ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA VÝKONOVÉ ELEKTRONIKY A STROJŮ

BAKALÁŘSKÁ PRÁCE

Pokročilé řízení servomotorů pro simulátor padákového kluzáku

Richard Sivera **2021**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Richard SIVERA**
Osobní číslo:	**E18B0097P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Elektrotechnika a energetika**
Téma práce:	**Pokročilé řízení servomotorů pro simulátor padákového kluzáku**
Zadávající katedra:	**Katedra výkonové elektroniky a strojů**

Zásady pro vypracování

1. Prostudujte základní principy ovládání padákového kluzáku a řízení servomotorů.
2. Prostudujte řízení zvolených výkonových servomotorů, jejich zapojení a komunikaci s driverem.
3. Navrhněte způsob řízení servomotorů pomocí mikrokontroléru a služby v řídícím počítači.
4. Implementujte a experimentálně ověřte algoritmus se zpětnou vazbou na základě dat ze senzorů a enkodérů.

Rozsah bakalářské práce: **30 – 40 stran**
Rozsah grafických prací: **podle doporučení vedoucího**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Obergruber, Julian & Mehnen, Lars. (2016). Development of a Paraglide Control System for Automatic Pitch Stabilization to Increase the Passive Safety. Procedia Engineering. 147. 26-31. 10.1016/j.proeng.2016.06.184.
2. How do paraglider controls work [online]. Aviation, 2018, 14.1.2018 [cit. 2020-04-16]. Dostupné z: https://aviation.stackexchange.com/questions/47514/how-do-paraglider-controls-work.
3. YOTOV, Nikolay a Nikolay TSAROV. Aerodynamics Theory for Beginners Paragliding Pilots. SkyNomad [online]. 2013 [cit. 2020-04-16]. Dostupné z: http://skynomad.com/articles/beginners-aerodynamics.html.
4. Encyklopedie fyziky – kinematika [online]. (c) 2019 [cit. 9.4.2019]. Dostupné z: http://fyzika.jreichl.com/main.article/view/4-kinematika.

Vedoucí bakalářské práce: **Ing. Petr Kropík, Ph.D.**
Katedra elektrotechniky a počítačového modelování

Datum zadání bakalářské práce: **9. října 2020**
Termín odevzdání bakalářské práce: **27. května 2021**

L.Š.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Prof. Ing. Václav Kůs, CSc.
vedoucí katedry

Abstrakt

Předkládaná bakalářská práce je zaměřená na vývoj programu pro pokročilé řízení servomotorů pro simulátor padákového kluzáku. Program bude běžet na mikrokontroleru Arduino Mega, který bude servomotory řídit. První část bude věnována problematice letu padákovým kluzákem. V druhé části budou rozebrány principy řízení servomotorů a obecné principy jejich regulace. Třetí část bude zaměřena na řízení zvolených výkonových servomotorů Kinco, na jejich funkce a použitý hardware. Za čtvrté bude představeno, jak jsem postupoval při vývoji řídícího programu a optimalizaci algoritmu v programovacím jazyce C++ pro aplikaci na simulátoru padákového kluzáku. Poslední, pátá část, bude věnována vyhodnocení výsledků řízení na základě praktických zkoušek. Cílem této práce je implementovat a ověřit algoritmus se zpětnou vazbou na základě dat z řídícího počítače.

Klíčová slova

Simulátor, pokročilé řízení, servomotor, mikrokontroler, padákový kluzák, algoritmus, C++, optimalizace, Arduino, Kinco

Abstract

This bachelor thesis deals with algorithm development for advanced control of paraglider simulator servomotors. This algorithm will run on Arduino Mega microcontroller, which will control the servomotors. First part will be about flying with paraglider theory. In the second part, we will discuss general principles for controlling servomotors and will present general ways of their regulation. Third part will present ways of controlling chosen Kinco servomotors, their function modes and used hardware. Fourth part will present how I developed the algorithm and its optimization for this specific application in C++ programming language. Last chapter will be dedicated to evaluation of practical results. Goal of this bachelor thesis is algorithm implementation and verification of its function based on control commands from computer.

Key words

Simulator, advanced control, servomotor, microcontroller, paraglider, algorithm, C++, optimization, Arduino, Kinco

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této bakalářské práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

.....

podpis

V Plzni dne 26.5.2021 Richard Sivera

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce Ing. Petru Kropíkovi, Ph.D. za praktické rady a kolegiální přístup. Také děkuji svým blízkým za podporu během mého studia.

Obsah

Úvod

Předkládaná práce se zabývá návrhem řízení servopohonů pro simulátor letu padákovým kluzákem. Létání padákovým kluzákem je sportem velmi nebezpečným. Každá příprava letu je časově náročná a pro nováčka může být celý proces učení velmi frustrující a náročný. Právě proto byl založen projekt FlyOnVision pro vyvinutí simulátoru, který má realisticky simulovat létání padákovým kluzákem pomocí VR brýlí a natáčením pilota v prostoru. Natáčení pilota má za úkol věrně simulovat síly, které na něj během letu působí. Právě proto simulátor obsahuje sedm servomotorů, z nichž pět polohuje pilota v prostoru navíjením lan a dva kladou pilotovi odpor při stahování řídících šňůr. Právě zde přichází na řadu úkol vyvinout funkční program pro řízení servomotorů.

Nároky na simulátory kterýchkoliv činností jsou vždy velmi vysoké a z hlediska řízení je kladen důraz především na rychlost a spolehlivost. Nedílným požadavkem je možnost servomotory dynamicky řídit a zadávat jim požadované parametry. Tato práce má za úkol navrhnout řízení servopohonů pomocí mikrokontroleru Arduino Mega, který bude ovládat pohony skrz jejich drivery za použití komunikačního rozhraní RS-232. Cílem je funkční ovládání servomotorů simulátoru na základě povelů z řídícího počítače.

První část práce je věnována základním poznatkům k teorii řízení padákového kluzáku, které jsou nezbytné pro realistické natáčení sedačky s pilotem servopohony. Druhá část se zabývá základními principy řízení servopohonů. Ve třetí části je popsáno řízení zvolených výkonových servomotorů, jejich zapojení a komunikace s mikrokontrolerem. Čtvrtá část už se bude zabývat samotným vývojem programu pro mikrokontroler s tím, že mikrokontroler musí umět komunikovat jak se servopohonem, tak řídícím počítačem a musí být schopen na základě povelů z řídícího počítače dynamicky měnit vlastnosti servomotorů. Program bude napsán v jazyce C++. V poslední části budou vyhodnoceny praktické experimenty o funkčnosti simulátoru z hlediska daných servopohonů.

