkumu v oblasti autonomních vozidel. Současně nová jednotka umožní změnu a rozšíření funkcionality vozidla.

Prvotní osobní motivací k tvorbě práce byl zájem o sběrnici CAN, která mě zaujala během letní stáže, kde jsem viděl její skutečné použití při řízení brzdy kolejového vozidla. Zároveň byla možnost pracovat s autonomním vozidlem velice lákavá a zajímavá. Přestože jsem již na začátku věděl, že bude tvorba práce náročná, bral jsem ji jako osobní výzvu a možnost naučit se novým věcem, což mi umožní prohloubit a prakticky vyzkoušet teoretické znalosti nabyté při výuce.

2 REŠERŠE

Rešerše má za úkol přiblížit podstatné části této práce a vytvořit podklad pro některé další kapitoly. Obsahuje tři hlavní bloky. Prvním z nich představuje autonomní vozidlo TAROS, pro které se řídící jednotka vyvíjela. Druhý popisuje výpočet kinematiky vozidla využívaný při řešení cílů práce. Poslední blok rešerše se zabývá komunikační sběrnicí CAN, kde je zahrnut princip funkce, včetně struktury zprávy. Nakonec jsou zde popsány hlavní výhody CAN busu podtrhující, jak velký má v dnešní době význam.

2.1 TAROS 6x6

Středobodem této práce je, třínápravové vozidlo s pohonem 6x6, TAROS (Obrázek 1). Tento taktický automatizovaný robotizovaný systém je demonstrátorem autonomního pozemního robotizovaného vysoce automatizovaného systému širokého spektra procesů, včetně automatické navigace a činnosti ve složitém urbanizovaném nebo otevřeném terénu.

Vyvinula ho společnost VOP CZ ve spolupráci s brněnskou Univerzitou obrany. Jeho účelem je bojová a logistická podpora bojových jednotek v operačně složitém prostředí. Dokáže plnit řadu úkolů transportního, bojového a průzkumného charakteru. Jelikož se jedná o bezposádkové zařízení, může být nasazen i ve vysoce rizikovém prostředí. Vozidlo je koncipováno modulárně, s širokou řadou příslušenství z ranku robotických, zbraňových nebo senzorických systémů.

Z hlediska pohonu, podstatného pro tuto práci, je vozidlo osazeno 6 lineárními aktuátory, které slouží k natáčení kol. Akceleraci a pohyb vozidla má na starost 6 servopohonů, každý s výkonem 4,8kW. Další technické specifikace lze vidět v Tabulka 1.

Tabulka 1: TAROS – technické údaje

Technické údaje	
Délka	2 740 mm
Šířka	1770 mm
Výška	2040 ÷ 2540 mm
Výsuv sloupu	500 mm
Hmotnost	1050 kg
Počet náprav/kol	3/6
Rozchod	1 410 mm
Rozvor	(800 + 800) mm
Pohon	Elektrický/hybridní
Výkon	6 x 4,8kW

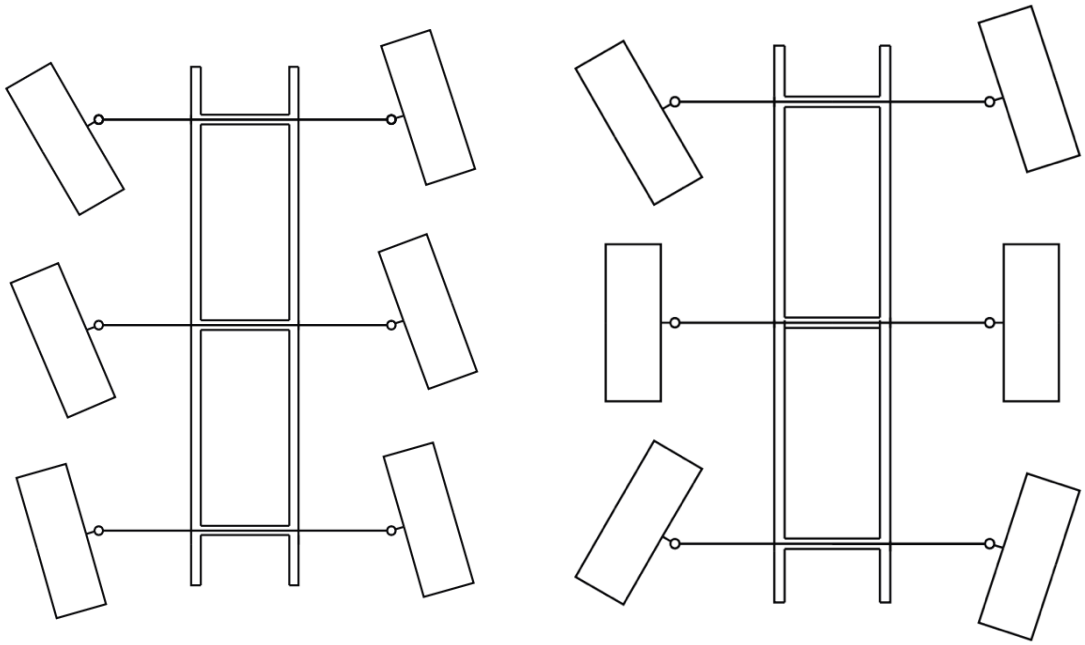
Obrázek 1: Autonomní vozidlo TAROS

2.2 Ackermannovo řízení

Jelikož má TAROS pohon 6x6 se třemi nápravami, využívají se dva režimy řízení. Prvním je *KRAB*, kdy mají všechna kola stejný úhel natočení a stejnou rychlost. Druhý, kruhový, režim *CIRCULAR* spočívá v protichůdném natáčení přední a zadní nápravy, zatímco prostřední náprava zůstává nenatočená. Konfigurace kol pro oba řídící režimy je zobrazena na Obrázek 2. K určení rychlostí a úhlů natočení kol pro kruhový režim je využívána teorie Ackermannova řízení.

Jde o jeden z několika druhů řízení, se kterým se setkáváme každý den v osobních automobilech. Stačí se podívat na vozidlo při parkování a lze si všimnout, že jedno kolo (konkrétně to vnitřní, které je blíže středu zatáčení) má výrazně větší úhel natočení než kolo vnější. Při malých poloměrech zatáčení, jako u zmíněného parkování, je rozdíl mezi úhly natočení kol největší. Naopak klesá při malých zásazích do řízení [1]. Požadavek pro tento děj vychází z faktu, že při průjezdu zatáčkou opisuje každé kolo kružnici o jiném poloměru (vnitřní kolo menší a vnější kolo větší), takže se musí otáčet jinou rychlostí. Rozdílným úhlem natočení kol a jejich rychlostí se zamezí bočnímu prokluzu při pohybu podél křivky.

Tento typ řízení byl vyvinut stavitelem koňských povozů Georgem Lankenspergerem v roce 1816 v Mnichově, ke zlepšení ovladatelnosti povozů. Nápad si ale nechal patentovat Rudolph Ackermann, po němž je pojmenovaný[2]



Obrázek 2: Režimy řízení (nalevo KRAB napravo CIRCULAR) převzatý z [16]

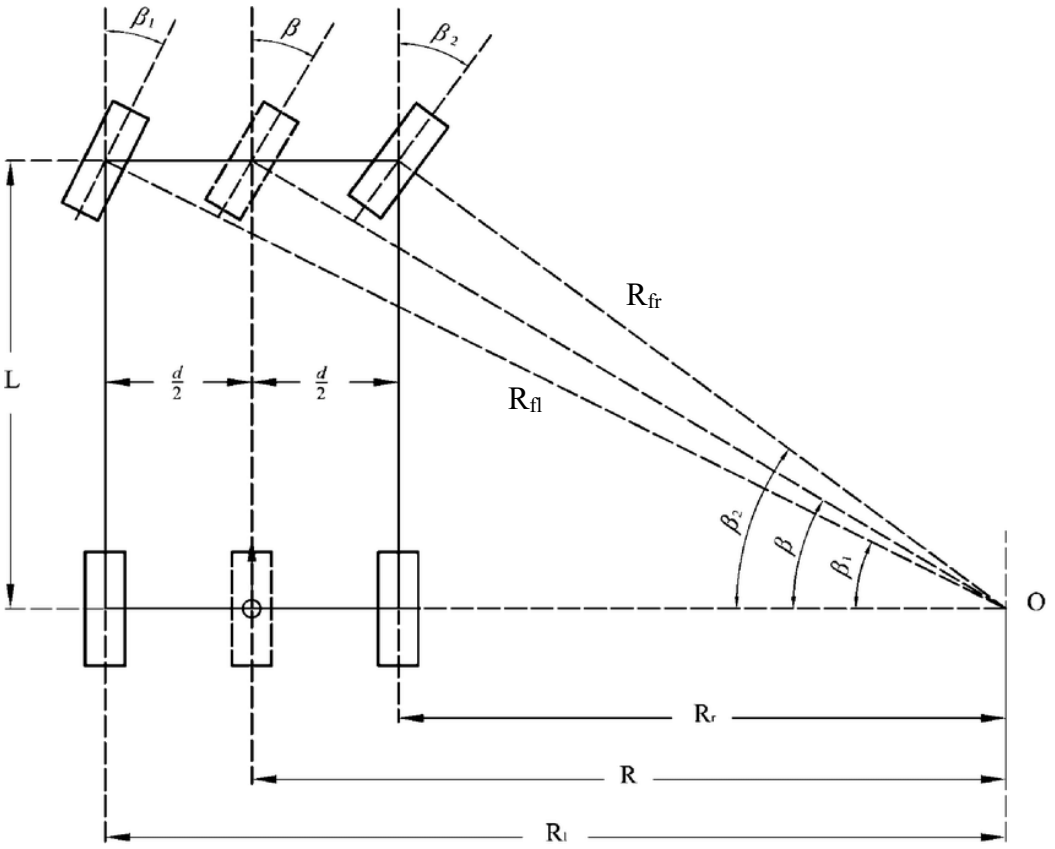
Aby se dal tento druh řízení použít, musí být splněna Ackermannova podmínka, která říká, že střed otáčení musí ležet na prodloužené ose neřízené nápravy [3]. Při natočení kol jejich osy stále směřují do středu otáčení. Ackermannův podvozek má typicky řídící mechanismus, skládající se ze dvou řídících pák a jedné spojovací tyče, poskládaných do tvaru lichoběžníku s otočnými rejdovými čepy v rozích.

Ackermannova geometrie platí přesně za předpokladu malých rychlostí a tuhých kol, díky tomu se využívá kupříkladu u nákladních robotů nebo u předmětu této práce, vojenského terénního robotizovaného vozidla.

Dynamické síly mohou způsobit, že se kola oproti žádané pozici vychýlí, což má za následek posunutí okamžitého středu otáčení mimo prodlouženou osu neřízené nápravy. Model použitý v této práci je čistě kinematický a působením dynamických sil se nezaobírá.

2.2.1 Výpočet Ackermannovy kinematiky

Pro výpočet potřebných úhlů natočení bylo zavedeno zjednodušení v podobě nahrazení obou kol nápravy pouze jedním kolem umístěným do středu této nápravy (viz Obrázek 3).



Obrázek 3: Ackermannova kinematika pro dvou podvozkové vozidlo, převzato z [4]

Obecně je k výpočtu potřeba znát:

- rozvor kol L
- rozchod kol d
- žádaný úhel natočení β

Z těchto vstupů lze vypočítat poloměr okamžitého středu otáčení pro prostřední kolo R (2.1) a kola krajní (R_r (2.2) a R_l (2.3)). Následující rovnice jsou převzaty z [4].

$$R = \frac{L}{\tan\beta} \tag{2.1}$$

$$R_r = R - \frac{d}{2} = \frac{L}{\tan\beta} - \frac{d}{2} \tag{2.2}$$

$$R_l = R + \frac{d}{2} = \frac{L}{\tan\beta} + \frac{d}{2} \tag{2.3}$$

Pro výpočet úhlu natočení levého (vnějšího) kola β_1 a pravého (vnitřního) kola β_2 je využita rovnice (2.4) a (2.5).

$$\beta_1 = \tan^{-1}\left(\frac{L}{R_l}\right) \tag{2.4}$$

$$\beta_2 = tan^{-1}\left(\frac{L}{R_r}\right) \tag{2.5}$$

Pomocí rovnic (převzatých z [5]) se vypočítají úhlové rychlosti zadních, hnacích kol.

$$\omega_{ml} = \omega.\left(1 + \frac{d}{2R}\right) \tag{2.6}$$

$$\omega_{mr} = \omega.\left(1 - \frac{d}{2R}\right) \tag{2.7}$$

Kde:

- ω_{ml} je úhlová rychlost levého hnacího kola
- ω_{mr} je úhlová rychlost pravého hnacího kola
- ω je úhlová rychlost prostředního aproximovaného kola na hnací nápravě:

$$\omega = \frac{v}{r} \tag{2.8}$$

Přičemž:

- *v* je dopředná rychlost vozidla
- *r* je poloměr kola

Jelikož jsou všechna kola poháněna, je nutné určit rychlost předních a zadních kol. Pro tečnou rychlost pravého předního kola v_{fl} platí vztah:

$$v_{fl} = \omega_{celk}.R_{fl} \tag{2.9}$$

Kde:

- R_{fl} je vzdálenost mezi okamžitým středem otáčení a osou levého předního kola, kterou lze vyjádřit z pravoúhlého trojúhelníku jako:

$$R_{fl} = \frac{L}{sin\,\beta_1} \tag{2.10}$$

- ω_{celk} je úhlová rychlost prostředního kola na hnací nápravě vzhledem k okamžitému středu otáčení:

$$\omega_{celk} = \frac{v}{R} \tag{2.11}$$

Dosazením (2.1) do (2.11) a následným dosazením (2.11) a (2.10) do (2.9) získáme výsledný vztah:

$$v_{fl} = v\frac{tan\,\beta}{sin\,\beta_2} \tag{2.12}$$

Pro úhlovou rychlost levého předního kola ω_{fl} tedy platí:

$$\omega_{fl} = \frac{v}{r}\frac{tan\,\beta}{sin\,\beta_2} \tag{2.13}$$

Stejným způsobem lze odvodit vztah pro úhlovou rychlost pravého předního kola ω_{fr}:

$$\omega_{fr} = \frac{v}{r}\frac{tan\,\beta}{sin\,\beta_1} \tag{2.14}$$

Pro zadní kola platí vztahy (2.13) a (2.14).

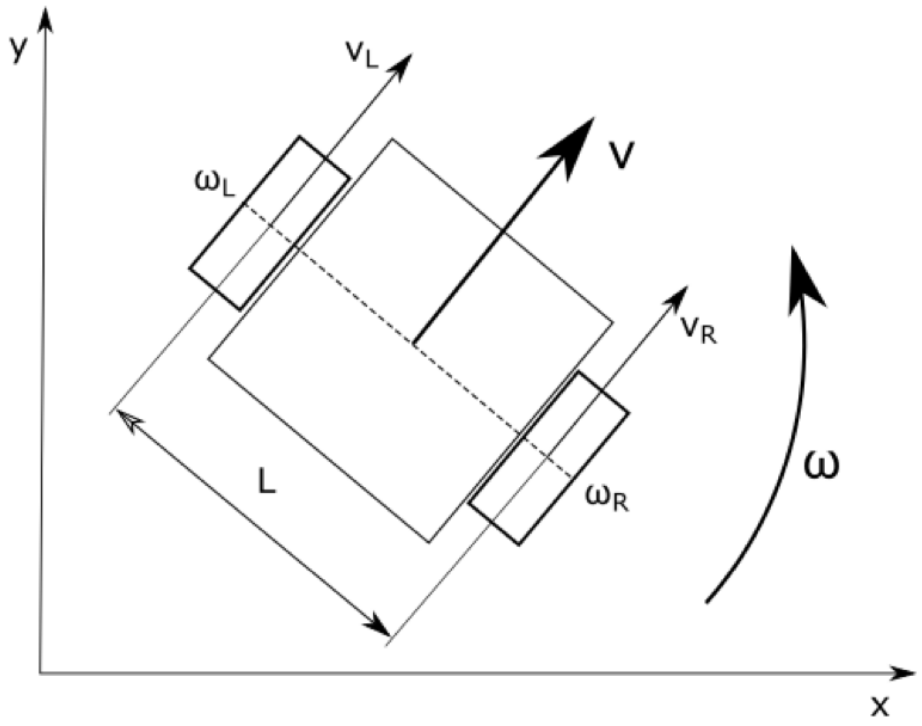
2.3 Diferenciální podvozek

Tento typ podvozku (znázorněný na Obrázek 4) má dvě poháněná kola, která jsou často doplňována všesměrovými přídavnými koly, nebo jinými opěrnými body zajišťujícími stabilitu. Podvozek se nazývá diferenciální, protože zatáčení je realizováno odlišnou rychlostí otáčení poháněných kol. Například pokud se pravé kolo robota s diferenciálním podvozkem otáčí větší rychlostí než to levé, tak začne robot zatáčet doleva.

Na vozidle TAROS není tento typ řízení přímo implementován, ale využívá se v kapitole 4.5.3 k výpočtu skutečné dopředné rychlosti pro kruhový režim, jelikož kola prostřední nápravy se sama o sobě nenatáčí, ale při průjezdu zatáčkou má vnitřní a vnější kolo odlišnou úhlovou rychlost.