Seznam symbolů a zkratek

$1f$,$3f$ ..... Počet fází

a ..... Zrychlení servomotoru (rad/s^2)

a_{int} ..... Zrychlení v interních jednotkách driveru (dec)

$CHKS$ ..... Kontrolní součet

DIN ..... Digitální vstup driveru

$d_{hřídele}$ ..... Průměr hřídele rotoru (mm)

ENC_RES ..... Rozlišení enkodéru (inc)

f ..... Frekvence (Hz)

f_s ..... Přenosová rychlost (Bd)

I ..... Proud (A)

I_{int} ..... Proud v interních jednotkách driveru (dec)

I_{maxcmd} ..... Maximální proud omezující maximální moment servomotoru (A)

I_N ..... Jmenovitý proud (A)

K_{cp}, K_{ci} ..... Zesílení regulační smyčky proudu

K_{pp} ..... Zesílení polohové regulační smyčky

K_{vp}, K_{vi} ..... Zesílení rychlostní regulační smyčky

L ..... Logický stav digitálního vstupu$v$ Rychlost servomotoru (ot/min)

M ..... Točivý moment servomotoru (Nm)

M_N ..... Jmenovitý točivý moment (Nm)

M_p ..... Přírůstek točivého momentu pro přepočet (Nm)

M'_p ..... Přírůstek točivého momentu pro druhou mocninu přepočtu (Nm)

M_Z ..... Základ točivého momentu pro přepočet (Nm)

n_N ..... Jmenovité otáčky (ot/min)

$Offset$ ..... Vychýlení pro režim dojetí na výchozí polohu (inc)

P_N ..... Jmenovitý výkon (W)

Q ..... Výsledek kódování pro digitální vstupy

RS-232 ..... Typ sériové komunikace

Rx ..... Vodič pro přijímání dat ze sérové linky

Tx ..... Vodič pro vysílání dat po sériové lince

V ..... Váha digitálního vstupu

v_{int} ..... Rychlost v interních jednotkách driveru (dec)

v_{max} ..... maximální rychlost servomotoru (ot/min)

X_a ..... Aktuální poloha řidiček (inc)

X_{max} ..... Maximální poloha řidiček (inc)

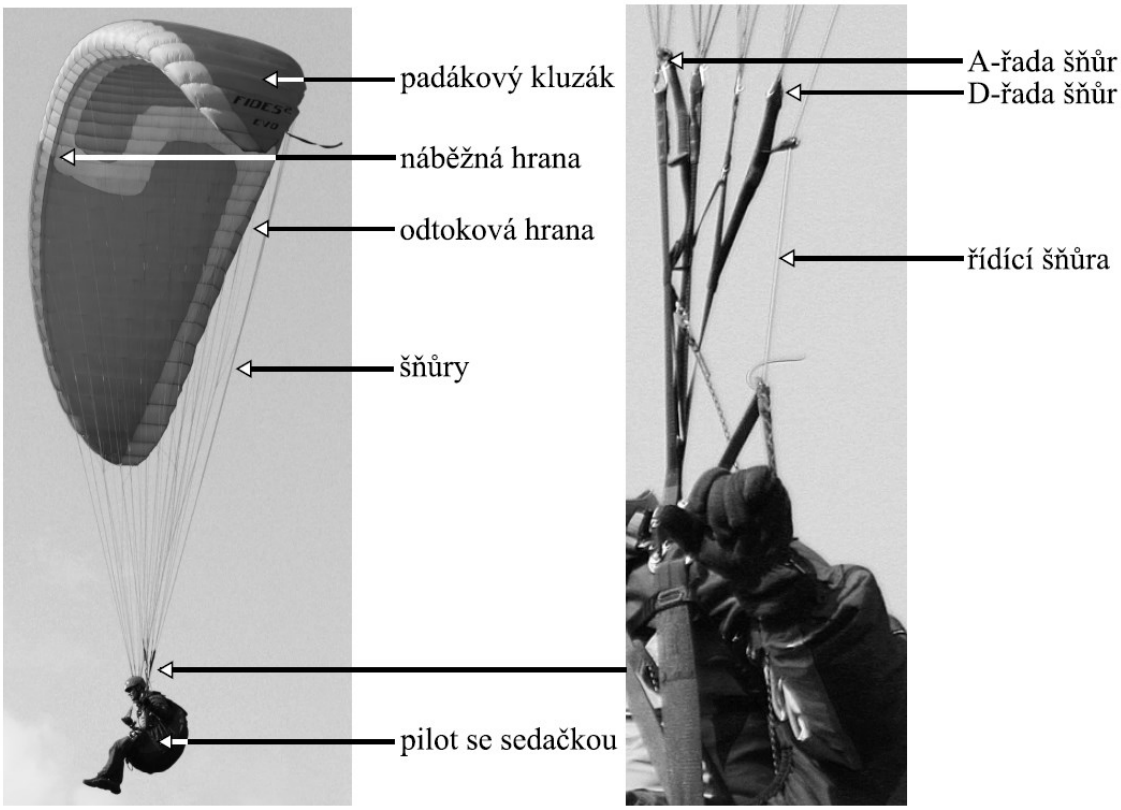
X_{pos} ..... Posun výchozí polohy řidiček vlivem natočení pilota (inc)

α ..... Úhel náběhu (°)

1 Poznatky k teorii řízení padákového kluzáku

1.1 Základní části padákového kluzáku

Komplet se skládá ze sedačky, padákového kluzáku a záložního padáku s dalšími periferiemi. Konstrukce kluzáku je vidět na následujícím obrázku.



Obr. 1.1: Základní části padákového kluzáku [4]

Umělá tkanina je hlavním materiálem padákového kluzáku. Vrchní část celého kluzáku se nazývá vrchlík. Je to skládatelná konstrukce mnoha propojených vnitřních komor, které se při letu po nafouknou a tvoří jeden souvislý tvar, tak jak je vidět na obrázku. Vrchlík se při správném letu padákovým kluzákem nafukuje samovolně náporem vzduchu. [1]

Na vrchlík jsou našity čtyři řady řídících šňůr, které jsou ve směru letu A, B, C, D přišity na různá místa vrchlíku. Řadou A se tedy ovládá náběžná (přední) hrana vrchlíku, zatímco řadou D se řídí jeho odtoková (zadní) hrana. Za řadou D jsou ještě vedeny řídící šňůry, ty jsou našity na boky a odtokovou (zadní) hranu vrchlíku. [1]

Všechny šňůry jsou poté spojeny do ocelového oka připojeného v karabině, která je připevněná k sedačce. Výjimkou jsou řídící šňůry, které jsou vedeny většinou malým ocelovým okem za D-šňůrami a jsou po celou dobu letu drženy pilotem.