Dopředná rychlost *v* se určuje pomocí rovnice ((2.15) převzaté a odvozené v [6]

$$v = \frac{r(\omega_l + \omega_r)}{2} \tag{2.15}$$

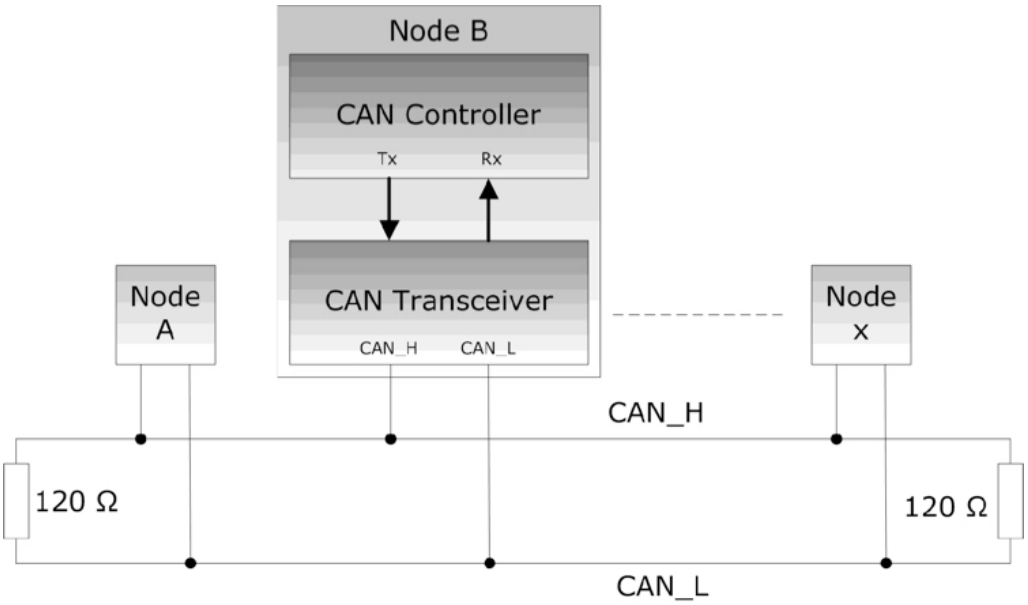


Obrázek 4: Schéma diferenciálního podvozku převzaté z [17]

2.4 CAN bus

CAN je komunikační protokol, který dovoluje vzájemně propojit dva a více elektronických řídících jednotek pouze dvěma vodiči, vytvářející jednu komunikační linku, přes kterou všechny ECU vzájemně komunikují, za předpokladu stejného baud rate. Nejjednodušší analogií k vysvětlení této technologie je nervový systém v lidském těle, protože všechny části jsou vzájemně propojeny po jedné „lince“ [7].

Hardwarová výbava je nenáročná (Obrázek 5), konkrétně jsou potřebné dva vodiče CAN High a CAN Low, ke kterým se nezávisle na sebe připojí potřebný počet uzlů. Každý uzel potřebuje svůj vlastní transceiver (kombinace slov transmitter a receiver), ten je propojen se svým vlastním nadřazeným CAN kontrolérem, který je většinou zabudován přímo do MCU[8]. Dále obvod je zakončen dvěma terminačními rezistory s odporem 120 Ω k uzavření linky [9]. Zpráva je poslána po vodičích a dostane se ke všem připojeným uzlům, které naslouchají a na základě nastaveného filtru se rozhodují, zda zprávu přijmou nebo ne. Každý uzel může zprávu vyslat a může zprávu i přijmout.



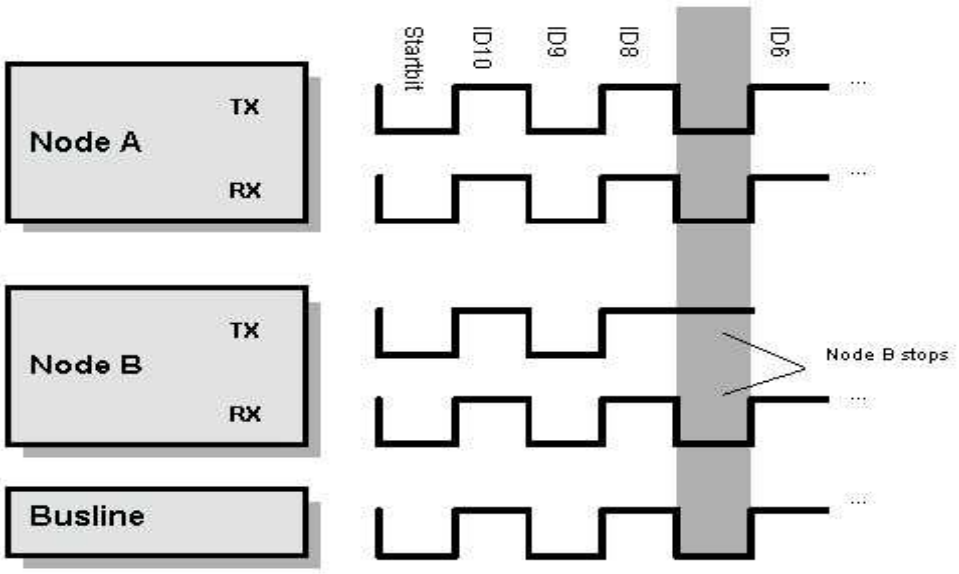
Obrázek 5: CAN bus – schéma zapojení převzaté z [18]

2.4.1 Princip funkce

Jde o takzvanou *Peer to Peer* síť. Tím pádem neobsahuje žádný nadřazený uzel (master) ani podřazený uzel (slave). Jak bylo zmíněno v předchozí kapitole, tak CAN využívá pouze jednu linku. Co se tedy stane, když chce více než jeden uzel vyslat zprávu? Protokol využívá nedestruktivní bitovou arbitrační fázi. Ta zajišťuje, že pokud vysílá zprávu jeden z uzlů, tak nemohou vysílat další, pokud jejich zpráva nemá vyšší prioritu nebo se linka neuvolní [10] . Jde tedy o Half Duplex systém.

K porozumění je nutné si definovat, že dominantní bit je v CAN protokolu 0 a recesivní bit je 1. Arbitrace funguje takto. Všechny uzly monitorují bity, které se pohybují po lince, což platí i pro jejich vlastní bity, které vysílají. Arbitrace se určuje na základě identifikátoru zprávy. Řekněme, že společně komunikují uzel A a uzel B (Obrázek 6), přičemž A má nižší číselnou hodnotu identifikátoru a B má vyšší. A a B vysílají své identifikační bity a dokud se tyto bity budou shodovat, vysílají dál, zatímco sledují, jestli jsou stejné bity, které vysílají, přítomny na lince. Tento proces přetrvává do chvíle, kdy A s nižší hodnotou ID, pošle dominantní bit, který přepíše recesivní ID bit B. Tím pádem B nedostane jako odpověď stejný bit, který poslal a

pozastavuje své vysílání, dokud se nedokončí odeslání zprávy s vyšší prioritou a linka se neuvolní.

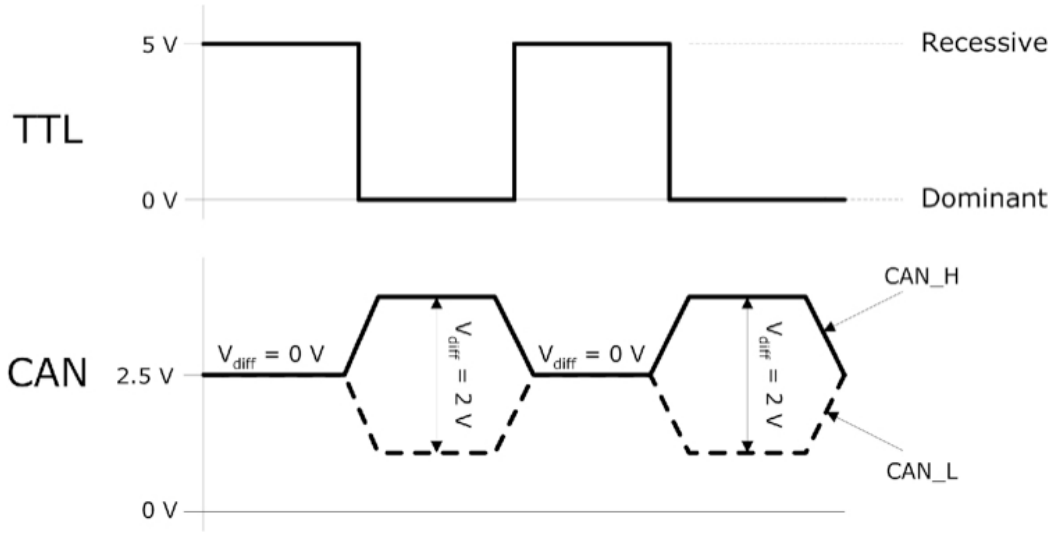


Obrázek 6: Arbitrační fáze, obrázek převzatý z [12]

Priorita je tedy určena identifikátorem zprávy, přičemž platí, že nižší číselná hodnota identifikátoru signalizuje vyšší prioritu [11].

Dalším stádiem komunikace je detekce chyb, například pomocí CRC bitů. Pokud je na nějakém uzlu detekována chyba, přesune se do pasivního režimu, dokud se neobnoví běžný chod komunikace – více o chybových stavech v kapitole 2.4.2 (podkapitola *Chybový rámec*).

K přenosu dat využívá diferenciálního signálu. To znamená, že dvojice vodičů (CAN_H a CAN_L) má vzájemně obrácenou polaritu signálu. Logická úroveň bitu se pak vyčítá z rozdílů těchto dvou signálů. Tyto úrovně jedna a nula jsou reprezentovány vysokou a nízkou úrovní napětí (TTL většinou 0 V a 5 V nebo 0 V a 3,3V, záleží však na použitém hardwaru). To má za následek výraznou odolnost vůči rušení a velkou spolehlivost, protože rozdíl napěťových úrovní je i při zarušení signálu stále stejný.



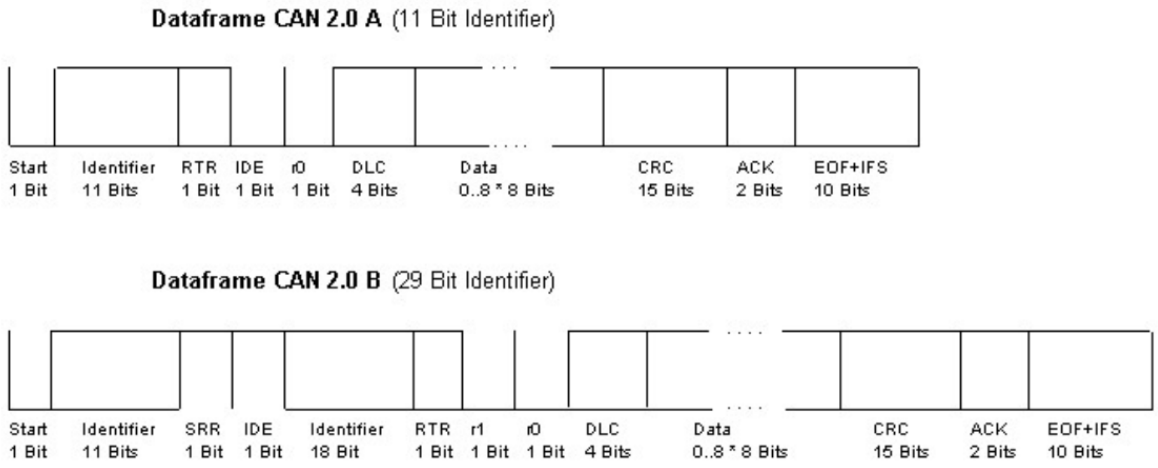
Obrázek 7: TTL a CAN signál, obrázek převzatý z [18]

2.4.2 Struktura zprávy

Komunikaci přes CAN sběrnici lze rozdělit na 4 rámce (anglicky frame): datový, vzdálený, chybový a rámec přetížení.

Datový rámec (Data frame)

Jak název napovídá, tento rámec slouží k odesílání dat. Může mít dva formáty – buď standardní nebo rozšířený. Jedná se o nejčastěji používaný CAN rámec. Tyto zprávy mají následující strukturu (Obrázek 8):



Obrázek 8: Datový rámec, obrázek převzatý z [12]

- Celá zpráva začíná SOF (*Start Of Frame*) bitem, který indikuje začátek zprávy a dává všem uzlům vědět, že mají sladit svou vzorkovací frekvenci.
- Po něm následuje arbitrační část, která obsahuje identifikátor. Délkou identifikátoru se liší standardní a rozšířený formát. Standardní identifikátor má délku 11 bitů, zatímco délka rozšířeného je 29 (11 základních a 18 rozšiřujících). Identifikátor také určuje prioritu zprávy.
- SRR (*Substitute Remote Request*) je bit používaný pouze v rozšířeném rámci, který se umístí na pozici RTR ve standardním, tedy za základní část identifikátoru a umožňuje při arbitraci porovnávat zprávy s oběma rámci mezi sebou. Musí být recesivní.
- IDE (*Identifier Extencion*) bit označuje, o jaký formát identifikátoru se jedná (dominantní bit pro standardní rámec a recesivní pro rozšířený). Je následovaný rozšiřujícími bity identifikátoru (pokud se jedná o tento formát).
- Za identifikátorem se nachází RTR (*Remote Transmisition Request*) bit odlišující vzdálený rámec od datového rámce. Logická nula indikuje datový rámec a logická jednička indikuje vzdálený rámec. Pokud by se na lince objevil datový a vzdálený rámec se stejným identifikátorem, tak má datový rámec vyšší prioritu.
- Bity r0 a r1 jsou rezervní, při standardním formátu se ve zprávě nachází pouze r0 a je dominantní. V rozšířeném formátu se nachází oba a jsou recesivní.
- DLC (*Data Length Code*) část nese informaci o počtu bytů v datovém poli. Datové byty dále obsahují samotnou zprávu, která může mít ve standardním CANu až 8 bytů (pro CAN FD je to až 64).
- Další částí je CRC (*Cyclic Redundancy Check*). Tato kontrola funguje na principu kontrolního součtu, který při přijetí zprávy vypočítá i přijímající uzel. CRC na lince

musí souhlasit s CRC vypočítaným přijímačem. Pokud bity touto kontrolou neprojdou, tak je hlášena CRC chyba[12].

- *Acknowledgement Field* (ACK) se stará o signalizaci správného přijetí zprávy nebo o případné vyžádání znovu odeslání zprávy.
- Konec zprávy se označuje jako *End of Frame* (EOF) a za ním je *Inter-frame spacing* (IFS), neboli pauza do odeslání další zprávy.
- Zpráva může obsahovat ještě vložené bity (Stuff bity). Pokud je totiž někde ve zprávě za sebou 5 stejných bitů, vloží se mezi ně tento bit, opačné polarity, tak aby nedošlo ke špatnému vyhodnocení dat. Hlavním důvodem jejich implementace je synchronizace. Sběrnice spoléhá na detekci náběžné a sestupné hrany k synchronizaci vnitřních hodinových signálů jednotlivých uzlů [13].

Vzdálený rámec (Remote frame)

Tento rámec se identifikuje tím, že RTR bit je překlopený do recesivní logické 1. Jedná se v podstatě o vyžádání datových bitů od uzlu s daným identifikátorem. Sám o sobě neobsahuje datovou oblast. Oproti datovému rámci je méně používaný.

Chybový rámec (Errror frame)

Využívá se, pokud nějaký z uzlů detekuje chybu. Zjednodušeně řečeno, dá o chybě vědět i ostatním uzlům sběrnice, které začnou také vysílat tento rámec. Chybový rámec porušuje standardní formát zprávy, což zaručí zničení všech datových a vzdálených rámců, dočasné přerušení komunikace a uvolnění linky. Zpráva, při které došlo k chybě, je poté znovu odeslána. Strukturou se dělí na dvě části: *Error Flags* o velikosti 6 bitů a *Error Delimetres*, tedy oddělovač, o velikosti 8 recesivních bitů. První zmíněný signalizuje, že daný uzel zaznamenal chybu. Může dojít celkem k pěti typům chyb:

- *Bit Error* – k této chybě dochází, pokud není odeslaná zpráva stejná jako zpráva přijatá.
- *Stuffing Error* – pokud zpráva obsahuje více než 5 bitů stejné logické hodnoty, tak je zahlášena tato chyba. K předcházení této chyby slouží stuff bity (rozebrané v kapitole 2.4.2, podkapitola *Datový rámec*), které takovéto vzorce „rozbíjí“.
- *Form Error* – chyba je detekována v případě, že není dodržena daná struktura zprávy.
- *CRC Error* – při vysílání zprávy je na základě dat ve zprávě vypočítána hodnota CRC o velikosti 15 bitů a vložena do zprávy. Uzel přijímající zprávu CRC opět vypočítá a pokud se kontrolní součet odlišuje, tak z toho vyplývá, že zpráva byla během přenosu poškozena nebo nějakým způsobem upravena [14].
- *ACK Error* – ACK bit má být u vysílané zprávy recesivní (logická 1) a u odpovědi od přijímače dominantní (logická 0). Tím je signalizováno, že zpráva dorazila. Pokud není tato konvence dodržena, je zahlášena ACK chyba.