1.2 Základní řídící prvky padákového kluzáku

V této kapitole budou představeny základní řídící prvky padákového kluzáku a jejich účinky na let. Ještě před jejich představením je ale nutné uvézt důležitou veličinu, která má na ovládání a let padákového kluzáku velký vliv. Tou veličinou je úhel náběhu α, který představuje úhel mezi tětivou profilu vrchlíku a vektorem rychlosti padákového kluzáku. Tento úhel ovlivňuje každý aspekt letu a musí se při letu udržovat v určitých mezích. Při překročení dovoleného úhlu náběhu se padákový kluzák chová nestabilně.

Řídící šňůry

Řídicí šňůry, zkráceně řidičky, napojující se na odtokovou hranu vrchlíku, zastávají funkci zatáčecí a brzdící. Při stažení řidičky na jedné straně vrchlíku se zadní část vrchlíku na této straně deformuje a zvýší se její aerodynamický odpor, díky čemuž druhá strana vrchlíku začne tuto stranu předbíhat a padákový kluzák tím začne zatáčet. [1]

Co se rozsahu chodu řidiček týče, zatímco školní padákové kluzáky mívají rozsah zhruba 140 cm, závodní verze mají z důvodu citlivějšího a rychlejšího ovládání rozsah pouze okolo 100 cm. [4]

Řidičky standartně kladou pilotovi odpor, který při jejich stahování není lineární, ale který se progresivně zvětšuje.

Zatáčení náklonem sedačky

Náklonem pilota v sedačce lze přesouvat váhu více k jednomu konci křídla a tím uvolňovat konec druhý. Tím vznikne mírná deformace vrchlíku viditelná na Obr. 1.2, která má za následek rozdílné generování vztlaku na levé a pravé polovině vrchlíku. [4]

Obr. 1.2: Pilot zatáčející náklonem sedačky

1.3 Let padákovým kluzákem

Díky mnoha ovládacím prvkům padákového kluzáku a rozmanitému charakteru povětrných podmínek existuje mnoho způsobů, jak padákový kluzák ovládat. V této kapitole budou popsány techniky startu, letu a nabírání výšky při letu padákovým kluzákem.

Start letu

Start je první fází letu a lze jej u padákového kluzáku realizovat dvěma hlavními způsoby. Tím prvním a klasickým způsobem je start rozběhnutím se ze svahu ideálně proti větru, ovšem startovat lze i s bočním nebo lehkým zadním větrem. Druhý způsobem startu je start pomocí navijáku. Při tomto startu je pilot připojen k lanu, které je rychle navíjeno. [1]

Klasický start spočívá v rozběhnutí se s padákem na dostatečnou rychlost, při které se vytvoří dostatečně velká vztlaková síla a padák se vznese do vzduchu. Tato rychlost bývá zhruba 25 km/h (závisí na povětrnostních podmínkách). [1]

Pro účely simulátoru lze zřejmě napodobit oba druhy startu, ovšem simulátor nedokáže svému uživateli napodobit rozběhnutí a práci nohou. Z tohoto důvodu bychom start na simulátoru mohli realizovat pouze přednastaveným rozběhem digitálního pilota v softwaru a pilot by padákový kluzák ovládal pouze řidičkami.

Základní dvě metody startu rozběhem jsou:

1. Čelní start – Start, kdy má pilot vrchlík za sebou a rozběhem proti větru se vznese. Před vzlétnutím musí zkontrolovat, že se vrchlík plně nafoukl vzduchem. [1]
2. Křížový start – Ve výchozí pozici pilot stojí zády k větru a vrchlík má před sebou. Levé a pravé šňůry jsou překřížené a pilot nechá vrchlík naplnit vzduchem. Poté se pilot otočí o 180 stupňů tak, že už šňůry již nejsou překřížené a rozběhem proti větru se vznese. [5]

Základní ovládání

Základní ovládání při letu je prováděno pomocí řidiček. Principy ovládání se dají vysvětlit dvěma následujícími způsoby.

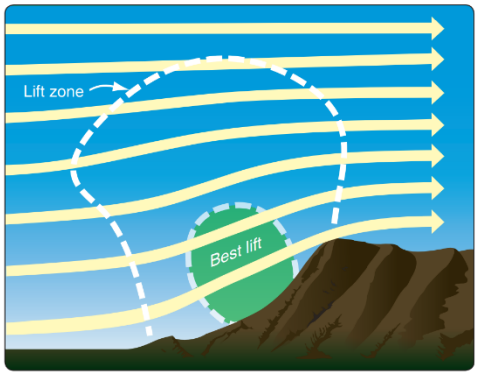
1. Přímý let – Stav, kdy padák letí ve vzduchu rovně dopředu. Pokud opomineme dynamicky měnící se povětrnostní podmínky, při přímém letu padák letí s určitou dopřednou rychlostí odpovídající úhlu náběhu α. Při současném stačení řidiček se dopředná rychlost sníží. [1]

2. Zatáčka – Provádí se přitažením jedné z řidiček. Na straně stažené řidičky se poté vrchlík na této straně přibrzdí a začne zatáčet. Ostrost zatáčení je přímo úměrná přitažení řidičky. [1]

Techniky nabírání výšky

Tato podkapitola bude věnována technikám nabírání výšky při letu s padákovým kluzákem. S padákovým kluzákem lze také pouze sletět z kopce dolů, ale i přes absenci jakéhokoli pohonu lze nabírat výšku následujícími způsoby.

1. Termické létání – Je pokročilým stylem létání, kdy se pilot snaží navádět svůj padákový kluzák na místa, na kterých se domnívá že budou stoupavé proudy, které jeho kluzáku umožní nabrat výšku. [1] Stoupavé proudy vzduchu jsou proudy vzduchu teplejšího, než je ten okolní. Díky vyhledávání stoupavých vzduchových proudů může padákový kluzák ve vzduchu setrvat téměř neomezenou dobu a urazit tak mnoho kilometrů. [5]
2. Svahování – Je spíš začátečnickou technikou udržování padákového kluzáku ve vzduchu následujícím způsobem. Pilot vyhledá svah, proti kterému zepředu proudí dostatečně silný vítr. Po vznesení pilot setrvává ve vzduchu nad tímto svahem. [1] Padákový kluzák je tedy zespoda ofukován a vynášen do výšky větrem, který se ohýbá o daný svah.



Obr. 1.3: Znázornění proudů vhodných pro svahování; Zóna kde se dá dosáhnout největšího vztlaku je označena zeleně [2]

1.4 Nebezpečné režimy letu

Při letu s padákovým kluzákem může dojít k určitým situacím které lze pokládat za nebezpečné, nebo nevhodné, zejména pro začínající piloty. Tyto režimy mohou nastat také chtěně, když si pilot přeje provést daný manévr. [1]

Přetažení a přebrzdění padákového kluzáku

Jedná se o nebezpečné stavy, kdy pilot řidičkami jednu stranu padákového kluzáku přibrzdí příliš. Mezní hodnota přebrzdění se také může měnit vhledem k aerodynamickým podmínkám. Uvedeme dva příklady přetažení:

1. Full stall – Stav, kdy se přílišným stažením obou řidiček zároveň dostane úhel náběhu vzduchu nad kritickou úroveň a dojde úplnému odtržení proudu vzduchu od vrchlíku. [5] Po dosažené minimální rychlosti vrchlík začne ztrácet svůj tvar a pilot padá se svým padákovým kluzákem k zemi. [1]
2. Negativní zatáčka – K tomuto stavu dojde při přílišném stažení jedné řidičky, zatímco má padák malou rychlost, nebo k němu dojde v jiných aerodynamický podmínkách. Při negativní zatáčce se proud vzduchu odtrhne jen od jedné strany vrchlíku, mluvíme o tzv. asymetrickém odtržení proudu vzduchu. [6] [1]

Nebezpečná deformace vrchlíku

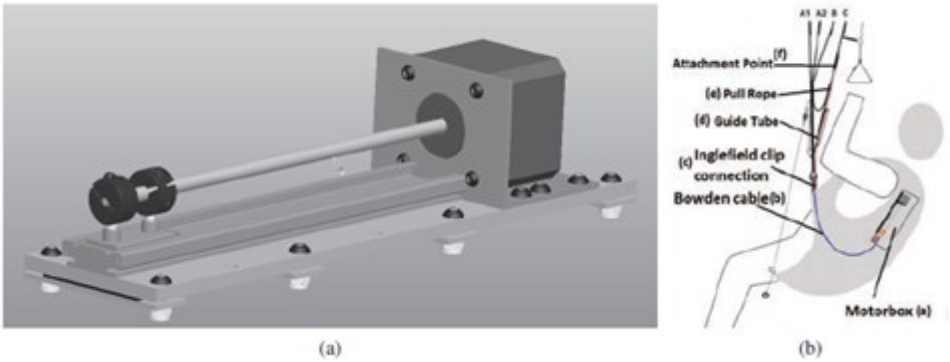
K nebezpečné deformaci (kolapsu) vrchlíku dochází zpravidla v turbulentních podmínkách, kdy vzhledem k nim pilot nesprávně ovládá svůj padákový kluzák. Aktivní pilotáží a správnými korekcemi lze většině těchto kolapsů předejít. [1]

Při kolapsu dochází k zaklopení některé části padáku následujícími způsoby:

1. Čelní zaklopení (Front stall) – K čelnímu zaklopení dochází v turbulentních podmínkách, kdy se setkávají dva vertikálně protiběžné proudy vzduchu, čímž se přední část vrchlíku může zaklopit směrem dozadu. [4] [1]
2. Asymetrické zaklopení – Situace, kde se jedna strana vrchlíku asymetricky zaklopí vlivem turbulentních podmínek, například na místě setkání protiběžných proudů vzduchu. Zaklopená strana vrchlíku má v tu chvíli větší odpor než jeho zdravá strana, a padákový kluzák začne směrem k zaklopené straně zatáčet. [1]

Situace nebezpečných režimů letu by se v simulátoru realizovala jen těžce a nepříliš věrohodně, ale bylo by možné připravit simulaci tak, aby se pilot mohl naučit těmto situacím předejít. K tomuto nácviku by v simulaci mohl být integrovaný vypínatelný podpůrný asistent, který by pilotovi pomáhal zvládat tyto situace a radil by mu, jak jim předejít.

Podobný asistent už byl navržen i pro reálné použití viz. [3], kde autoři vyvinuli mechanismus, který dokáže detekovat a automaticky zkorigovat tyto nebezpečné stavy detekcí úhlu náběhu vrchlíku pod bezpečnou úroveň. Potom, co elektronika detekuje nebezpečně nízký úhel náběhu vrchlíku, motor přitáhne D-šňůry a úhel náběhu vrchlíku se dostane zpět na přijatelnou úroveň. [3]



Obr. 1.4: (a) Lineární motor pro přitahování D-šňůr; (b) Celkový design mechanismu [3]

Podobně jako tento mechanismus by pak i podpůrný asistent v simulátoru pomáhal začínajícím pilotům korigovat nebezpečné stavy.

Možná implementace nebezpečných situací do simulátoru

Pro účely simulátoru by se daly jednotlivé nebezpečné situace nasimulovat do přednastavených scenárií, kde by si trénující pilot mohl nacvičit zvládání těchto situací. Pro každou situaci by bylo určené dané scenário a jednoduchou funkcí „opakovat“ by si mohl pilot i vícekrát zkusit, jak se z dané nebezpečné situace dostat.

Například pokud dojde v simulaci letu k natočení vrchlíku pod kritický úhel náběhu a dojde k čelnímu zaklopení, tak se zvýší aerodynamický odpor vrchlíku, který se tím zpomalí. Sedačka v tomto případě bude v poloze, ve které bude zadní strana sedačky níže než strana přední. To by simulovalo reálný stav, ve kterém pilot při čelním zaklopení předběhne vrchlík.

Při čelním zaklopení a kolapsu vrchlíku by se také znatelně odtížily ovládací řidičky padáku. To se na simulátoru promítne na nastavení točivého momentu servomotorů řidiček, které jejich šňůry tahají směrem nahoru. Točivý moment by se pro účely odtížení řidiček tedy nastavil na velmi nízkou úroveň.

2 Principy řízení servomotorů

V této kapitole se budeme zabývat obecnými principy řízení servomotorů. Servomotory jsou regulovanými elektrickými pohony, kterými lze dosáhnout velmi přesného polohování v mechatronických systémech. Systémy se servomotory se skládají z několika hlavních částí.