Na základě počtu chyb se CAN kontrolér přepíná mezi dvěma stavy. K určení stavu slouží dvě počítadla TEC (*Transmission Error Counter*) a REC (*Receive Error Counter*). Jsou určité podmínky určující, jak moc bude uzel tyto čítače inkrementovat nebo dekrementovat [9]. Prvním ze stavů je *Error Active*, výchozí stav. V tomto režimu při detekci chyby vysílá kontrolér 6 dominantních bitů k přerušení komunikace. Hranice přechodu do dalšího stavu je pro TEC i REC stejná, konkrétně 127. Po překročení této hranice přechází do *Error Passive* stavu, během kterého při zaznamenání chyby vysílá 6 recesivních bitů, což má za následek, že tento uzel nemůže přerušit komunikace na lince. Pokud oba chybové čítače překročí hranici 255, dojde k enormnímu nahromadění chyb, a daný uzel se odpojí od sítě [14].

Rámec přetížení (Overload frame)

Tento rámec uzel generuje v případě, že dojde k přetížení uzlu. Struktura je stejná jako u chybového rámce, obsahuje tedy *Overload flag* (6 bitů) a *Overload delimiter* (8 bitů). Jeho funkce je v podstatě vytvoření časové mezery tak, aby uzel stíhal zprávy zpracovávat. Moderní CAN kontroléry tento rámec v podstatě nepoužívají. Využívá ho například původní kontrolér Intel 82526 [8].

2.4.3 Výhody CAN busu

První výhodou je jeho jednoduchost a dostupnost. Již před více než 50 lety se v automobilech začaly objevovat elektronické čipy a s neustálým vývojem technologií jejich počet stále rostl. V dnešním průměrném automobilu najdeme alespoň 40 elektronických řídících jednotek (ECU – *Electronic Control Unit*), záleží však na výbavě. V prémiovém vozidle s maximální výbavou jich můžeme najít až 150. Dříve, když musely být všechny komunikující jednotky propojeny navzájem vodiči, docházelo k velkým, nákladným, těžkým a omezujícím komunikačním spojům. Tento problém vyřešil CAN, jelikož všechny uzly se připojí na jednu linku a více už se nemusí navzájem propojovat. To má za následek i mnohem jednodušší diagnostiku systému. Také se snížily náklady (čipy s CAN jsou dostupné) a váha, protože dřívější vozidla obsahovala kilometry izolovaných vodičů.

Další podstatnou výhodou je vysoká odolnost vůči rušení. Za to vděčí CAN svému diferenčnímu signálu. Pokud jsou logické úrovně odečítány klasicky, tak může vlivem rušení přijímač špatně rozeznat napěťovou úroveň. To se ale u CAN sběrnice nemůže stát, jelikož pokud se zaruší, tak se šum objeví jak na CAN_H, tak i na CAN_L, ale rozdíl těchto dvou napěťových úrovní zůstane pořád stejný. Tato odolnost dělá CAN ideálním pro automotive průmysl, jelikož vlivem ostatních součástek, reproduktorů a dalších externích vlivů by mohlo snadno dojít k zarušení zpráv. Spolehlivost této sběrnice zaručuje také detekce chyb realizovaná chybovým rámcem, rozebraným v kapitole 2.4.2 *Chybový rámec*. Důležitou výhodou je zároveň rychlost přenosu, která je u CAN nadstandardní (až 1 Mbit/s při maximální délce vodiče 40 m)[15].

3 Analýza předešlé řídící jednotky

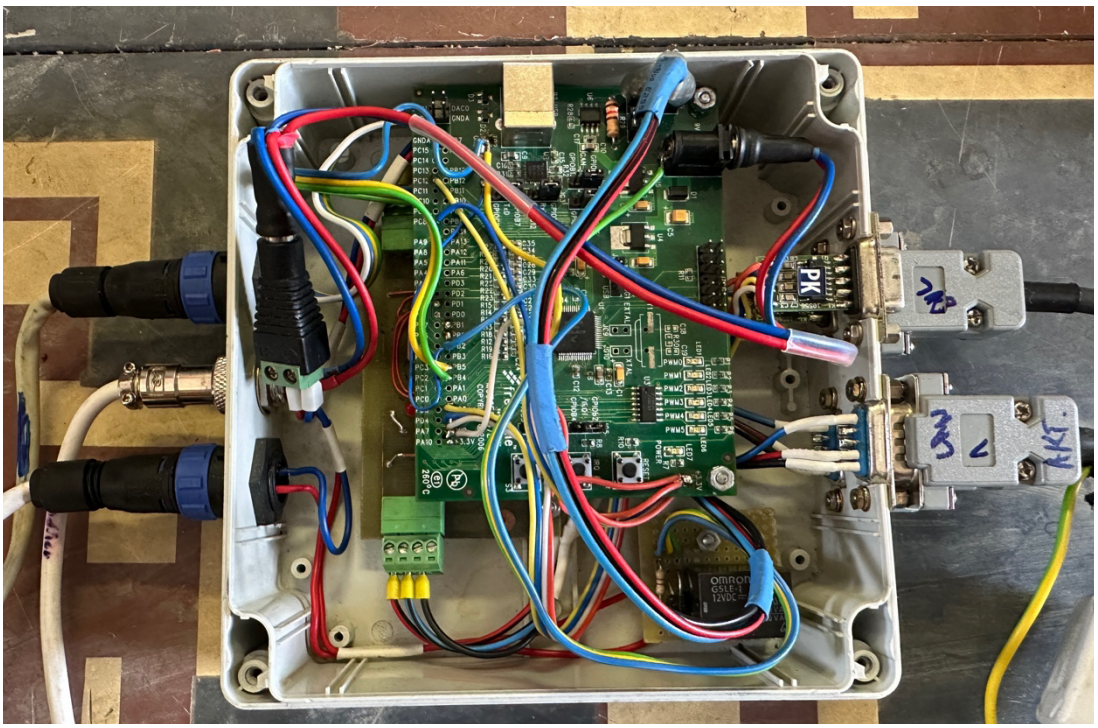
Tato práce spočívá v nahrazení (modernizaci) současné jednotky vozidla, u které, mimo jiné, není přístup ke zdrojovému kódu, tím pádem se její funkcionalita nedá nijak modifikovat. Pro její úspěšné nahrazení bylo potřeba předešlou jednotku zanalyzovat, což je také jedním z cílů této práce.

3.1 Součásti jednotky

Nejprve je nutné definovat, jaké součásti jednotka obsahuje. Jejím srdcem je deska s mikrokontrolerem MC56F8037, kterou lze vidět na Obrázek 9 . Deska je napájena ze stejnosměrných 12 V. Jednotka obsahuje převodník RS232 na TTL připojený k UART, který umožňuje přijímat a odesílat zprávy nadřazenému PC.

Další částí je CAN sběrnice. Deska obsahuje pouze jednu CAN bus linku, ve vozidle jsou však potřeba dvě. Z tohoto důvodu je CAN hlavní desky vyveden na další pomocnou, separátně napájenou desku, která jednu linku rozdělí na dvě.

Součástí jednotky je také ovládání sloupu vozidla. Ten je vyroben z výsuvného kancelářského stolu a o jeho spínání se starají dva obvody s 12V relé *KUAN S1A120000*. Tyto obvody dále obsahují bipolární tranzistor, bázový odpor, diodu a LED diodu signalizující, které relé je v daný moment sepnuté. Jedno zajišťuje vysouvání sloupu, druhé jeho zasouvání. Ve stavu, kdy není ani jedno relé sepnuté se sloup nepohybuje. Spínací signál vychází z GPIO pinů mikrokontroleru.

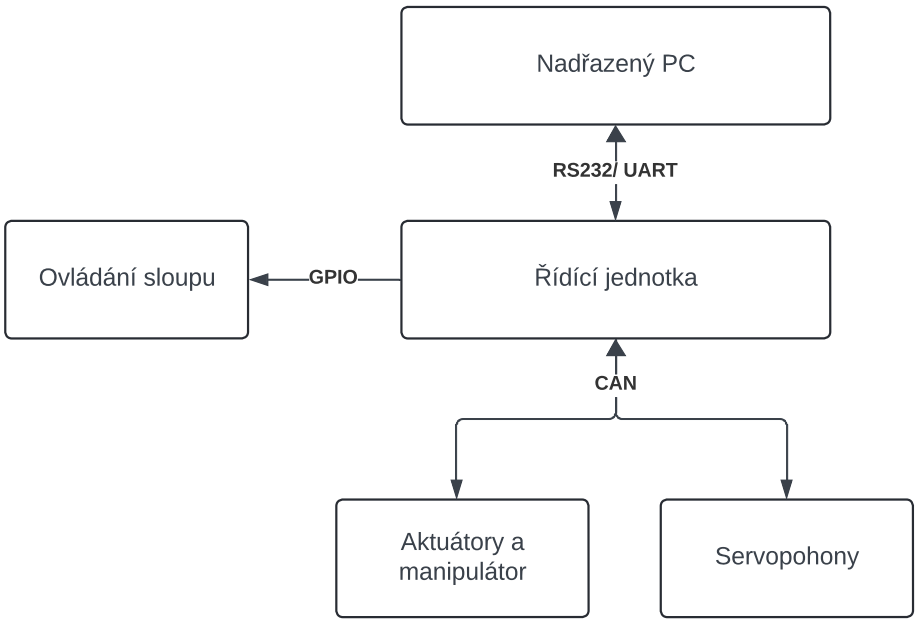


Obrázek 9:Předešlá řídící jednotka

Poslední částí jednotky je další obvod s relé, tranzistorem a odporem. Je také připojený na GPIO piny desky. U tohoto obvodu nebylo jasné, jaká je jeho funkce ve vozidle. Pokusy o zjištění této funkce, za různých příkazů vozidlu, neuspěly. Proto bylo potřeba se obrátit na firmu VOP. Ta dodala informaci, že slouží k ovládání laserových skenerů, které již na vozidle nejsou, tím pádem se nebude objevovat v nové řídící jednotce.

3.2 Vstupní a výstupní signály

Pro lepší pochopení této kapitoly je třeba specifikovat, jak jednotka principiálně funguje (schéma zobrazeno na Obrázek 10). Tento princip je stejný pro předešlou i novou jednotku. Nadřazený PC odesílá po sériové lince RS232, převedené na TTL logiku, řídící zprávu, jejíž přesná struktura je popsána v kapitole 4.3.1. Jednotka data zpracovává a odesílá dané pokyny akčním členům po CAN bus a zároveň ovládá výsuvný sloup.



Obrázek 10: Schéma principu jednotky

Strukturu většiny zpráv bylo možno vyčíst z návodů poskytnutých vedoucím práce. Pro odposlechnutí reálných zpráv byl využit adaptér CAN na USB, který se připojil do vstupní svorkovnice desky rozdělující CAN sběrnice. Převodník umožňuje odposlech a odeslání zprávy. Většina zpráv se držela návodu, ať už se jednalo o příkazy nebo odpovědi od pohonů.

Posledním výstupním signálem je odpověď řídícímu počítači, převedená z UART na RS232, jejíž struktura je blíže specifikovaná v kapitole 4.4.

3.3 Vstupní a výstupní konektory

Předešlá řídící jednotka má dva vstupy. Prvním z nich je sériová linka RS232 s 9pinovým D-sub konektorem, připojená k převodníku na UART, jenž přivede zprávu na *Rx* pin desky MC56F8037. Je zde ještě jeden totožný RS232 konektor s převodníkem pro GPS, ten už se ale nevyužívá.

Druhý vstup je napájecí napětí 12 V, přivedeno 4pinovým kulatým konektorem se závitem M12. Tato napájecí linka se dále dělí do třech větví. Jedna napájí mikrokontroler, další přídavnou desku rozdělující jeden CAN sběrnice do dvou. Poslední je zdrojem pro obvod ovládání sloupu vozidla, jenž obsahuje 12 V relé.

Co se týká výstupů, tak jednotka má dva CAN busy, oba vyvedené pomocí 9pinového D-sub konektoru, jehož standardní zapojení lze vidět na Obrázek 11. Sběrnice je zde ale zapojena netradičně, viz Tabulka 2.

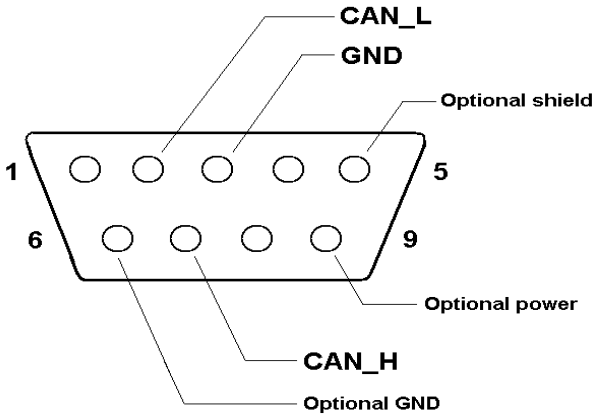
Tabulka 2: Zapojení CAN pinů

CAN pin	Tradiční zapojení	Zapojení na předešlé jednotce
CAN_H	2	2
CAN_L	7	5
GND	3	1

První z CAN busů zprostředkovává komunikaci s aktuátory a manipulátorem. Jeho baud rate je 1 Mbit/s. Přes druhý jednotka odesílá a přijímá zprávy z aktuátorů při baud ratu 250 kbit/s.

Z jednotky je dále vyveden výstup relé pro pohyb sloupu. K tomu je využit 4pinový konektor BULGIN PX0412/S.

Deska má ještě jeden výstup k ovládání laserových skenerů, který se ale už nepoužívá a v nové jednotce není.

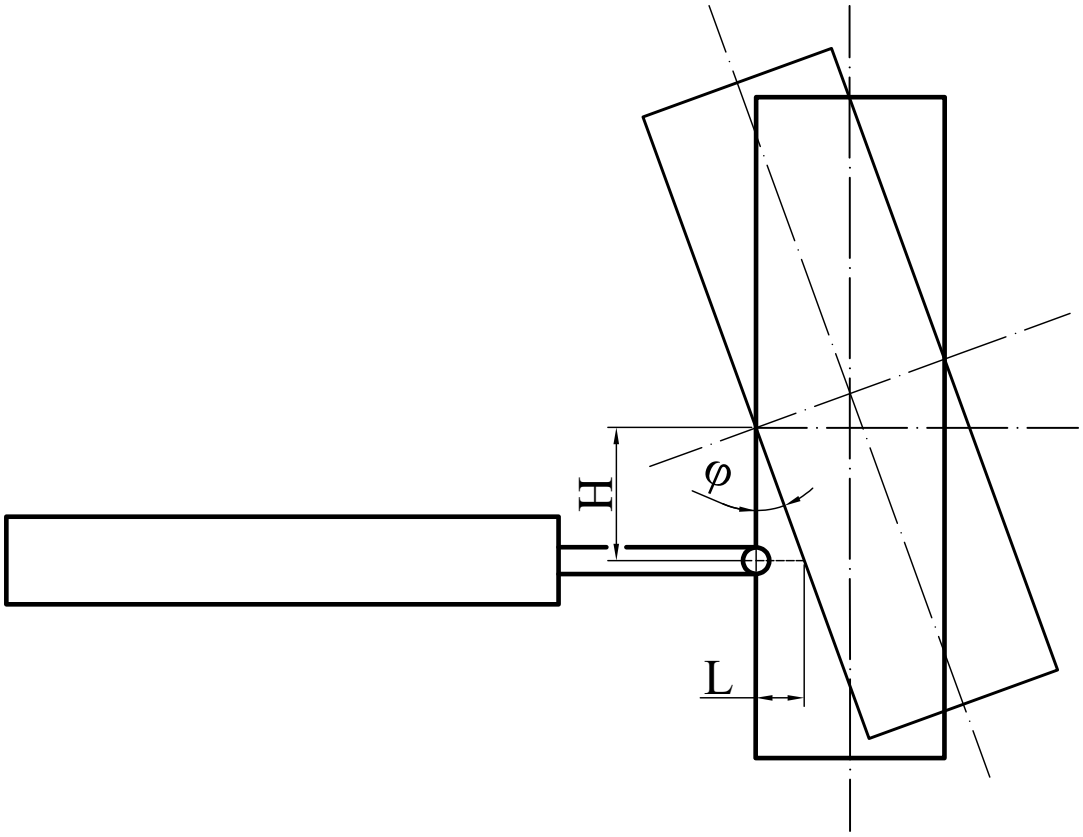


Obrázek 11: Tradiční zapojení CAN busu, obrázek převzatý z [19]

3.4 Určení maximálního úhlu natočení

Ve zprávě od PC je řídící úhel v rozsahu ±512 bez jakékoli jednotky. Pro správné řízení vozidla bylo nutné určit přepočet na reálný úhel ve stupních. Další otázkou bylo, jestli je úhel natočení kol přímo úměrný vysouvání aktuátorů. Jako první varianta bylo zvoleno, že závislost je vskutku lineární. Pro přepočet je ale potřeba znát maximální úhel, o který se kola natočí jak do kladného, tak do záporného směru, pokud tedy tyto úhly nejsou stejné. Nejjednodušší způsob, jak úhly zjistit, bylo měření. Konkrétně tedy měření změny jedné strany *L* trojúhelníku z Obrázek 12, kdy se vysouváním aktuátoru (natáčením kola) délka této strany měnila, zatímco strana *H* zůstávala stejná. Ze vztahu (3.1) se poté úhel dopočítal. Na aktuátory se přes CAN převodník z osobního počítače zadávaly jednotlivé vysunutí aktuátoru, vždy tři ekvivalentně vzdálená vysunutí v kladném i záporném rozsahu. U všech kol vycházela tato vysunutí stejně s tím, že maximální úhel je 17° a to jak v kladném, tak i záporném směru. Nutno podotknout, že tento způsob měření byl nepřesný kvůli špatné přístupnosti k samotnému kolu, jednalo se ale o nejjednodušší variantu. Následné testy však ukázaly, že tento maximální naměřený úhel dostatečně odpovídá skutečnosti.

$$\varphi = tan^{-1}\frac{L}{H} \tag{3.1}$$



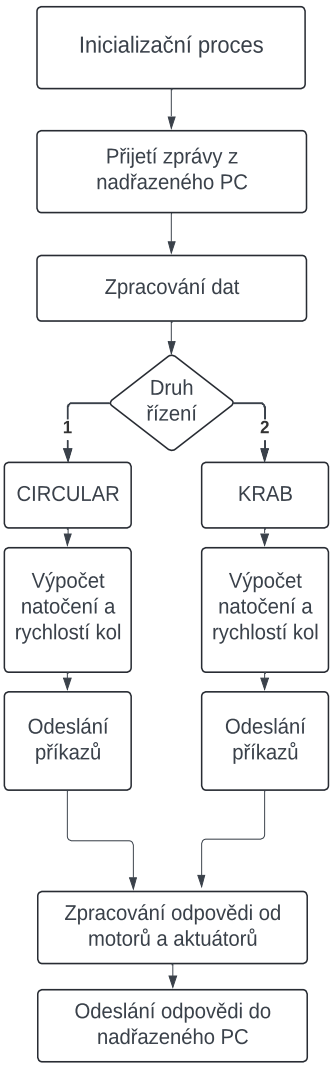
Obrázek 12: Schématické znázornění měření maximálního úhlu

Zároveň z rovnoměrného vysouvání aktuátoru (při zadání poloviny kladného rozsahu se kolo natočilo o polovinu maximálního úhlu, stejně pro čtvrtinu, polovinu atd.) vyplynulo, že natáčení se dá aproximovat lineárně s dostatečnou přesností.