- Elektrický pohon s enkodérem.
- Napájecí výkonový měnič.
- Řídící a regulační obvody.

Přesnosti je dosaženo zejména použitím přeného enkodéru, který udává řídícímu systému změnu polohy v nespojitě v krocích. Délka kroku, tedy přesnost enkodéru, je dána jeho rozlišením.

Řízení servopohonů se realizuje v uzavřené regulační smyčce. Tato smyčka může obsahovat více vnořených regulačních smyček jako např. rychlostní, polohovou nebo proudovou. [7]

Negativně na regulaci servopohonů působí hlavně teplota okolí, vlhkost, voda a rušivé vlivy ze sítě (např. podpětí a napěťové špičky). [7] Nejvýznamnějším rušivým vlivem na simulátoru padákového kluzáku potom bude rušivý vliv změn zatěžovacího momentu, který se bude vzhledem k nepředvídatelným pohybům pilota měnit neustále.

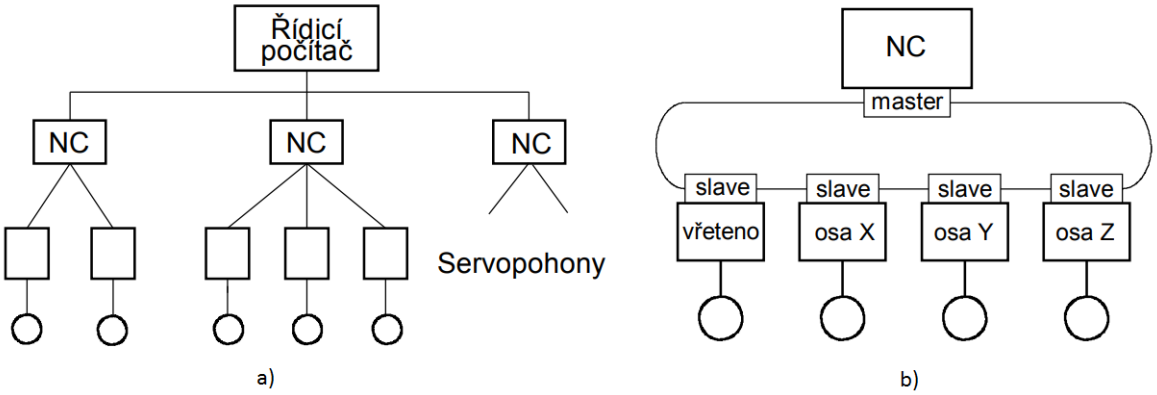
2.1 Hierarchická struktura řízení servomotorů

Při řízení servomotorů v daném systému se používá hierarchické struktury řízení servomotorů. Hierarchické řízení se používá zejména v rozsáhlejších systémech řízení. [7]

Hierarchická struktura se dá rozdělit do třech hlavních částí, od nejnižší po nejvyšší úroveň.

1. Servopohony – Jsou na nejnižší úrovni hierarchické struktury. Můžou to být autonomní rychlostní nebo polohové servomechanismy, zajišťující dynamiku pohybu a diagnostická hlášení. [7] Tato část se skládá ze samotného servomotoru a jeho driveru.
2. NC (Numerické řízení) – Střední úroveň hierarchické struktury, která zajišťuje zadávání cílové polohy, rychlosti nebo točivého momentu. [7] Každý NC řídí svůj servopohon v reálném čase a zároveň přijímá nejrůznější data o provozu servomotoru. Může být realizováno například mikrokontrolerem.

3. Řídící počítač – Je na nejvyšší úrovní hierarchické struktury řízení. S jednotlivými NC komunikuje většinou paralelní nebo sériovou komunikací. Paralelní komunikace je realizována společnou sběrnicí a je vhodná k řízení servomotorů v jednom mechanickém celku. Sériová komunikace jako kruhová síť se používá spíše pro řízení servomotorů na delší vzdálenost, jak je ukázáno na Obr. 2.1 b). [7]



Obr. 2.1: a) Hierarchická struktura řízení; b) Příklad komunikace kruhovou sítí [7]

Na Obr. 2.1 b) je vidět bližší pohled na řízení servomotorů. Je vidět, že je zde využíváno master/slave komunikace, která se v řízení servomotorů používá výhradně. Podobné hierarchické struktury budeme využívat i při realizaci řízení servomotorů simulátoru, kde PC bude realizován řídícím počítačem, NC bude realizovaný mikrokontrolerem a jednotlivé servomotory budou představovat jejich řídící drivery.

2.2 Způsob regulace servomotorů

Jak už bylo řečeno, servomotory jsou regulovány v uzavřené regulační smyčce. V regulační smyčce se nachází většinou několik zpětnovazebních smyček, konkrétně jde o zpětnou vazbu zápornou. [7] To znamená, že řídící veličina je porovnávána s regulovanou veličinou a výstupem je regulační odchylka, podle které se pohon přibrzdí nebo zrychlí.

Regulační odchylka je v regulačním obvodu dále zesílena, převedena na požadovanou veličinu a použita k natočení servomotoru ve směru takovém, aby se regulační odchylka snížila na nulu. Servomotory jsou pak většinou regulovány PWM modulací. [8]

Servopohony lze z hlediska regulace rozdělit na dva typy, a to rychlostní a polohový servopohon. Zatímco rychlostní servopohon je určen primárně k udržování zadané rychlosti, polohový servopohon sleduje požadovanou polohu (úhel natočení, absolutní poloha). [7] U simulátoru padákového kluzáku budeme využívat hlavně polohový druh servopohonu, jelikož budeme chtít sledovat určitou polohu odpovídající simulaci letu.

Polohové regulace lze dále rozdělit na dva základní typy:

1. Cílová – Slouží k dosažení cílové polohy v optimálním čase. Cílem je dosáhnout polohy v co nejkratším možném čase při maximálním možném zrychlení pro danou aplikaci. [7]
2. Sledovací – Využívá se pro sledování cílové polohy s přesně zadanou rychlostí. [7] Tohoto druhu regulace budeme využívat u řízení servomotorů simulátoru.