4 Firmware

Úkolem jednotky je přijmout zprávu z nadřazeného počítače vozidla, jenž plánuje trasu, tuto zprávu zpracovat a odeslat příkazy na dané akční členy (lineární motory, servopohony, sloup a na něm posazený manipulátor). Dále musí jejich odpovědi zformulovat do zprávy, kterou odesílá po RS232 zpět do PC a informovat ho o současném stavu vozidla. S tím souvisí také bezpečnostní prvky, jmenovitě zastavení při ztrátě komunikace. Těmto funkcím je věnována kapitola 4.8.

Splnění všech těchto úkolů má na starost právě firmware jednotky psaný v jazyce C. Struktura firmwaru je schematicky znázorněna na Obrázek 13.



Obrázek 13: Schéma funkce firmwaru

4.1 Inicializace

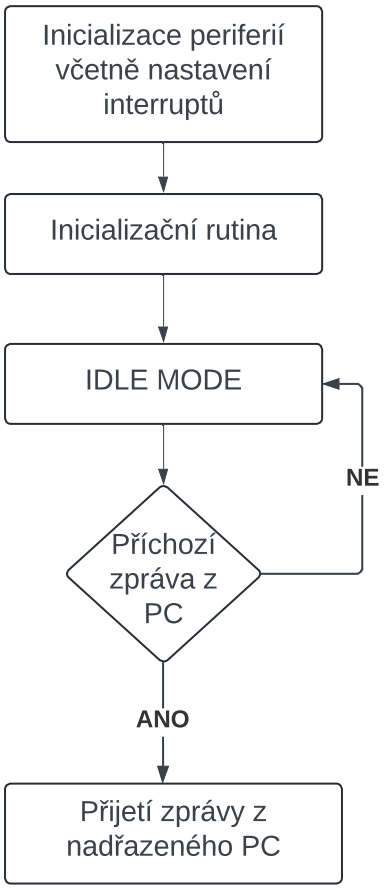
První, co se po spuštění programu stane, je inicializace všech používaných periferií (CAN, UART, GPIO piny atd.), nadefinují se CAN filtry a potřebná přerušení (CAN, UART, TIMER).

Následuje inicializace vozidla (Obrázek 14). Nejprve se počká 3,5s a začíná první fáze, kde script odešle zprávu, která deaktivuje drivery servopohonů. Další zpráva vynuluje všechny možné chyby, které mohly být zaznamenány při předchozím chodu

vozidla. Následuje zpráva, jež nastaví na servopohonech rychlostní režim, tedy, že motory budou řízeny na základě požadované rychlosti (dále je možné odesílat například požadovanou polohu, ale tato možnost není v řídící jednotce využita). Ihned po tom se odešle zpráva nulové rychlosti a příkaz na aktivaci driveru.

Za dalších 3,5 s proběhne druhá fáze inicializační rutiny, kde se v tomto pořadí odešle nulová rychlost, vyčtení rychlosti, vyčtení pozice, spuštění driveru, vymazání chyb, nastavení rychlostního režimu, opět nulová rychlost, spuštění driveru a vyčtení chyb. Nakonec se program přesune do IDLE režimu.

Tato inicializační rutina v nové jednotce vychází z předchozí jednotky. Rozdílem je že u té se první fáze opakovala dvakrát po sobě a prodleva byla 7 sekund. Nová jednotka i s těmito změnami funguje stejně jako ta první, jde tedy o zkrácení inicializačního procesu.



Obrázek 14: Inicializační proces

4.2 IDLE režim

Jak již název napovídá, tak je to režim, ve kterém TAROS čeká na další příkazy a setrvává v něm, dokud je neobdrží. Jediná chvíle, kdy se do tohoto režimu dostane (bez restartu kontroléru) je po inicializaci, před příchodem první řídící zprávy. Vozidlo v IDLE režimu setrvává v klidu. Na lineární pohony je třikrát po sobě odeslána střední pozice jejich rozsahu, kola se tedy nenatáčí. Stejně tak se nepohybuje sloup, ani manipulátor. Zároveň se odesílá nulová rychlost, takže se kola ani neotáčí. Jednotka neodesílá žádnou odpověď řídícímu počítači. Tento režim je přerušen přijetím zprávy z nadřazeného PC.

4.3 Zpráva od nadřazeného PC

Komunikace s nadřazeným počítačem probíhá přes sériovou linku RS232. Instrukce jsou převedeny na TTL logiku a jednotka je přijímá skrze UART. Vždy na začátku kroku svého pracovního cyklu jednotka přijme řídící zprávu a na konci odesílá odpověď.

4.3.1 Struktura řídící zprávy

Zpráva od nadřazeného PC obsahuje 13 bytů a má následující strukturu (Tabulka 3):

Tabulka 3: Struktura řídící zprávy

Byte	Funkce
Byte 0	Identifikátor (konstantní hodnota 255)
Byte 1	Zapnutí/vypnutí pohonu
Byte 2	Žádaná rychlost kol (vyšší byte)
Byte 3	Žádaná rychlost kol (nižší byte)
Byte 4	Druh řízení (KRAB/CIRCULAR)
Byte 5	Žádaný úhel natočení kol (vyšší byte)
Byte 6	Žádaný úhel natočení kol (nižší byte)
Byte 7	Frekvence manipulátoru v ose x
Byte 8	Frekvence manipulátoru v ose y
Byte 9	0
Byte 10	Vysouvání/zasouvání sloupu, restart servopohonů
Byte 11	2
Byte 12	Kontrolní součet

Přičemž žádaná rychlost kol je v km/h s rozsahem ± 20 km/h. Typ řízení může být buď 2 (režim KRAB, kapitola 4.6) nebo 1 (režim CIRUCULAR, kapitola 4.7). Žádané natočení má rozsah +-512, ale ve zprávě je navýšeno o 512 neboli rozsah 0 ÷ 1024. Frekvence os manipulátoru mají obě rozsah ± 100. Ve chvíli, kdy dorazí zpráva a je úspěšně přijata se spustí Interrupt Callback.

4.3.2 UART Interrupt Callback

Nejprve se znovu aktivuje *Receive Interrupt*, aby se mohl Callback při přijetí další zprávy znovu spustit. Poté se vyčte rychlost a úhel natočení, více o rychlosti a úhlu natočení pro jednotlivé režimy v kapitola 4.6.1 a 4.7.1 .

V programu jsou využívány dvě varianty řídícího úhlu, jedna snížená o 512, tedy v rozsahu -512 ÷ 512. Tato hodnota se ukládá do proměnné s názvem *steering_angle_ackermann* a je využívána především, pokud se jedná o kruhový režim s Ackermannovým řízením a všude, kde je potřebné vědět, jestli je úhel natočení kladný nebo záporný. Využívá se také v rampě natočení pro kruhový režim, kde se z ní dopočítává žádaný úhel v plném rozsahu, více v kapitole 4.7.2. Úhel v rozsahu 0 až 1024 je uložen v proměnné *steering_angle.* Tato varianta je vstupem do vztahu (4.9).

Z příchozí zprávy se rovněž vyčte typ řízení, data pro manipulátor (více o nich v kapitole 4.5.1), pohyb sloupu a případný restart servopohonů, který spustí stejnou rutinu jako při inicializaci těchto motorů.

4.4 Odpověď nadřazenému PC

Stejně jako příchozí zpráva od nadřazeného PC, tak i odpověď pro něj má přesně danou strukturu, jinak by PC nebylo schopno data správně zpracovat. Tato odpověď obsahuje až 118 bytů se strukturou dle Tabulka 4.

Tabulka 4: Struktura řídící zprávy

Byte	Funkce
Byte 0	Konstantní hodnota 255
Byte 1	Počet bytů ve zprávě
Byte 2	Skutečná rychlost kol – vyšší byte
Byte 3	Skutečná rychlost kol – nižší byte
Byte 4	Žádaná rychlost kol– vyšší byte
Byte 5	Žádaná rychlost kol– nižší byte
Byte 6 ÷ 9	Skutečná poloha servopohonu 1
Byte 10 ÷ 13	Skutečná poloha servopohonu 2
Byte 14 ÷ 17	Skutečná poloha servopohonu 3
Byte 18 ÷ 21	Skutečná poloha servopohonu 4
Byte 22 ÷ 25	Skutečná poloha servopohonu 5
Byte 26 ÷ 29	Skutečná poloha servopohonu 6
Byte 30 ÷ 31	Skutečná poloha aktuátoru 1
Byte 32 ÷ 33	Žádaná poloha aktuátoru 1
Byte 34 ÷ 35	Skutečná poloha aktuátoru 2
Byte 36 ÷ 37	Žádaná poloha aktuátoru 2
Byte 38 ÷ 39	Skutečná poloha aktuátoru 3
Byte 40 ÷ 41	Žádaná poloha aktuátoru 3
Byte 42 ÷ 43	Skutečná poloha aktuátoru 4
Byte 44 ÷ 45	Žádaná poloha aktuátoru 4
Byte 46 ÷ 47	Skutečná poloha aktuátoru 5
Byte 48 ÷ 49	Žádaná poloha aktuátoru 5
Byte 50 ÷ 51	Skutečná poloha aktuátoru 6
Byte 52 ÷ 53	Žádaná poloha aktuátoru 6
Byte 54 ÷ 105	V nové jednotce se nevyužívají
Byte 106 ÷107	Stav chyb pohonů
Byte 108 ÷116	Specifikace chyb pohonů (nemusí v odpovědi být)
Poslední byte	Kontrolní součet

4.4.1 Chybové byty

Nejprve je nutné se podívat na tyto byty na konci zprávy, jelikož se od nich odvíjí její délka (byte 1). Byte 106 a 107 jsou vyhrazeny pro specifikaci, na kterém aktivním členu došlo k jakékoliv chybě. Bit 0 až 5 signalizuje chybový stav na servopohonech (bit 0 pro servo 1, bit 1 pro servo 2 a tak dále). Bit 10 až 12 je vyčleněn pro aktuátory, respektive nápravy (bit 10 pro přední nápravu, bit 11 pro prostřední nápravu a tak dále). Toto je v programu řešeno maskou, která bitovým AND propíše 1 na daný bit.

Byty 108 až 116 jsou určeny pro sdělení konkrétní chyby, ke které došlo (přesněji popsáno v kapitole 4.5.5), přičemž byte 108 až 113 je určen pro servopohony a byte 114 až 116 pro aktuátory.

Na jaký bit se která chyba servopohonů propíše lze vidět v Tabulka 13. Když ale nedojde k žádné chybě, tak se byty 108 až 116 ve zprávě vůbec neobjeví. Zároveň, pokud by se například v odpovědi od servopohonu 5 (byte 112) objevila nějaká chyba (a pouze na tomto akčním členu), tak budou ve zprávě byty 108 až 111 (jejich hodnota bude 0x00), byte 112 (s příslušným chybovým hlášením), byte 113 (a další, kromě kontrolního součtu) však ve zprávě nebudou. Tato proměnná délka je ošetřena tím, že se odesílá vždy jen tolik bytů, kolik je potřeba. Prvotní délka zprávy je vždy maximální (118 bytů). Skutečná délka se odvíjí od proměnné *error_index*, která nabývá hodnotu 0 až 9, podle toho, na kterém pohonu (servo či aktuátor) došlo k chybě, čímž se mění počet odesílaných bytů. Například, dojde-li k chybě na servu 4, *error_index* nabude hodnoty 4 a délka odesílané zprávy se spočítá dle (4.1). Pro servo 4 je vyčleněn byte 111, přičemž poslední byte je vždy kontrolní součet, proto se odešle 112 bytů.

$$délka\ zprávy\ =\ maximální\ délka - 10 + error_{index} \tag{4.1}$$

4.4.2 Skutečná a žádaná rychlost kol

Pro skutečnou rychlost kol jsou vyčleněny byty 3 a 4. Jejich výpočet se liší podle řídícího režimu:

- V krabím režimu se odesílá aktuální rychlost předního pravého kola (při pohledu zepředu), která je z otáček za minutu přepočítaná zpět na dopřednou rychlost v km/h. Před rozložením na dva byty se však musí vynásobit 100 a k hodnotě přičíst 2000. Jedná se o opačný proces k získávání rychlosti od nadřazeného PC.
- V kruhovém režimu je vypočítána výsledná diferenciální rychlost z úhlových rychlostí prostřední nápravy dle (2.15).

Finální hodnota se rozloží na vyšší a nižší byte – nižší byte posunutím hodnoty o 8 bitů doprava a vyšší bitovým AND s hodnotou 0xff.

V obou řídících režimech se do odpovědi do bytu 5 a 6 rovnou propíše neupravená žádána hodnota, ze zprávy od počítače, jelikož ta už je ve formátu potřebném do odpovědi.

Ani jeden z výše zmíněných bytů (byty 2 až 5) se nesmí rovnat 255, protože je to hodnota 1. bytu odpovědi a počítač by pak mohl chybně rozeznat začátek zprávy. Ve scriptu je tento požadavek ošetřen *if* podmínkou, která případný byte nahradí hodnotou 254.

4.4.3 Aktuální poloha servopohonů

Byty 6 až 29 jsou určeny aktuálním polohám servopohonů, přičemž pro každý motor jsou vyčleněny 4 byty. Současné natočení každého kola se ze čtyř bytů zkompletuje do jednoho integeru, ke kterému se přičte konstanta $255^4/2$. Výsledek se opět rozloží na 4 byty pro odpověď. Nejvyšší byte je bitově posunut doprava o 24 pozic, druhý nejvyšší o 16 pozic a stejným způsobem až do nejnižšího bytu. Tato procedura je stejná pro oba řídící režimy.

4.4.4 Aktuální a žádaná poloha aktuátorů

Stejně jako servopohony, tak i každý aktuátor má 4 byty v odpovědi (byte 30 až 53). Do prvních dvou bytů se ukládá aktuální poloha, ta je vyčtena z CAN z odpovědi dané nápravy (kapitola 4.5.4) a přepočítá se zpět na původní rozsah ±512 ((4.5) nebo (4.6)). Před rozložením na vyšší a nižší byte se přičte 512, tedy nový rozsah je 0 až 1024, odpovídající příchozí zprávě od nadřazeného PC. Tímto způsobem se aktuální poloha aktuátorů určuje v obou režimech.

Určování žádané poloha aktuátorů se mezi režimy liší. V krabím režimu se vypočítá ihned ze žádaného natočení od nadřazeného PC uvnitř UART Callbacku. Je možné to takto udělat, neboť se všechna kola natáčí o stejný úhel a stejným směrem. V režimu CIRCULAR se jako žádaná hodnota odesílá natočení vypočítané v rámci konkrétního kroku cyklu. Zahrnuje tedy i rampové hodnoty. Stručně řečeno, odešle se taková hodnota, která je v konkrétním kroku odesílaná na aktuátory. Opět je nutné přičíst 512 a rozložit integer na vyšší a nižší byte.

4.4.5 Ostatní byty a kontrolní součet

Zbylé byty obsahují informace jako napětí akumulátorů, teploty motorů a další. Vedoucí práce určil, že se bude místo těchto informací odesílat 0, což je logické, jelikož z informací, které jednotka dostává, není možné hodnoty vyčíst.

Byte 86 až 105 jsou v nové i předchozí řídící jednotce rovny 0. Následují informace o stavu chybového registru, rozebrané v kapitole 4.4.1.

Posledním bytem zprávy je kontrolní součet. Ten je realizován *for* cyklem, který po délce zprávy (upravené podle počtu chyb) sečte decimální hodnotu všech bytů (kromě bytu 0). *While* smyčka od sumy odečítá hodnotu 256, dokud suma není menší nebo rovna 256. Poté se zkontroluje, jestli se nerovná 255, aby došlo ke kolizi se začátkem další zprávy a kontrolní součet se vloží do posledního bytu odpovědi. Odpověď se přes UART a následně RS232 odešle do nadřazeného PC, který ji zpracuje.

4.5 CAN komunikace

Jednotka se všemi akčními členy (s výjimkou sloupu) komunikuje přes CAN sběrnici. Tato kapitola se zaměřuje primárně na strukturu a sestavení zpráv odesílaných po CANu a na zpracování odpovědí od akčních členů.

4.5.1 Sestavení zpráv

Rychlostní zpráva

Nejprve se musí přepočítat otáčky za minutu a upravit do příslušného tvaru dle návodu poskytnutého vedoucím práce. Vypočítané otáčky, ať už se jedná o hodnotu z rampy nebo ne, se vynásobí číslem $2^{31}/30000$. Tato hodnota je uložená v 32bitovém integeru, který se následně bitovým posunutím rozloží na 4 8bitové integery. První byte je původní hodnota posunutá o 24 bitů doprava, druhý byte o 16 bitů, třetí o 8bitů a poslední byte je pouze prvních 8 bitů původní proměnné.

Poté se sestavuje rychlostní zpráva, ta má jako všechny ostatní zprávy odesílané po CAN busu 8 bytů. Žádaná rychlost se zapisuje až na poslední 4 byty, ty se do zprávy ukládají způsobem *Little Endian*. Pokud bychom tedy chtěli odeslat rychlost 1000 *otáček/min,* vypadal by přepočet dle rovnice (4.2). Na servomotory 2, 4, 6 se odesílá záporná rychlost dle rovnice (4.3).

$$rychlostní\ zpráva = n_{kola} \cdot \frac{2^{31}}{30000} \tag{4.2}$$

$$rychlostní\ zpráva = -\mathrm{n}_{kola} \cdot \frac{2^{31}}{30000} \tag{4.3}$$

Zpráva bude vypadat následovně: výsledná hodnota je dle (4.2) *85 899 346*. Tomu odpovídá hexadecimální číslo: *0x051EB852*.

Rychlostní zpráva má tedy tvar dle Tabulka 5, přičemž první 4 bity jsou pro rychlostní zprávu vždy stejné.

Tabulka 5: Struktura rychlostní zprávy

Byte	Hodnota
Byte 0	0x22
Byte 1	0x8E
Byte 2	0x00
Byte 3	0x02
Byte 4	0x52
Byte 5	0xB8
Byte 6	0x1E
Byte 7	0x05

V kruhovém režimu je stěžejní správně rozřadit vypočítané rychlosti a natočení na jednotlivé pohony. V provozu se mění, která kola jsou vnitřní a která vnější, ta mají zároveň i jiné otáčky.

Pro úhel natočení 0 se musí všechna kola otáčet stejnou rychlostí. Do zprávy se propisují pouze otáčky vypočítané z dopředné rychlosti ((4.2) nebo (4.3)), a to buďto ty vypočítané v rampě (pokud se rampa využila) anebo ze žádané rychlosti od PC.

Když úhel natočení není nulový, vnější a vnitřní kola mají jiné otáčky. Pro kruhový režim se se tedy zformulují následující zprávy:

- Vnitřní s kladnými otáčkami
- Vnitřní se zápornými otáčkami
- Vnější s kladnými otáčkami
- Vnější se zápornými otáčkami
- Prostřední vnitřní s kladnými otáčkami
- Prostřední vnitřní se zápornými otáčkami
- Prostřední vnější s kladnými otáčkami
- Prostřední vnější se zápornými otáčkami

Zpráva pro aktuátory

V krabím režimu je sestavení této zprávy snadné, jelikož se všechna kola natáčí stejným směrem. Pro kruhový režim je nutné vzít v potaz, že nápravy zatáčí protichůdně, aby bylo vozidlo schopno opsat trajektorii zatáčky. To je zpracováno *if* podmínkou, která sleduje znaménko aktuálního úhlu natočení (z proměnné *steering_angle_ackermann*) a podle toho se uvnitř přepočte chtěný vnitřní a vnější úhel natočení kola na rozsah daného aktuátoru. V podmínce na kladné znaménko je zahrnuta i nula, nezáleží ale na tom, ve které je zahrnuta, protože by fungovala v obou.

Z hlediska natočení se finální hodnota (určená přímo pro rozsah aktuátoru, ať už vycházející z rampy nebo požadavku nadřazeného PC) musí také rozložit, tentokrát na dva byty. První (vyšší) byte se vytvoří bitovým posunutím o 8 bitů doprava. Druhý bitovým AND s hodnotou 0xff. Zpráva má strukturu dle Tabulka 6.

Tabulka 6: Struktura zprávy pro aktuátory

Byte	**Funkce**
Byte 0	Konstantní hodnota 0x01
Byte 1	Konstantní hodnota 0x00
Byte 2	Žádaná poloha pravého aktuátoru nápravy (při pohledu zepředu) – vyšší byte
Byte 3	Žádaná poloha pravého aktuátoru nápravy (při pohledu zepředu) – nižší byte
Byte 4	Žádaná poloha levého aktuátoru nápravy (při pohledu zepředu) – vyšší byte
Byte 5	Žádaná poloha levého aktuátoru nápravy (při pohledu zepředu) – nižší byte
Byte 6	Konstantní hodnota 0x00
Byte 7	Konstantní hodnota 0x00

Zpráva pro manipulátor

Data potřebná k sestavení příkazu, konkrétně tedy frekvence v ose x a y, jsou vyčtena uvnitř UART Callbacku (kapitola 4.3.2).

Jelikož nebyly k manipulátoru žádné podklady, nebylo známé, jakou má tato zpráva strukturu, proto se musela kontaktovat firma VOP, která vozidlo konstruovala. Ta dodala strukturu zprávy pro manipulátor (Tabulka 7):

Tabulka 7: Struktura zprávy pro manipulátor

Byte	**Funkce**
Byte 0	Konstantní hodnota 0x01
Byte 1	Spuštění otáčení osy y
Byte 2	Směr otáčení osy y
Byte 3	Perioda (rychlost otáčení) osy y
Byte 4	Spuštění otáčení osy x
Byte 5	Směr otáčení osy x
Byte 6	Perioda (rychlost otáčení) osy x
Byte 7	Konstantní hodnota 0x00

Byte 1 a byte 4 určuje, jestli se manipulátor v dané ose bude pohybovat (1) nebo nikoli (0). Propisují se *if* podmínkou, která hlídá, jestli se frekvence v jednotlivých osách nerovná nula. Pokud se nule rovná, tak se do zprávy do daného bytu propíše 0.

Byty 2 a 5 nabývají hodnoty 0 (kladný směr otáčení) nebo 1 (záporný směr otáčení). Jejich propsání do zprávy je opět ošetřeno podmínkou, která hlídá znaménko u frekvence.

Rychlost má rozsah 2 (nejvyšší rychlost) až 200 (nejnižší).

Dále bylo nutné vyřešit přepočet mezi frekvencí ze zprávy od PC na periodu (rychlost otáčení) manipulátoru. Předchozí jednotka odesílala stejnou hodnoty pro dané intervaly, viz Tabulka 8.

Tabulka 8: Frekvence-perioda manipulátoru u předchozí jednotky

Frekvence od PC	**Perioda (rychlost otáčení)**
10 ÷ 19	20
20 ÷ 29	10
30 ÷ 39	6
40 ÷ 49	4
50 ÷ 59	4
60 ÷ 69	2
70 ÷ 79	2
80 ÷ 89	2
90 ÷ 99	2
100	2

Přepočet frekvence na rychlost otáčení je v nové jednotce dle (4.4). Přičemž je *if* podmínkou ohlídáno znaménko frekvence. Pokud by se frekvence rovnala 100, tak se vzorec přeskočí a program pokračuje s hodnotou 2, odpovídající nejvyšší rychlosti. V (4.4) je vztah

uveden pro osu *x*, ale pro druhou osu je princip naprosto totožný. Toto řešení přináší větší rozsah rychlosti pro otáčení manipulátoru než v předešlé jednotce.

$$rychlost_x = (100 - f_x) * 2 \tag{4.4}$$

4.5.2 Odesílání zpráv

V souladu s předchozí řídící jednotkou se jako první odešlou zprávy na aktuátory a manipulátor, a to třikrát po sobě. Ve scriptu je toto vytvořeno *while* smyčkou s čítačem, který se při opuštění smyčky vynuluje.

Samotné odesílání je realizováno dvěma funkcemi, konkrétně:

- *HAL_FDCAN_AddMessageToTxBuffer*
- *HAL_FDCAN_EnableTxBufferRequest*

K odesílání i přijímaní CAN zpráv jsou tedy využity Tx a Rx buffery, nikoli FIFO.

Program využívá „*Polling*“ metodu, to znamená, že za pomocí *while* smyčky procesor čeká, než se zpráva úspěšně odešle před odesláním další. Smyčka je následována 5 ms dlouhou prodlevou. Z testů totiž vyšlo najevo, že bez této pauzy se zprávy neodesílaly správně a některé z nich byly vynechány vlivem zahlcení linky.

Zprávy na jednotlivé servopohony se v daném kroku cyklu odesílají pokaždé pouze jednou a mají následující posloupnost:

- SERVO 1–6 – rychlostní zpráva (pro kruhový režim viz Tabulka 9)
- SERVO 1-6 – vyžádání rychlosti
- SERVO 1–6 - vyžádání pozice
- SERVO 1-6 - vyžádání chyb.

Tabulka 9: Možné varianty rychlostních zpráv (kruhový režim)

Úhel zatáčení	Servopohon (při pohledu zepředu)	Rychlostní zpráva
Kladný	1 (pravý přední)	Vnější s kladnými otáčkami
	2 (levý přední)	Vnitřní se zápornými otáčkami
	3 (pravý prostřední)	Prostřední vnější s kladnými otáčkami
	4 (levý prostřední)	Prostřední vnitřní se zápornými otáčkami
	5 (pravý zadní)	Vnější s kladnými otáčkami
	6 (levý zadní)	Vnitřní se zápornými otáčkami
Záporný	1 (pravý přední)	Vnitřní s kladnými otáčkami
	2 (levý přední)	Vnější se zápornými otáčkami
	3 (pravý prostřední)	Prostřední vnitřní s kladnými otáčkami
	4 (levý prostřední)	Prostřední vnější se zápornými otáčkami
	5 (pravý zadní)	Vnitřní s kladnými otáčkami
	6 (levý zadní)	Vnější se zápornými otáčkami
Nulový	1,3,5 (pravá strana)	Záporné otáčky dle (4.3)
	2,4,6 (levá strana)	Kladné otáčky dle (4.2)

4.5.3 Odpovědi od servopohonů

Při odeslání jakékoliv zprávy po CAN sběrnici (s výjimkou manipulátoru) daný uzel okamžitě odesílá odpověď. Přijímaní těchto odpovědí je ve scriptu vyřešeno přerušením na přijetí nové zprávy do *RxBufferu* a následným Callbackem.

K samotnému vyzvednutí zprávy z bufferu je využívána funkce: *HAL_FDCAN_GetRxMessage*, jejíž argumenty jsou:

- O který FDCAN se jedná (na desce jsou dva).
- O který *RxBuffer* se jedná (každý má přiřazený svůj filtr odpovídající identifikátoru daného uzlu).
- *RxHeader* zprávy.
- Proměnná, do které se mají data zapsat.

Z kapitoly 4.5.2 vyplývá, že na servopohony se odesílají 4 druhy zpráv. První z nich zadává rychlost otáčení. Na tuto zprávu se ihned od motorů odešle odpověď, ale dále se nezpracovává.

Druhou je požadavek na aktuální rychlost. Odpověď má strukturu dle Tabulka 10 a zpracovává se následovně: nejprve je nutné složit 4 byty rychlosti do jednoho. O to se v programu stará bitové posunutí v kombinaci s bitovým OR, přičemž poslední bit rychlosti se posouvá na doprava o 24 bitů na první pozici, druhý za něj a tak dále. Tento postup vychází z endianity zprávy. K získání rychlosti v *km/h* za hodinu se musí provést opačný proces než při sestavování rychlostní zprávy. Podmínka také hlídá, jaké je znaménko u rychlosti. Jelikož se na servopohony 2,4,6 posílá rychlost s opačným znaménkem, tak i odpověď má opačné znaménko. Rychlost ze serva 1 (přepočítaná zpět na *km/h*) se dále používá pro rychlostní rampu v krabím režimu (kapitola 4.6.3). K určení aktuální rychlosti pro rychlostní rampu v kruhovém režimu je využívána zpráva od střední nápravy (servopohon 3 a 4). Rychlost v *km/h* se v tomto případě dopočítává dle (2.15).

Tabulka 10: Struktura odpovědí od servopohonů

Druh odpovědi	**Byte 0**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**
Aktuální rychlost	0x43	0x1C	0x00	0x02	Aktuální rychlost (little endian)			
Aktuální poloha	0x43	0x1E	0x00	0x02	Aktuální poloha (little endian)			
Chybový registr	0x43	0x02	0x00	0x02	Aktuální stav chybového registru (little endian)			

Třetí je požadavek na aktuální pozici kola (úhel pootočení, nikoli natáčení způsobené aktuátorem). Zpracování je obdobné. Stejným způsobem jako u rychlosti program složí 4 byty do jednoho a hodnota se dále zpracovává do odpovědi pro nadřazené PC, rozebrané v kapitole 4.4.

Program druh odpovědi rozlišuje pomocí funkce *switch,* aplikované na druhý byte v odpovědi. Byte 0x1C značí aktuální rychlost, 0x1E značí pozici a 0x02 je zpráva nesoucí informace o chybovém registru.

4.5.4 Odpovědi od aktuátorů

Aktuátory po přijetí zprávy okamžitě odesílají odpověď, která má formát dle Tabulka 11.

Tabulka 11: Strukutra odpovědi od aktuátorů

Byte	**Funkce**
Byte 0	Konstantní hodnota 0x01
Byte 1	Konstantní hodnota 0x00
Byte 2	Skutečná poloha pravého aktuátoru nápravy (při pohledu zepředu) – vyšší byte
Byte 3	Skutečná poloha pravého aktuátoru nápravy (při pohledu zepředu) – nižší byte
Byte 4	Skutečná poloha levého aktuátoru nápravy (při pohledu zepředu) – vyšší byte
Byte 5	Skutečná poloha levého aktuátoru nápravy (při pohledu zepředu) – nižší byte
Byte 6	Skutečný proud pravého aktuátoru nápravy (při pohledu zepředu)
Byte 7	Skutečný proud levého aktuátoru nápravy (při pohledu zepředu)

Složením daných bytů (2 a 3 pro pravé kolo, 4 a 5 pro pravé kolo, při pohledu zepředu) v odpovědi se získá aktuální vysunutí daného aktuátoru. Pro další využití v UART odpovědi pro nadřazený počítač se musí přepočítat zpět na rozsah ±512 dle (4.5) či (4.6) převzatých z návodu pro komunikaci s aktuátory.

$$Aktuátor_{odpověď} = \frac{512 + 512.\,(\,Aktuátor_{aktuální} - \,Aktuátor_{střední\ pozice})}{(Aktuátor_{horní\ pozice} - Aktuátor_{spodní\ pozice});} \tag{4.5}$$

Pro aktuální hodnotu pod středem rozsahu:

$$Aktuátor_{odpověď} = \frac{512 - 512.\,(Aktuátor_{střední\ pozice} - Aktuátor_{aktuální})}{(Aktuátor_{střední\ pozice} - Aktuátor_{spodní\ pozice});} \tag{4.6}$$

4.5.5 Vyčtení chyb

U servopohonů je čtvrtým a posledním požadavkem chybový stav. Dle souboru dodaného vedoucím práce mohou na pohonech nastat následující chybové stavy, o kterých mohou ve své odpovědi informovat (Tabulka 12).

Tabulka 12: Možné chyby na servopohonech

Chyba	Bit v odpovědi o aktuálním stavu chybového registru
Přepětí DC Link	3
Podpětí DC Link	4
Chyba proudové sondy	8
Vysoká teplota motoru	9
Vysoká teplota chladiče	11
Chyba zpětné vazby	12
Chyba sledování pozice	15
Chyba trajektorie	16
Chyba komunikace	17
Spuštění externího uzamčení	20
Saturace měniče – zkrat	21

Celkový počet možných chyb je 11, ale v odpovědi pro nadřazený PC je na specifikaci těchto chyb vyčleněn pouze jeden byte. Bylo tedy nutné zvolit pouze 8 chybových stavů a ty dále zakódovat do jednotlivých bitů v odpovědi. Tři vynechané nejsou pro chod tak stěžejní jako ty ostatní. V Tabulka 13 lze vidět chybový stav, jeho polohu v odpovědi od pohonu a pozici v bytu odpovědi pro PC.