2.3 Způsoby zjišťování polohy

K zjišťování polohy hřídele servomotoru se používají snímače polohy (enkodéry). Mezi základní typy snímačů polohy pro servomotory patří:

1. Absolutní – U absolutního odměřování polohy má každá z možných poloh hřídele jasně definovanou polohu. Výhodou je, že je poloha hřídele jasná i po zapnutí servomotoru. [7]
2. Inkrementální – Jsou vysoce přesné a mají vysoké rozlišení, ale po každém zapnutí servomotorů je jakákoliv jejich poloha stanovena jako nulová. Po připojení je tedy nutné, aby servomotory najely na tzv. referenční (výchozí) bod, tj. nulovou polohu v dané ose. Skutečná poloha je dána obsahem čítače, který se při otáčení motoru inkrementálně zvětšuje podle odměřovacích impulsů. [7]
3. Cyklicky absolutní – Rozdílem oproti absolutnímu odměřování je odměřování polohy jen v omezeném rozsahu otáčení. [7]

Dále se snímače polohy rozdělují na přímé a nepřímé. Přímé snímače polohy jsou umístěné přímo na mechanických částech prováděné operace. Například by se mohlo jednat o umístění snímače polohy na lanu, které je navíjeno servomotorem na naviják.

U nepřímých snímačů polohy jsou rotační snímače umístěné přímo na hřídeli motoru, čímž ale vznikne dodatečná chyba řízení polohy vlivem nelinearity mechanických částí a převodů. [7]

Kvalitativně se dají enkodéry hodnotit podle svého rozlišení. Rozlišení u inkrementálních enkodérů je číslo vyslaných impulsů na danou jednotku délky. Tyto impulzy dále přicházejí do registru, který udržuje polohu hřídele motoru. Rozlišení se udává buď v PPM (pulzy na milimetr), PPR (pulzy na jednu otáčku) nebo v PPI (pulzy na palec). [9]

3 Řízení zvolených výkonových servomotorů

V této kapitole se budeme zabývat různými úhly pohledu na řízení konkrétních výkonových servomotorů pro simulátoru padákového kluzáku. Seznámíme se se zvolenými servomotory pro simulátor a principy jejich ovládání. Dále představíme jejich režimy ovládání a bude ukázáno zapojení driveru s řídícím mikrokontrolerem, který bude v tomto případě představovat blok numerického řízení z Obr. 2.1. Poté bude detailně popsána komunikace s driverem, práce s jeho digitálními vstupy a výstupy, a nakonec bude představen jednoduchý program pro zapínání driveru.

3.1 Zvolené servomotory pro simulátor

Pro účely padákového kluzáku byly už před mým působením na projektu zvoleny servomotory Kinco. Tyto servomotory byly zvoleny s ohledem na jejich aplikaci, požadovaný výkon a cenu. Nakonec byly zvoleny servomotory od čínského výrobce Kinco, u kterého je možnost objednat širokou škálu servomotorů. Výrobce k nim nabízí i drivery.

Zvolené servomotory řady SMH jsou střídavé motory vyznačují se výkonem od 50 W do 7,5 kW. Dále se v této bakalářské práci budu věnovat pouze následujícímu modelu, na kterém jsem vykonal nejvíce testů a který jsem používal pro účely programovaní. Na všechny ostatní motory se vztahují identické principy řízení.

Model servomotoru použitý při odlaďování programu je Kinco SMH60S-0040-30AAK-3LKH. Jeho specifikace je uvedena v následující tabulce.

Tab. 1: Specifikace uvedeného servomotoru Kinco [10]

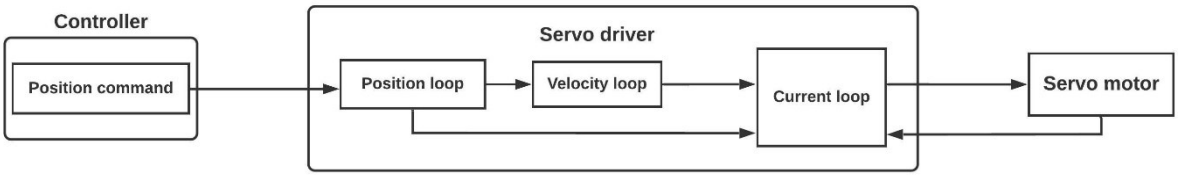
P_N (W)	M_N (Nm)	n_N (ot/min)	I_N (A)	$d_{hřídele}$ (mm)
400	1,27	3000	3,1	14

Uvedený servomotor se na simulátoru padákového kluzáku používá k ovládání řidiček. Řídící šňůra, kterou bude tento servomotor navíjet bude stahována rukami pilota, a proto by měl být jeho výkon naprosto dostačující.

Pro řízení servomotoru slouží driver Kinco CD422S-AA-000. Ten má vstup pro připojení do 1f sítě 230 V a jeho výstupem je řízené 3f napětí o maximálním výkonu 750 W a proměnnou frekvencí 0 až 400 Hz.

3.2 Princip ovládání zvolených servomotorů driverem

Servomotory Kinco jsou ovládány v uzavřené, zpětnovazební smyčce. V blokovém schématu řízení se nachází kontrolér, driver a zpětnovazební řídící smyčka. [10]



Obr. 3.1: Blokové schéma řízení servomotoru Kinco řady SMH [10]

Celá řídící smyčka servomotoru Kinco obsahuje celkem tři smyčky.

1. Proudová smyčka („Current loop“) – Primární cíl této smyčky je řízení točivého momentu. Proudová smyčka je smyčka vnořená oproti smyčce rychlostní. K regulaci slouží PI regulátor, jehož proporcionální a integrační zesílení lze nastavit parametry K_{CP} a K_{CI}. Kinco uvádí, že K_{CP} není třeba nastavovat a že K_{CI} je užíváno pouze ke kompenzování malých chyb v proudové smyčce. [10]
2. Rychlostní smyčka („Velocity loop“) – K nastavení rychlostní smyčky se zde používají parametry zesílení proporcionálního K_{VP} a integračního K_{VI} členu. K_{VP} přímo ovlivňuje velikost regulační odchylky, zatímco K_{VI} přímo ovlivňuje rychlost, se kterou se odstraní menší regulační odchylky. [10]
3. Polohová smyčka („Position loop“) – Kinco dovoluje nastavit pouze zesílení proporcionální složky regulátoru K_{PP}. Zvýšením zesílení K_{PP} dojde ke zrychlení odezvy, čímž se sníží doba dojetí na cílovou pozici. Ovšem Kinco uvádí, že nastavení zisku K_{PP} na příliš vysokou úroveň má za následek zvýšenou hlučnost nebo dokonce oscilace pohonu. [10]

Parametry jednotlivých smyček lze nastavit k tomu určenými interními proměnnými driveru. Proudová zpětnovazební smyčka přímo souvisí s parametry zvoleného motoru a její optimální parametry jsou už defaultně nastaveny pro každý motor. Parametry pro rychlostní a polohovou smyčku by měly být nastaveny podle charakteru zátěže. [10]

Enkodér servomotorů Kinco série SMH

Použitý střídavý servomotor SMH60S-0040 i další servomotory ze série SMH využité na simulátoru padákového kluzáku mají inkrementální enkodér. [10]

Inkrementální enkodéry jsou rotační snímače zjišťující informace o relativní změně polohy. Jak již bylo řečeno v kapitole 2.3, kvalitativně se dají rozdělit podle svého rozlišení.