Tabulka 13: Pozice jednotlivých chyb v daném bytu odpovědi

Chyba	**Bit v odpovědi nadřazenému PC (byte 108 až 113 dle servopohonu)**
Přepětí DC Link	0
Podpětí DC Link	1
Chyba proudové sondy	2
Vysoká teplota motoru	3
Vysoká teplota chladiče	4
Chyba zpětné vazby	5
Chyba komunikace	6
Saturace měniče – zkrat	7

Propsaní chyb je v programu ošetřeno bitovou maskou. Motory dodají zprávu o 4 bytech, celkem tedy 32 bitech. Tyto 4 byty složí do jednoho integeru (stejně jako u odpovědi rychlosti), který má odpovídající hodnotu v desítkové soustavě. Tento integer se poté dá do bitového AND s jednotlivými maskami a tím se určuje, ke kterému erroru došlo.

Například při chybě komunikace přijde zpráva: *0x43 0x02 0x00 0x02 0x00 0x00 0x02 0x00*. Při složení posledních čtyř bitů dostaneme decimální číslo 131072. Maska má stejnou hodnotu, takže při bitovém AND se do dané proměnné propíše právě číslo 131072. Podmínka poté zkontroluje, jestli k tomuto došlo a propíše chybu do odpovědi nadřazenému PC.

Aktuátory samy neodesílají žádnou zprávu vypovídající o stavu jejich chybového registru, proto jsou ve scriptu chyby aktuátorů omezeny pouze na ztrátu komunikace. Ta je implementována jednoduchým timerem, který se spustí ve chvíli, kdy program odešle zprávu na aktuátory a vynuluje se ve chvíli, kdy dorazí odpověď. Pokud nedorazí v daném čase, tak se na příslušný byte v odpovědi pro PC, specifikující chybu na aktuátoru, objeví 1. Tento čas je v programu jako proměnná a dá se změnit dle potřeby.

4.6 Režim KRAB

Po ukončení UART Callbacku se program přesune do jednoho ze dvou řídících režimů. Prvním z nich je režim KRAB. Toto řízení spočívá v tom, že se všechna kola natáčí stejným směrem pod stejným úhlem se stejnou dopřednou rychlostí. TAROS tedy nezatáčí (neopisuje žádný rádius), ale jezdí úhlopříčně (Obrázek 2) nebo rovně. Tento se režim se v reálném provozu moc často nepoužívá.

4.6.1 Určení potřebné rychlosti a natočení (KRAB)

Potřebné vstupy program získává z UART Callbacku, popsaného v kapitole 4.3.2 . Co se týče rychlosti, ta se skládá dohromady ze dvou bytů. K tomu je využito bitové posunutí vyššího bytu o 8 pozic. Posunutý byte je dále položen do bitového OR s nižším bytem. Od vzniklé hodnoty je nutné odečíst 2000 a podělit ji 100, protože před odesláním z PC se žádaná hodnota vynásobí 100 a přičte se k ní 2000. Například pro 15 *km/h* je odeslána hodnota 3500, rozložená do dvou bytů. Z dopředné rychlosti v *km/h* se dopočítává úhlová rychlost kola (4.7) a jeho otáčky za minutu (4.8) .

$$\omega_{kola} = \frac{v_{dopředná}}{3{,}6 \,.\, R_{kola}} \tag{4.7}$$

$$n_{kola} = 10\frac{\omega.60}{2.\pi} \tag{4.8}$$

Z analýzy předešlé jednotky bylo zjištěno, že na kola je nutné odeslat desetkrát větší otáčky za minutu, proto jsou ve vztahu (4.8) otáčky vynásobeny 10, aby se žádaných otáček dosáhlo.

Obdobným způsobem se získává úhel natočení, který se také musí složit z nižšího a vyššího bytu. Zde je ale hodnota navýšena o 512, takže při žádaném úhlu natočení 100 přijde ve zprávě hodnota 612 a celkový rozsah je 0 až 1024, namísto původních ± 512. Dále je nutné změnit znaménko pro žádaný úhel natočení u aktuátorů 2, 4 a 6 (celá pravá strana vozidla při pohledu zepředu). Jelikož jsou akční členy na vozidle zrcadleny symetricky. Vždy tedy musí být jedna strana v horní polovině rozsahu, zatímco druhá v dolní, aby bylo dosaženo natočení stejným směrem. To je v programu vyřešeno tak, že u aktuátorů 2,4,6 se vezme hodnota uložená v proměnné *steering_angle_ackermann* (rozsah ± 512) se záporným znaménkem a přičte se k ní 512. Kupříkladu pro úhel natočení -400 se u aktuátorů 1,3,5 dále počítá s hodnotou 112 a u 2,4,6 s hodnotou 912.

Dalším krokem je přepočítání žádaného úhlu v rozsahu 0 až 1024 [1] na rozsah všech lineárních aktuátorů dle (4.9) nebo (4.10), protože ten je u každého aktuátoru jiný. Například aktuátor 1 (pravý přední, při pohledu zepředu) má rozsah 370 až 640 se středem 500. Hodnoty se přepočítávají přímou úměrou podle toho, jestli je žádané natočení pod středem (512) nebo nad ním, to je v programu ošetřeno *if/else* podmínkou.

$$Aktuátor_{výstupní} = \frac{Aktuátor_{žádaná}.\left(Aktuátor_{střední\ pozice} - Aktuátor_{spodní\ pozice}\right)}{512 + Aktuuátor_{spodní\ pozice}} \tag{4.9}$$

[1] S tímto rozsahem se počítá, jelikož přepočty (4.9) a (4.10) vychází z návodu pro komunikaci CAN na vozidle, poskytnutého vedoucím práce, rovnice na přepočet je převzata právě z tohoto návodu.

Pro natočení pod střední hodnotou aktuátoru:

$$Aktuátor_{výstupní} = \frac{(Aktuátor_{žádaná} - 512).(Aktuátor_{horní\ pozice} - Aktuátor_{střední\ pozice})}{512 + Aktuátor_{střední\ pozice}} \quad (4.10)$$

Tyto přepočty nezahrnují nulové natočení, proto je v programu ošetřeno zvlášť *if* podmínkou. Pokud je podmínka splněna, tak se nic nepočítá a program pokračuje se střední hodnotou rozsahu všech lineárních pohonů.

4.6.2 Rampa natočení (KRAB)

Jelikož natáčení kol probíhá poměrně rychle a při náhlém natočení větším než polovina rozsahu aktuátoru, vytváří rázy, které snižují životnost pohonů. Bylo nutné zajistit pozvolnější natáčení, ke kterému slouží rampa natočení. Zároveň byla tato funkcionalita implementována i v předchozí řídící jednotce, takže pro zachování co největší zpětné kompatibility je rampa nutností.

Koncept rampy v nové řídící jednotce je pojat následovně: vždy na začátku cyklu se vypočítá rozdíl aktuálního natočení (z pozice aktuátoru 1) s požadovaným úhlem natočení. Pokud je výsledek větší než maximální tolerance, k aktuálnímu natočení se přičte nebo se od něj odečte krok (záleží na směru zatáčení). Tato rampová hodnota se následně přepočítá dle (4.9) nebo (4.10) a odešle se na aktuátory. Porovnávané hodnoty jsou v původním rozsahu ±512. Dvě *if* podmínky rozlišují, jestli se zatáčí do kladné nebo záporné hodnoty.

Porovnávání s maximální tolerancí zaručuje, že natočení nepřekmitne ani nezačne oscilovat, protože pokud je rozdíl porovnávaných hodnot menší než tolerance, tak se na pohony odešle rovnou žádaná hodnota od nadřazeného PC. Zároveň je tato tolerance dostatečně vetší než maximální krok. Velikost kroku i tolerance byla zjištěna experimentálně. Optimálních výsledků bylo dosaženo při maximálním kroku 220 a toleranci 250. V rampě je také využíván flag, který určuje, jestli se má odeslat hodnota rampy nebo finální hodnota od PC. Pokaždé když je využita rampa se flag přepne do 1 a na konci současného kroku cyklu se vždy vynuluje, aby se program nezasekl v rampě.

4.6.3 Rychlostní rampa (KRAB)

Důvody k zakomponování této funkce jsou ještě podstatnější než u rampy natočení. Zatímco bez ní by vozidlo při nižších rychlostech (přibližně do 5 km/h) a pomalém zatáčení bylo schopno fungovat, tak bez rychlostní rampy by byl provoz téměř nemožný. Při testu funkcionality nového řídícího programu bylo totiž zjištěno, že servopohony nejsou schopny většího skokového navýšení rychlosti než 2 km/h a už při tomto skoku bylo jasné, že se jím snižuje jejich životnost. Ideální je tedy skokové zvýšení rychlosti maximálně o 1 *km/h*. Rampa je řešena několika podmínkami rozlišujícími stavy, které mohou nastat:

- žádaná rychlost kladná, aktuální rychlost kladná
- žádaná rychlost záporná, aktuální rychlost kladná
- žádaná rychlost nulová, aktuální rychlost kladná
- žádaná rychlost kladná, aktuální rychlost záporná
- žádaná rychlost záporná, aktuální rychlost záporná
- žádaná rychlost nulová, aktuální rychlost záporná

Princip rychlostní rampy je stejný jako v případě rampy natočení, porovnává se aktuální rychlost se žádanou. Pokud je rozdíl větší než tolerance a zároveň má správné znaménko, spustí se rampa. Tato dvě kritéria jsou provázána logickým AND a druhé z nich určuje, jestli se zpomaluje nebo zrychluje. Například při požadavku na zrychlení bude rozdíl žádané rychlosti a aktuální rychlosti kladný, při zpomalování záporný, záleží však i na znaménku rychlosti (jízda dopředu nebo couvání).

V rampě se k aktuální rychlosti přičte nebo se od ní odečte maximální krok a ihned se z této hodnoty vypočítá úhlová rychlost (4.7) a otáčky za minutu (4.8).

I u této rampy se využívá samostatný flag, který funguje zcela stejně jako v kapitole 4.6.2.

U obou ramp v nadzvednutém stavu, nezatíženém vlastní vahou, byl pohyb mírně trhavý. To je způsobeno skokovým inkrementování nebo dekrementováním dané veličiny. Avšak při zatížení pohonů vlastní vahou již trhání zjevné nebylo. Hodnoty maximálního skoku a tolerance byly opět zjištěny experimentálně a nejlepších výsledků bylo dosaženo při maximálním skoku 0,7 km/h a maximální toleranci 1 km/h. Při nižším skoku byla akcelerování i zpomalování plynulejší, ale příliš pomalé, což mělo obzvlášť při zpomalování značný vliv na ovladatelnost vozidla.

4.7 Režim CIRCULAR

Druhý z implementovaných režimů se nazývá CIRCULAR (kruhový režim). Principem řízení je natáčení kol na přední a zadní nápravě, ta se natáčí protichůdně, zatímco kola na prostřední nápravě se nijak nenatáčí a celou dobu zůstávají ve střední poloze. Pro reálný provoz TAROSe je tento režim používanější než KRAB.

4.7.1 Získání úhlu β

Tento řídící režim využívá kinematiku Ackermannova podvozku (kapitola 2.2). Pro její určení je nutné znát požadovaný úhel natočení aproximovaného prostředního kola. Tento úhel se nejprve vyčte v UART Callbacku (kapitola 4.3.2) a lineární závislostí se přepočte (4.11) z žádaného úhlu natočení v rozsahu (±512).

$$\beta = \frac{\text{úhel natočení} \,.\, 13{,}55}{512} \tag{4.11}$$

Maximální úhel aproximovaného kola je 13,55º. To lze určit ze vztahu (2.5) za předpokladu, že maximální natočení vnitřního kola je 17º. Číslo 512 vystupující v (4.11) je rozsah natočení kol od nuly do maxima pro jeden směr ze zprávy od nadřazeného PC. Pro výpočet kinematiky je využíván vždy kladný úhel, což je při přepočtu zabezpečeno *if* podmínkou, která změní znaménko *úhlu natočení* tak, aby byl vždy kladný.

4.7.2 Rampa natočení (CIRCULAR)

Jelikož se vnitřní a vnější kola nenatáčí o stejný úhel jako u předchozího typu řízení, bylo nutné postupně zvětšovat samotný úhel β (úhel natočení aproximovaného prostředního kola), aby natáčení kol bylo rovnoměrné. Před tímto řešením byla otestována varianta, kdy se rampa realizovala pro každý aktuátor zvlášť. Toto řešení nebylo vhodné, jelikož se kola nenatáčela rovnoměrně, protože musely některé aktuátory urazit menší dráhu.

Rampa je opět realizována porovnáváním současné a chtěné hodnoty a následným zvýšením současné hodnoty o daný krok.

K zjištění současného úhlu bylo využito vzorce (2.4) pro výpočet natočení vnitřního kola a z něj vyjádřený úhel β.

$$\beta = \tan^{-1}\left(\frac{2L \,.\, \tan \beta_{vnitřní}}{T \,.\, \tan \beta_{vnitřní} + 2L}\right) \tag{4.12}$$

Vzorec (4.12) platí pouze pro natočení vnitřního kola, ale to se na základě směru zatáčení mění. Kladný směr zatáčení je doprava (při pohledu na vozidlo zepředu), tím pádem jsou při tomto směru vnitřní kola připevněna k aktuátorům 2, 4 a 6 (pravá strana vozidla při pohledu zepředu). Při záporném zatáčení je to naopak. Problém by se teoreticky dal vyřešit odvozením vzorce pro výpočet úhlu β z vnějšího kola. Programová implementace tohoto řešení se ale ukázala jako problémová. Script nepřecházel mezi těmito vzorci korektně, a tak rampa fungovala pouze pro jeden směr zatáčení. Pro druhý směr se ostatní kola nenatočila dostatečně. Bylo zřejmé, že se hodnoty musí vyčítat ze dvou kol, ideálně na stejné nápravě, u nichž se bude střídat, které je v daném okamžiku vnitřní. Zde přišel další problém, tedy jak zajistit plynulý přechod mezi tím, ze kterého kola se bude hodnota odečítat. Z analýzy předchozí řídící jednotky vozidla vyplynulo, že vnitřní kolo bude vždy v záporném rozsahu daného aktuátoru, respektive bude aktuátor vždy v dolní části rozsahu, pod svým středem. Proto se hodnota vyčítá jen za této podmínky. To zaručilo plynulý přechod a snadné rozpoznání vnitřního kola, jelikož obě kola

v tomto řídícím režimu pod středem rozsahu svého aktuátoru jednoduše být nemohou. Zároveň pokud se úhel počítá z aktuátoru 1, tak je záporný, protože aktuátor 1 je vnitřním pouze při záporném směru zatáčení. Pokud se odečítá z aktuátoru 2, tak je kladný.

Žádaná hodnota, se kterou se vypočítaná hodnota porovnává, je v tomto případě chtěné natočení od nadřazeného PC, přepočítané na úhel ve stupních, ale v plném rozsahu (-512 až 512), respektive přepočítané na rozsah (±13,55°) dle (4.11).

Maximální tolerance i krok byl určen experimentálně. Tolerance je 6° a krok je 5.5°.

Z rampové hodnoty se dále dopočítají oba potřebné úhly natočení pro vnitřní a vnější kola ((2.4), (2.5)). Pokud je tato hodnota 0, tak se vztahy neuplatňují a rovnou je vnitřní a vnější úhel natočení 0.

4.7.3 Rychlostí rampa (CIRCULAR)

Ze stejných důvodů, jako v kapitole 4.6.3 musela být i v tomto režimu využita rychlostní rampa. Princip je zde totožný, porovnává se aktuální hodnota se žádanou a pokud je jejich rozdíl větší než tolerance, tak se k aktuální hodnotě přičte krok a odešle se na motory. V případě, že je rozdíl žádané a aktuální hodnoty menší než tolerance, odešle se přímo žádaná hodnota. Vypnutí a zapnutí rampy je opět ošetřeno flagem. Tolerance, krok zrychlení a krok zpomalení jsou zde zvoleny stejně jako u krabího režimu (kapitola 4.6.3).

Následuje podmínka, která podle flagu rampy rozliší, jestli do dalších výpočtů dle vztahů (2.6), (2.7), (2.13), (2.14) vstupuje rampová hodnota nebo nikoli. Ovšem pro nulový úhel natočení vztahy v kapitole 2.2.1 z matematických důvodů nefungují, proto se souběžně z rampové rychlosti dopočítávají otáčky za minutu, stejně jako u režimu krab (rovnice (4.8)), jelikož při nulovém natočení se všechna kola otáčí stejnou rychlostí.