Enkodéry v užívaných servomotorech řady SMH mají rozlišení 2500 PPR (pulzů za otáčku) a jsou kvadraturní, to znamená, že dokážou sledovat jak rychlost, tak směr otáčení. Enkodéry využívají tzv. enkódování typu X4, takže celkové rozlišení enkodéru je 10 000 PPR. [10] To nám umožňuje žjišťovat deset tisíc různých poloh na jednu otáčku hřídele. Také to znamená, že servomotor je schopen kterékoliv z těchto deseti tisíc poloh dosáhnout, přesnost natočení hřídele je tedy 0,036 stupně.

3.3 Základní režimy ovládání servomotoru

V této kapitole budou stručně popsány základní režimy funkčnosti servomotoru. Těch má servomotor celkově šest, z nich zde bude ovšem popsáno pouze pět, které budou použitelné pro daný způsob řízení mikrokontrolerem. Změnu režimu a nastavení jeho parametrů je možné provést v driveru. Popis režimů jsem vytvořil na základě jejich testování.

1. Režim rychlosti bez nastavitelné akcelerace (Speed mode)

   Po nastavení tohoto režimu v driveru a zadání požadované rychlosti servomotor dosáhne požadované rychlosti při své maximální možné akceleraci. Se svojí rychlostí si motor taky drží maximální točivý moment, ale motor se může zastavit, pokud překročí svůj nastavený max. proud.

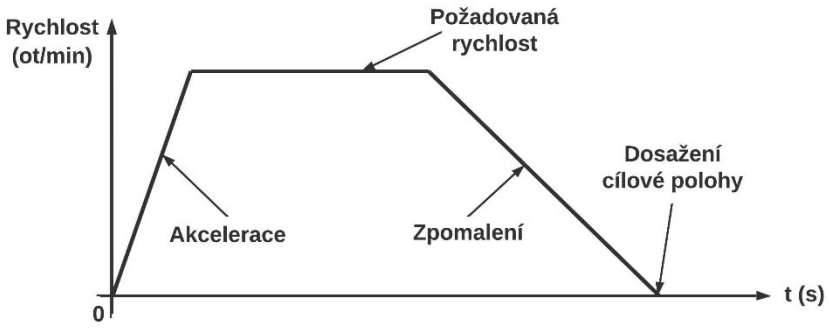
2. Režim rychlosti s nastavitelnou akcelerací (Speed mode‘)

   Pro použití tohoto režimu servomotoru je nutné nastavení požadované rychlosti, akcelerace a zpomalení. Po nastavení toho režimu se motor roztočí na požadovanou rychlost s požadovanou akcelerací a rychlost drží s maximálním točivým momentem. Při snížení požadované rychlosti servomotor sníží svou rychlost právě s nastaveným zpomalením. Při překročení nastaveného max. proudu se motor zastaví stejně jako v prvním režimu.

3. Režim dojetí na cílovou polohu (Position mode)

   Parametry nutné k použití tohoto režimu řízení servomotoru jsou požadovaná rychlost, akcelerace, zpomalení a cílová poloha. Po nastavení základních parametrů pro tento režim servomotoru se dojetí na cílovou polohu aktivuje následujícím způsobem. Je nutné tzv. aktivovat příkaz digitálním vstupem k tomu určeným. Aktivací příkazu, v tomto případě dojetí na cílovou polohu, se servomotor roztočí se zadanou akcelerací na požadovanou rychlost.

Rychlost si servomotor drží při maximálním točivém momentu, který je daný nastaveným max. proudem. Těsně před dojetím na cílovou polohu motor začne snižovat svoji rychlost s nastaveným zpomalením, až se na cílové poloze úplně zastaví.



Obr. 3.2: Závislost rychlosti na čase pro vysvětlení režimu dojetí na cílovou polohu [10]

4. Režim točivého momentu (Torque mode)

Tento režim umožňuje uživateli nastavit na servomotoru konstantní točivý moment. Před zapnutím tohoto módu je nutné nastavit požadovaný točivý moment v procentech (z maximálního možného točivého momentu pro servomotor) a maximální otáčky. Režim točivého momentu funguje následovně: po aktivaci režimu se motor bez zátěže roztočí na svoje nastavené maximální otáčky a točí se touto rychlostí, dokud proti jeho hřídeli nezapůsobí zátěžný moment o dané velikosti. V tuto chvíli se servomotor točí rychlostí, která je menší než nastavená maximální hodnota.

Pokud by byl zátěžný moment působící proti hřídeli servomotoru větší, než je nastavený kroutící moment, tak se hřídel servomotoru roztočí proti nastavenému směru otáčení. Takto je možno servomotor s nízkým nastaveným točivým momentem lehce přemoct například uchopením hřídele servomotoru rukou.

Tento režim řízení servomotorů simulátoru využívat u servomotorů řidiček. Díky možnosti pružně nastavovat točivý moment bude možné realisticky napodobit odpor řidiček padákového kluzáku. Zatímco servomotor bude naviják řidičky směrem nahoru, pilot je bude zase stahovat směrem dolů tak, jak je to i ve skutečnosti. Hlavní výhodou pak je možnosti přemoct servomotor, což bude nutné pro stažení řidiček směrem dolů.

5. Režim najetí na výchozí polohu (Homing mode)

   Tento režim se využívá k najetí na výchozí poloh. Je výhodné používat tento režim hned po zapnutí driveru. Po nastavení daných parametrů a vybrání jedné z možných 34 metod režimu je možno dosáhnout počáteční polohy například dojetím předmětu (který je ovládaný servomotorem) k fyzickému senzoru, nebo pootočením hřídele servomotoru o zadaný ofset.

   Došel jsem k závěru, že na simulátoru padákového kluzáku bude tento režim vhodné využít vždy při zapnutí simulátoru. Servomotory sedačku zvednou do výchozí polohy vhodné pro nasednutí a připoutání pilota před startem.