4.8 Bezpečnostní prvky

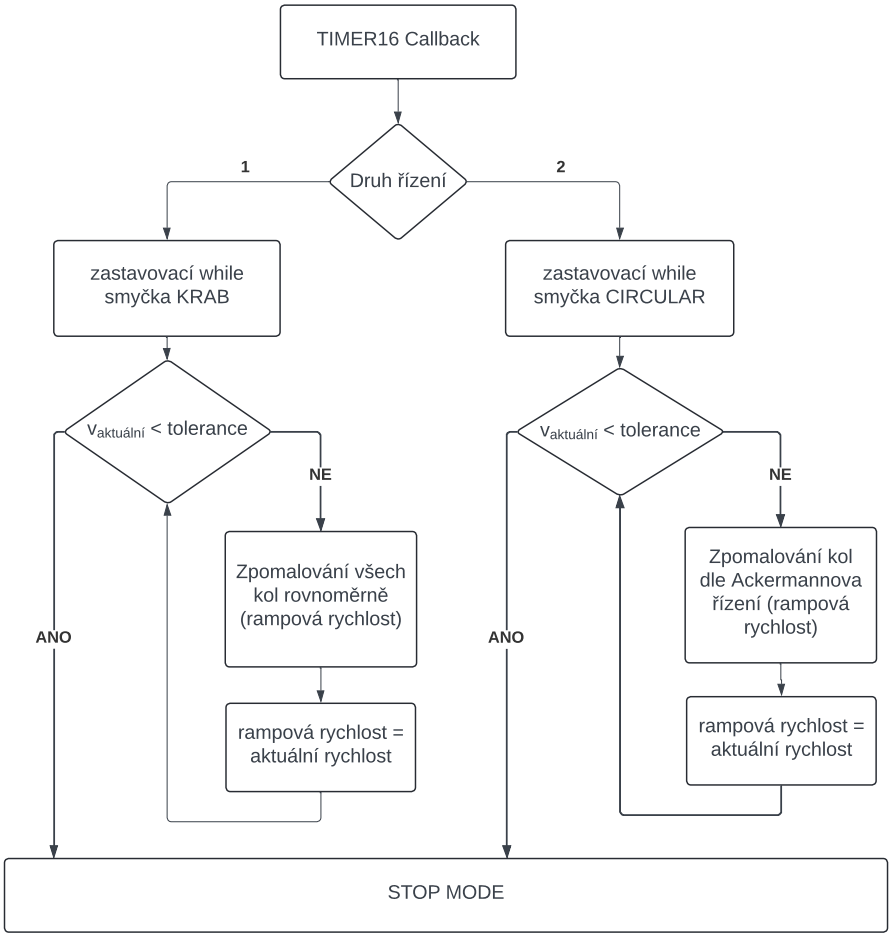
Jelikož je TAROS autonomní vozidlo, musí být kladen důraz na bezpečnost, a to jak při autonomním provozu, tak i během ovládání vozidla na dálku. Při závadě by mohlo dojít nejen ke škodám na majetku, včetně TAROSe samotného, který je velice drahým prototypem, ale i k újmě na zdraví.

4.8.1 Nouzové zastavení

Z výše zmíněných důvodu je v řídící jednotce dle zadání práce implementováno nouzové zastavení při ztrátě komunikace s nadřazeným počítačem, nebo dálkovým ovládáním při neautonomním provozu.

V programu se realizuje využitím timeru, který se spustí na začátku cyklu řídícího režimu a vynuluje se vždy uvnitř UART Callbacku, tedy když jednotka přijme novou zprávu od nadřazeného PC. Hranice přetečení timeru je ve scriptu uložena jako proměnná, takže je možné ji libovolně nastavit.

Při přetečení timeru se spustí Callback (schematicky znázorněn na Obrázek 15), uvnitř kterého se vozidlo zastaví. Realizace zastavení se dělí podle režimu řízení, ze kterého se do callbacku přešlo. V obou případech je ale princip stejný, opět se principiálně jedná o rychlostní rampu (kapitola 4.6.3), avšak v tomto případě, pokud není skutečná hodnota dostatečně blízko nulové rychlosti, se script uzavře do *while* smyčky, která se neukončí, dokud není tolerance splněna.



Obrázek 15: Schéma nouzového zastavení

Uvnitř této funkce se tedy aktuální hodnota snižuje o maximální krok. Aktuální hodnota pro tuto rampu je při vstupu do Callbacku vyčtená přímo z motorů. Pro další kroky už tomu tak

není. Na konci kroku cyklu se za aktuální přepíše hodnota, jež byla v daném kroku odeslána na pohony. Chybí zde tedy skutečná zpětná vazba.

Zastavení je takto provedeno z důvodu priority přerušení. Servomotory neustále odesílají odpovědi se skutečnou rychlostí. Interrupt, který by je zpracoval se však nespustí, protože nemůže narušit probíhající Callback.

Pokud se zastavuje ze středového režimu, tak se rychlosti přepočítávají a odesílají na správné pohony dle posledního úhlu natočení. Ten se nemění, jelikož vozidlo pouze zastavuje, ale kola se nevrací do své střední polohy, narozdíl od předešlé řídící jednotky. Tato změna je vylepšením, protože ve většině případů není praktické, aby se kola natáčela zpět, například, pokud by vozidlo uvízlo ve výmolu.

Na konci Callbacku se ještě jednou odešle nulová rychlost a script se přesune mimo Callback do stavu hlavní smyčky, kde se neustále odesílá nulová rychlost, takže vozidlo stojí na místě a čeká na další příkazy.

Dále je nutné zmínit, že uvnitř Callbacků nelze používat funkci *HAL_Delay* pro prodlevu mezi jednotlivými zprávami, aby nedošlo k zahlcení sběrnice. V Callbacku totiž nefunguje systémový čítač, který funkce využívá. Zpoždění je zde vyřešeno „hrubou silou“ za použití *for* cyklu, který inkrementuje proměnnou do dané hodnoty, čímž vytvoří žádanou pauzu.

Jelikož při zastavování nemá jednotka zpětnou vazbu v podobě skutečné rychlosti, bylo nutné zastavení vyzkoušet. Ukázalo se, že tento způsob provedení funguje a jelikož se ve smyčce odesílají pouze zprávy na motory, tak k zastavení dojde rychle a plynule, což bylo hlavním cílem.

Maximální krok zrychlení a tolerance byly určeny experimentálně. Konečné hodnoty jsou krok 0,7 km/h pro krabí režim a 0,25 km/h pro kruhový režim. Tolerance při zastavení z krabího režimu je 1 km/h a při zastavení z režimu CIRCULAR je 0,65 km/h. Nižší hodnoty pro kruhový režim byli způsobeny tím, že při stejných hodnotách jako u krabího režimu bylo zastavení moc prudké.

4.8.2 Ochrana proti zacyklení

V programu se nachází velké množství *while* smyček, ve kterých může uvíznout. Kupříkladu smyčka, ve které jednotka při odesílání zprávy setrvává dokud, se odeslání úspěšně nedokončí. Za normálního běhu, zde setrvá pouze několik milisekund, pokud je ale na lince nějaký problém, zasekne se. Proto je do každé *while* smyčky zařazena další podmínka provázaná s tou předchozí logickým AND. V této podmínce se hlídá hodnota počítadla, jež se uvnitř dané smyčky inkrementuje, pokud tedy přeteče, tak se program ze smyčky dostane a čítač se vynuluje. Pokud k tomuto dojde při odesílání zprávy na aktuátory, tak jednotka vyhodnotí chybu komunikace na daném aktuátoru.

5 HARDWARE

Nová řídící jednotka, stejně jako ta předešlá, je postavena na základě vývojové desky, konkrétně tedy Nucleo-H7A3ZI-Q od společnosti STMicroelectronics. Ostatní komponenty jako CAN transceivery, snižující měnič napájení nebo relé spínající sloup jsou realizovány ve formě přídavné desky (shieldu), která se nasadí na Nucleo.

5.1 Návrh přídavné desky

Návrh desky byl proveden v programu KiCAD. Deska má 3 hlavní celky.

Prvním je CAN bus. Pro správnou funkci sběrnice jsou využity transceivery, terminační odpory a filtrační kondenzátory. Obvod transceiveru má své dílčí komponenty, ale v této desce je použit již předem připravený modul s čipem TJA 1050. Piny modulu se zasunou do pinových lišt připravených v desce. Terminační rezistory mají dle normy velikost 120 Ω. Blokovací kondenzátory jsou zvoleny elektrolytické s kapacitou 10 µF. Všechny zmíněné součástky jsou na shieldu dvakrát, neboť jednotka využívá 2 CAN bus linky. Obě jsou vyvedeny do separátní svorek s výstupy CAN_L, CAN_H a GND.

Dalším celkem je napájení. Vstupem do jednotky je 12V stejnosměrná linka. Napájení Nuclea, ale neumožňuje takovéto napětí, proto je nutné ho snížit. K tomu slouží STEP DOWN měnič *TRACO TSR0.5–2450*, který má vstupní rozsah napětí 6,5 V až 36 V s konstantním výstupním napětím 5 V a výstupním proudem 0,5 A. Tento měnič je ideální volbou, jelikož Nucleo umožňuje externí napájení z 5V linky při maximálním proudu zdroje 500 mA. Další součásti jednotky jsou také napájené z 5 V. Napětí 12 V z vozidla je přivedeno na vstupní svorkovnici a filtrováno přes dvojici kondenzátorů na vstupu měniče. Jeden elektrolytický s kapacitou
22 µF a druhý keramický s kapacitou 100 nF. Na výstupu měniče je další elektrolytický kondenzátor s kapacitou 22 µF, stejný jako na vstupu. Všechny tyto kondenzátory jsou umístěny na druhé straně desky, co nejblíže u snižujícího měniče. Funkčnost je signalizována zelenou LED diodou na vstupu a výstupu.

K ovládání sloupu je na desce vytvořen samostatný obvod. Základem jsou dvě 5V relé *OMRON G5V-1* s výstupním proudem 1 A, která jsou pro účely spínání sloupu dostačující. Spínací obvod dále obsahuje blokovací diodu, lokální (decoupling) kondenzátor s kapacitou 100 nF, LED diodu se sériovým odporem k signalizaci sepnutí. Elektromagnetické relé spíná NPN bipolární tranzistor řízený GPIO pinem Nuclea, který má na výstupní napětí 3,3 V. Dále má u sebe tranzistor pull down rezistor a bázový odpor k vytvoření bázového proudu. Zvolený tranzistor má saturační proud I_{bSAT} = 5 mA při proudu kolektorem I_c = 100 mA. V tomto případě není proud kolektorem 100 mA, jelikož proud cívkou relé je I_r = 30 mA a signalizační diodou teče proud I_D = 20 mA. Maximální úbytek je na diodě U_d = 2,2 V. Proto byl sériový odpor LED diody zvolen dle (5.1).

$$R_D = \frac{U_z - U_d}{I_d} \tag{5.1}$$

Výsledný odpor nejbližší vypočítané hodnotě je 150 Ω. Dle prvního Kirchhoffova zákona (5.2) lze určit proud kolektorem:

$$I_c = I_d + I_R \tag{5.2}$$

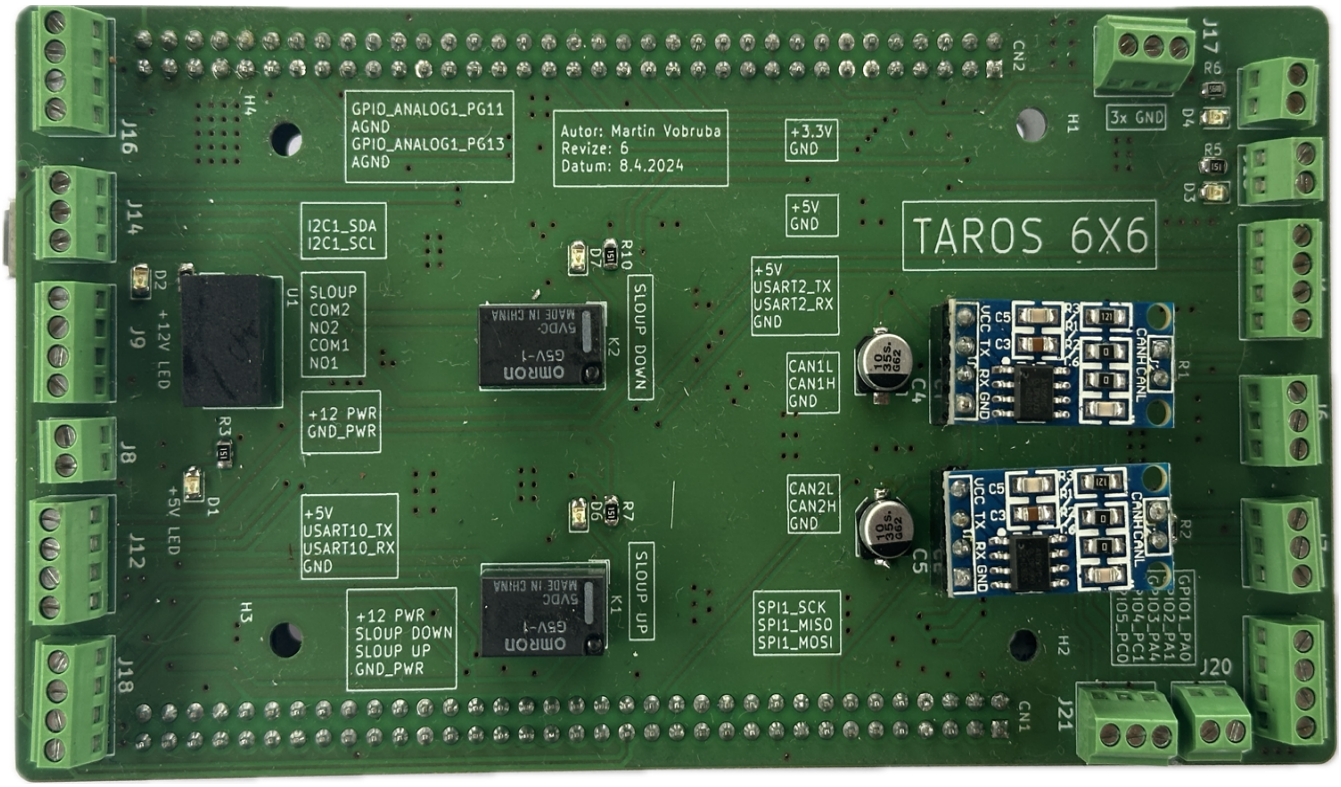
Proud kolektorem I_c = 50 mA, při této hodnotě saturačnímu proudu odpovídá I_{bSAT} = 2,5mA. Pro tuto hodnotu je dle (5.3) určen bázový rezistor.

$$R_B = \frac{U_{GPIO} - U_{BE}}{I_{bSAT}} \tag{5.3}$$

Nejbližší vyráběný rezistor je 1kΩ.
Na desce jsou dále vyvedeny do svorkovnic další piny vývojové desky:

- 5 GPIO pinů
- 2x UART (jeden používaný pro komunikaci s nadřazeným PC, druhý volný)
- I2C
- SPI
- +5 V (se signalizační LED diodou)
- +3,3 V (se signalizační LED diodou)
- 2x Analogové piny + analogová zem

Současně se nepoužívají, ale do budoucna umožní rozšíření funkcionality jednotky.
Osazenou přídavnou desku lze vidět na Obrázek 16.

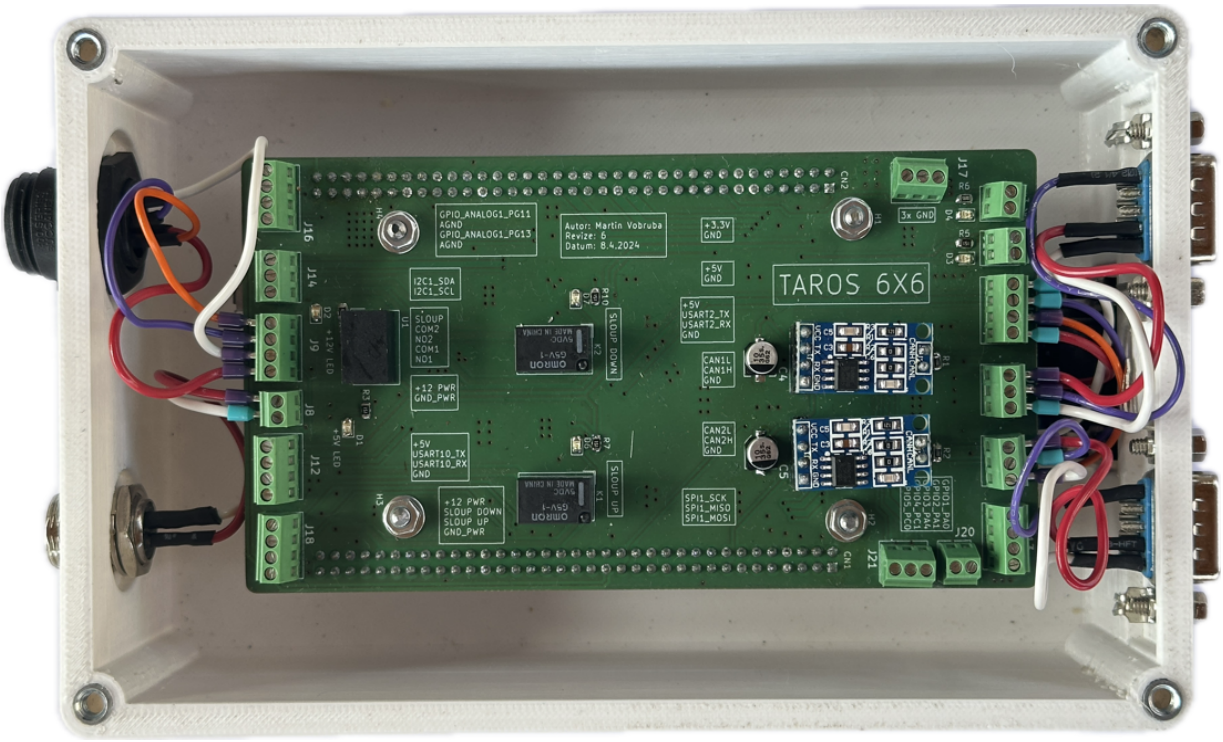


Obrázek 16: Osazená přídavná deska

5.2 Porovnání hardwaru řídících jednotek

Nová jednotka (Obrázek 17) oproti předchozí (Obrázek 9) obsahuje pouze jeden hlavní celek, skládající se z Nuclea a připojené přídavné desky, zatímco ta předešlá obsahuje kromě hlavní desky ještě tři přídavné, separátní obvody, které jsou vzájemně propojené vodiči, což značně snižuje celkovou přehlednost. Přídavná deska, zajišťující rozdělení jedné CAN sběrnice do dvou, v nové jednotce není potřeba, jelikož Nucleo obsahuje dvě sběrnice. Zároveň obvod, ovládající sloup je nyní umístěn přímo na desce s výrazně menšími elektromagnetickými relé. Nová jednotka má o dva konektory méně než ta předchozí, jelikož zastávaly funkci, která se už ve vozidle nevyužívá.

Nová krabice byla vymodelována přímo pro současný hardware, což zlepšuje kompaktnost a estetickou stránku věci. Na druhou stranu, pokud by měl být hardware rozšířen o další celek musí se vytvořit nová krabice. Další fotografie nové řídící jednotky lze vidět v příloze 3.



Obrázek 17: Nová řídící jednotka

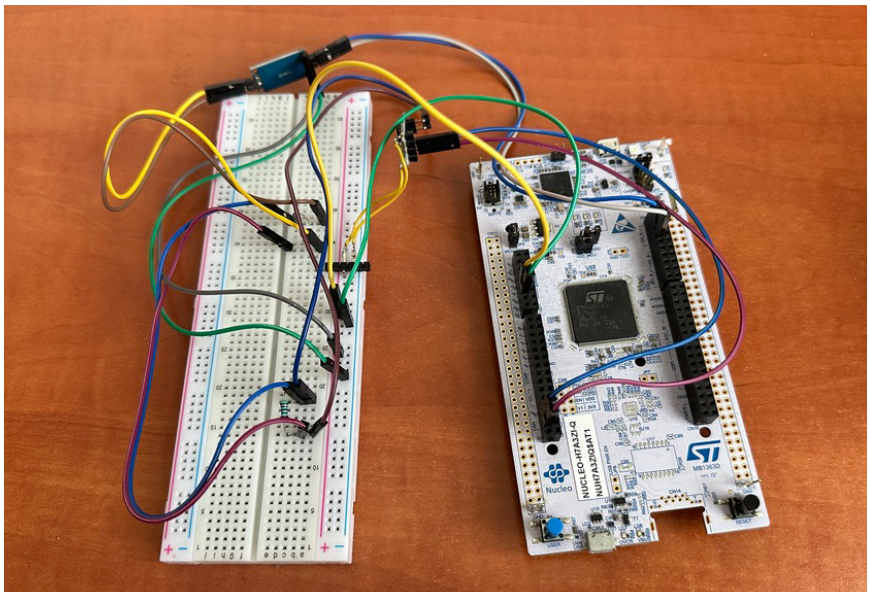
6 OTESTOVÁNÍ JEDNOTKY

Po úspěšném napsání programu, navržení a osazení přídavné desky bylo zapotřebí jednotku otestovat, konkrétně ověřit funkčnost a zpětnou kompatibilitu. Tato kapitola se zabývá plněním tohoto cíle práce a je rozdělena na testování firmwaru a hardwaru.

6.1 Testování firmwaru

Testování v tomto případě probíhalo průběžně ve dvou fázích. Nejprve se napsaný kód vyzkoušel na domácí testovací sestavě (Obrázek 18), která se skládala z Nuclea a dvou transceiverů, propojených skrze breadboard. To umožňovalo vyzkoušet, zda se zprávy správně odesílají a přijímají skrze CAN/USB převodník. Ten zároveň může zprávy odesílat a tím simulovat odpovědi od akčních členů vozidla. Nebylo ale možné napodobit přímo chování vozidla. Proto se ve druhé fázi testování program vyzkoušel přímo na vozidle v kasárnách Univerzity obrany, kde byl TAROS umístěn. Zde se testovalo se zvednutým vozidlem. Do předchozí jednotky vozidla se připojil vývod nových CAN linek a díky tomu bylo možné vždy odladit žádané chování vozidla, které ve většině případů nebylo po domácí zkoušce ideální. S vozidlem bylo nutno zacházet velice obezřetně.

Po ověření funkčnosti dílčích částí scriptu se firmware testoval jako celek v provozu na dálkové ovládání. Nejprve zvednutý, kde se odladily nově objevené nedostatky, jako například špatně nastavená rampa. Poté byl čas na reálných provoz, kdy TAROS jezdil na dálkové ovládání mimo dílnu. Hlavním kritériem bylo otestování rychlostní a natáčecí rampy a správné odvalování kol při kruhovém režimu, čímž se potvrdila korektnost výpočtů u Ackermannova řízení. Po úspěšné zkoušce byl firmware uznán jako funkční. Z důvodu časového rozpočtu (TAROS byl na začátku dubna 2024 odvezen do Jindřichova Hradce) tedy ještě nebyla navržena přídavná deska, tím pádem probíhaly testy firmwaru bez ní.



Obrázek 18: Testovací sestava

6.2 Testování hardwaru

Po návrhu a osazení přídavné desky bylo nutné vyzkoušet její funkčnost. Nejprve se multimetrem vyzkoušela přítomnost zkratu na napěťových linkách. Poté se shield zasadil do vývojové desky a připojil se na laboratorní zdroj, který dodával 12 V (stejně jako napájecí linka na vozidle). Snižující měnič funguje správně a snižuje 12 V na 5 V, které napájí Nucleo. Při tom si ze zdroje odebírá přibližně 90 mA.

Kvůli opožděnému doručení některých součástek nebylo ihned možné otestovat desku jako celek přímo s firmwarem jednotky. Z tohoto důvodu se nejprve testovalo spínaní relé ovládajících sloup a UART. K tomu sloužil testovací script, který podle UART zprávy poslané do desky spínal relé a zároveň odesílal zpět stav, ve kterém by se zrovna sloup nacházel („UP“, „DOWN“, „STILL“). Tento script fungoval bezchybně.

Další komplikace zbrzdily dodání CAN transceiverů, avšak jeden k dispozici byl, takže funkčnost na sběrnice na shieldu se dala ověřit jednoduchým testovacím programem, který odesílal zprávu. Jedna po druhé prošly obě sběrnice testem. Při dodání všech komponentů byla ověřena funkčnost navrženého hardwaru.

7 ZÁVĚR

V rešeršní části této bakalářské práce je popsáno autonomní vozidlo TAROS. Dále se rešerše zabývá kinematikou Ackermannova řízení, která je stěžejní pro jeden z režimů řízení používaných na vozidle. Poslední kapitolu rešeršní části tvoří popis sběrnice CAN, přes kterou řídící jednotka komunikuje s akčními členy.

První kapitolou praktické části se zabývá analýzou předešlé řídící jednotky. Jsou zde popsány její součásti, vstupní a výstupní konektory a signály. Rovněž přibližuje určení maximálního úhlu natočení jednotlivých kol, který nebyl známý.

Další a zároveň nejdelší kapitola se zaměřuje na tvorbu firmwaru nové jednotky. Tato část práce byla tou nejnáročnější. Jednotka umožňuje využívání dvou režimů řízení: krabí a kruhový režim, přičemž druhý zmíněný se ukázal jako výrazně komplikovanější, protože zahrnuje odesílání odlišných rychlostí a natočení na vnitřní a vnější kola. Za nejsložitější část bych označil zakomponování rychlostní rampy a rampy natočení, obzvlášť u kruhového režimu. Bylo pro mě výzvou nejen naprogramování rampy, ale hlavně odladění jednotlivých parametrů při reálném testování pro zajištění plynulého chodu vozidla.

Návrh hardwaru byl relativně přímočarý, jelikož přídavná deska obsahuje předpřipravené moduly jako CAN Transceiver modul nebo snižující měnič. Zároveň jsem si ale vyzkoušel návrh obvodu ovládajícího výsuvný sloup pomocí elektromagnetických relé, kde jsem zužitkoval znalosti nabyté v oblasti elektroniky. Mírné komplikace se vyskytly při osazování desky (například špatně zvolené pájecí plošky tranzistoru). Práce mi ale umožnila se výrazně zlepšit v pájení, ke kterému jsem se během studia nedostal.

Na závěr bylo nutné otestovat funkčnost jednotky. Testování firmwaru a hardwaru muselo bohužel z logistických důvodů (vozidlo od dubna 2024 nebylo k dispozici) probíhat odděleně. Nejprve proběhl test firmwaru. Celkově byla každá nová část kódu souběžně vyzkoušena přímo na vozidlo, kde se odladila funkčnost. V další fázi testování se vyzkoušely jízdní režimy při dálkovém ovládání se zvednutým vozidlem. Nakonec proběhl test jízdy mimo dílnu, při kterém se program ukázal jako funkční, zejména s ohledem na správné odvalování kol při využívání Ackermannova řízení. Jednotlivé části navrženého hardwaru jsem otestoval v mechatronické laboratoři, kde jsem z PC posílal řídící zprávu na UART piny desky, napájené z laboratorního zdroje. Odesílání a příjem zpráv přes UART i CAN probíhal v pořádku společně se spínáním relé.

Z hlediska budoucího vylepšení jsou na desce vyvedeny piny dalších komunikační sběrnic (I2C, další UART) k ovládání dalších prvků, o které může být nyní vozidlo rozšířeno. Ze strany firmwaru vidím jako možnost vylepšení do budoucna přesunutí některých funkčních bloků do samostatných souborů a volat je pouze jako funkce, což by program zkrátilo a zpřehlednilo.

SEZNAM ZKRATEK

ACK – Acknowledgement Field
CAN – Controller Area Network
CAN FD – Controller Area Network Flexible Data-Rate
CRC – Cyclic Redundancy Check
DLC – Data Length Code
ECU – Electronic Control Unit
EOF – End Of Frame
GND – Ground
GPIO – General-Purpose Input/Output
IFS – Inter-frame spacing
MCU – MicroController Unit
RTR – Remote Transmission Request
REC - Receive Error Counter
SOF – Start Of Frame
SRR – Substitute Remote Request
TEC – Transmission Error Counter
TTL – Transistor-Transistor-Logic
UART – Universal Asynchronous Receiver-Transmitter

SEZNAM POUŽITÝCH ZDROJŮ

[1] Ackerman Steering. In: *DataGenetics* [online]. 2016 [cit. 2024-04-27]. Dostupné z: http://datagenetics.com/blog/december12016/index.html

[2] Ackermann steering geometry. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2024-04-22]. Dostupné z: https://en.wikipedia.org/wiki/Ackermann_steering_geometry

[3] VLK, František. *Podvozky motorových vozidel*. 3. přeprac., rozš., aktualiz. vyd. Brno: Prof.Ing.František Vlk, DrSc, 2006, 464 s. : il. ISBN 80-239-6464-X.

[4] ZHANG, Zutao, Xingtian ZHANG, Hongye PAN, et al. A Novel Steering System for a Space-Saving 4WS4WD Electric Vehicle: Design, Modeling, and Road Tests. *IEEE transactions on intelligent transportation systems* [online]. PISCATAWAY: IEEE, 2017, **18**(1), 114-127 [cit. 2024-04-22]. ISSN 1524-9050. Dostupné z: doi:10.1109/TITS.2016.2561626

[5] HRBÁČEK, J, T RIPEL a J KREJSA. Ackermann mobile robot chassis with independent rear wheel drives. In: *Proceedings of 14th International Power Electronics and Motion Control Conference EPE-PEMC 2010* [online]. Ohrid: IEEE, 2010, T5-46-T5-51 [cit. 2024-04-22]. ISBN 1424478561. ISSN 1803–7232. Dostupné z: doi:10.1109/EPEPEMC.2010.5606853

[6] ČEPL, Miroslav. *Návrh a realizace modelu podvozku mobilního robotu pro framework ROS*. Brno: Vysoké učení technické v Brně. Fakulta strojního inženýrství, 2017. Bakalářská práce. VUT Brno.

[7] CAN Bus Explained - A Simple Intro [2023]. In: *CSS ELECTRONICS* [online]. 2023 [cit. 2024-04-21]. Dostupné z: https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial

[8] 101: Controller Area Network (CAN) standard. In: *NXP Community* [online]. 2021 [cit. 2024-04-21]. Dostupné z: https://community.nxp.com/t5/Blog/101-Controller-Area-Network-CAN-standard/ba-p/1217054

[9] Aplikování sběrnice CAN. In: *Vyvoj.hw.cz* [online]. 2004 [cit. 2024-04-21]. Dostupné z: https://vyvoj.hw.cz/navrh-obvodu/rozhrani/aplikovani-sbernice-can.html

[10] Controller Area Network (CAN). *EECS* [online]. 2008, **2008**(461), 2 - 4 [cit. 2024-04-21]. Dostupné z: https://www.eecs.umich.edu/courses/eecs461/doc/CAN_notes.pdf

[11] CAN Bus: What Is It and How Does It Work? In: *EMQX* [online]. 2023 [cit. 2024-04-26]. Dostupné z: https://www.emqx.com/en/blog/can-bus-how-it-works-pros-and-cons

[12] CAN - Controller Area Network. In: *CANLAB* [online]. b.r. [cit. 2024-04-27]. Dostupné z: https://www.canlab.cz/cs/can_bus

[13] Understanding Bit Stuffing In CAN Bus Systems. In: *CanBusHack* [online]. 2023 [cit. 2024-04-27]. Dostupné z: Can Errors: Basics

[14] Can Errors: Basics. In: *Influx Technology* [online]. c2022 [cit. 2024-04-27]. Dostupné z: https://www.influxbigdata.in/can-error-basics

[15] CAN protocol: Understanding the controller area network. In: *Engineers Garage* [online]. 2024 [cit. 2024-04-27]. Dostupné z:

https://www.engineersgarage.com/can-protocol-understanding-the-controller-area-network-protocol/

[16] BU, Haixiang, Aijuan LI, Xin HUANG, Wei LI a Jian WANG. Optimal Design of the Six-Wheel Steering System with Multiple Steering Modes. *Mathematical problems in engineering* [online]. New York: Hindawi, 2021, **2021**(12), 3 [cit. 2024-04-27]. ISSN 1024-123X. Dostupné z: doi:10.1155/2021/1716116

[17] BARRERA, Alejandra, ed. *Advances in Robot Navigation*. InTech, 2011. ISBN 978-953-307-346-0.

[18] A Brief Introduction to Controller Area Network. In: *Copperhill Technologies* [online]. c2024 [cit. 2024-04-27]. Dostupné z: https://copperhilltech.com/a-brief-introduction-to-controller-area-network/

[19] The CAN Bus Protocol Tutorial. In: *Kvaser* [online]. c2024 [cit. 2024-04-27]. Dostupné z: https://www.kvaser.com/can-protocol-tutorial/

SEZNAM OBRÁZKŮ

Obrázek 1: Autonomní vozidlo TAROS
Obrázek 2: Režimy řízení (nalevo KRAB napravo CIRCULAR) převzatý z [16]
Obrázek 3: Ackermannova kinematika pro dvou podvozkové vozidlo, převzato z [4]
Obrázek 4: Schéma diferenciálního podvozku převzaté z [17]
Obrázek 5: CAN bus – schéma zapojení převzaté z [18]
Obrázek 6: Arbitrační fáze, obrázek převzatý z [12]
Obrázek 7: TTL a CAN signál, obrázek převzatý z [18]
Obrázek 8: Datový rámec, obrázek převzatý z [12]
Obrázek 9:Předešlá řídící jednotka
Obrázek 10: Schéma principu jednotky
Obrázek 11: Tradiční zapojení CAN busu, obrázek převzatý z [19]
Obrázek 12: Schématické znázornění měření maximálního úhlu
Obrázek 13: Schéma funkce firmwaru
Obrázek 14: Inicializační proces
Obrázek 15: Schéma nouzového zastavení
Obrázek 16: Osazená přídavná deska
Obrázek 17: Nová řídící jednotka
Obrázek 18: Testovací sestava

SEZNAM TABULEK

Tabulka 1: TAROS – technické údaje 12
Tabulka 2: Zapojení CAN pinů 26
Tabulka 3: Struktura řídící zprávy 30
Tabulka 4: Struktura řídící zprávy 31
Tabulka 5: Struktura rychlostní zprávy 34
Tabulka 6: Struktura zprávy pro aktuátory 35
Tabulka 7: Struktura zprávy pro manipulátor 36
Tabulka 8: Frekvence-perioda manipulátoru u předchozí jednotky 36
Tabulka 9: Možné varianty rychlostních zpráv (kruhový režim) 38
Tabulka 10: Struktura odpovědí od servopohonů 39
Tabulka 11: Strukutra odpovědi od aktuátorů 39
Tabulka 12: Možné chyby na servopohonech 40
Tabulka 13: Pozice jednotlivých chyb v daném bytu odpovědi 41

SEZNAM PŘÍLOH

1 Zdrojový kód řídící jednotky
2 Schéma přídavné desky
3 Fotografie řídící jednotky