3.4 Použitý hardware

V této kapitole budou představeny základní komponenty a zapojení. RS-232 bude komunikačním rozhraním, které použijeme k propojení mikrokontroleru a driveru. Jedná se o sériovou komunikaci, která je u mnoha mikrokontrolerů běžná. Detailněji se komunikací budeme zabývat až v dalších kapitolách.

Servo Driver Kinco CD422S-AA-000

Driver propojený se 400 W servomotorem od stejného výrobce Kinco o výkonu 750 W, představený v kapitole 3.1. V následující tabulce jsou uvedeny jeho štítkové hodnoty.

Tab. 2: Štítkové údaje driveru Kinco [10]

Kinco CD422S-AA-000	
AC INPUT	AC OUTPUT
1PH AC220 -20/+15 %	3PH 0–Uin 4.0 A
47–63 Hz 5.5 A	750 W 0–400 Hz

Driver je schopen jmenovitého výstupního proudu 4 A a špičkového proudu 15 A. Napětí řídícího obvodu driveru je 24 V při proudu 1 A. [10]

Mikrokontroler bude k driveru připojen pomocí konektoru „X5“. Ten se u driveru používá ke komunikaci pomocí žádaného komunikačního rozhraní RS-232.

Mikrokontroler Arduino Mega 2560

Arduino Mega 2560, dále jen Arduino, je mikrokontroler využívající procesor ATmega2560. Obsahuje 54 digitálních vstupů/výstupů (15 z nich lze použít jako PWM výstup), 16 analogových vstupů, 4 porty pro sériovou komunikaci (kterou budeme využívat pro komunikaci s driverem) a 16 MHz oscilátor. Celková velikost paměti flash čítá 256 kB, z čehož 8 kB je použito pro zavaděč neboli „bootloader“. Dále deska obsahuje dva napájecí piny s napětím 5 a 3,3 V, které můžou dodávat proud o maximální velikosti 50 mA. Arduino obsahuje tři zemnící piny označené jako „GND“. Mikrokontroler je programovatelný přes Arduino Software (IDE) vývojové prostředí. Arduino nemá vlastní baterii a potřebuje externí napájení. [11]

3.4.1 Zapojení mikrokontroleru s driverem

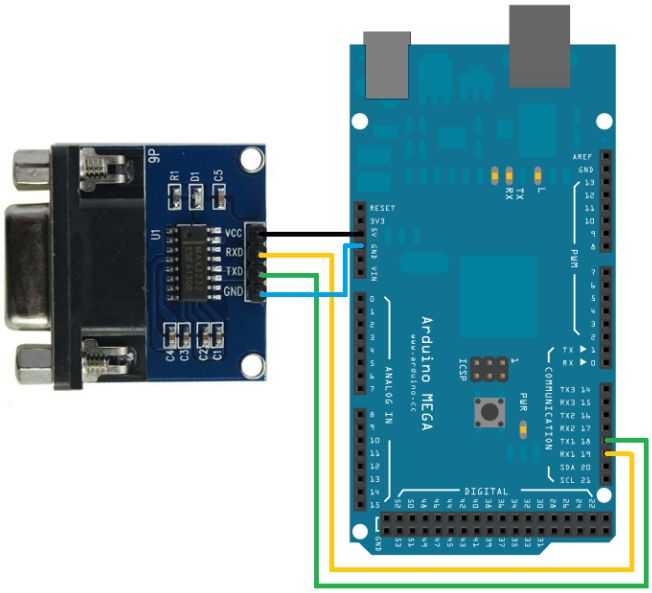
Jak již bylo zmíněno, pro komunikaci mezi použitým Arduinem a driverem byla vybrána sériová komunikace RS-232. U Arduina jsou k tomu určeny porty pro sériovou komunikaci, konkrétně zvolené Arduino má tyto porty čtyři. Každý port využívá dva digitální piny označené „Rx“ a „Tx“.

Sériová komunikace na Arduinu využívá TTL logických napěťových úrovní o velikosti 5 V. Pro logickou „1“ je tedy napětí rovno 5 V, zatímco pro logickou „0“ je napětí 0 V.

Naproti tomu RS-232 komunikace využívá napěťové úrovně o velikosti -5 až -15 V pro log. „1“ a +5 až +15 V pro log. „0“, které by mohly poškodit desku mikrokontroleru. [13] Proto mimo jiné nelze porty pro sériovou komunikaci připojit přes konektor přímo do driveru. K připojení driveru k Arduinu jsem tedy zvolil převodník mezi úrovněmi TTL na RS-232. Na desce převodníku je používán převodník MAX3232. Tento převodník lze napájet 3,3 nebo 5 V a jeho maximální přenosová rychlost s garantovanou přesností je 120 kbps, ovšem neměl by být problém dosažení rychlosti až 1 Mbps. [12] Driver pro sériovou komunikaci využívá výchozí rychlosti 38 400 Bd.[10] Cena převodníku s čipem MAX3232 je v řádu desítek korun a jedná se spíše o levnější převodník pro tyto účely.

Arduino lze pak pomocí převodníku připojit přímo k driveru. Pro připojení se používá kabel s devíti pinovým konektorem typu D-sub DE-9. Specifikem zde je, že pro propojení převodníku s driverem je nutné použít kabelu, který „Rx“ na straně driveru propojí k „Tx“ na straně převodníku a naopak. [10]

Na následujícím obrázku je zobrazeno zapojení Arduina s převodníkem. K propojení driveru s převodníkem je pak použit zmíněný propojovací kabel s dvěma konektory D-sub DE-9 typu samec.



Obr. 3.3: Zapojení Arduina s převodníkem

3.5 Komunikace mezi driverem a mikrokontrolerem

V této kapitole se budeme zabývat fundamenty komunikace mezi driverem a Arduinem. Zaprvé bude stručně představeno komunikační rozhraní RS-232, dále bude představen datový protokol používaného driveru a ovládání driveru digitálními vstupy a výstupy. Nakonec bude představen jednoduchý program pro zapínání driveru Arduinem.

3.5.1 Komunikační rozhraní RS-232 driveru

RS-232 je typem asynchronní sériové komunikace. Komunikace je realizována na dvou datových vodičích, jeden vysílací „Tx“ a druhý přijímací „Rx“. [14]

Komunikace s použitým driverem je založena na modelu master/slave.[10] Ve všech případech je driver slave, tedy on sám od sebe z hlediska komunikace sám od sebe nedělá nic a všechny informace si musí Arduino (master) vyžádat. Analogicky i každý požadavek na změnu otáček musí přijít od Arduina, za standartních podmínek provozu driver otáčky sám od sebe nemění. Na každý požadavek realizovaný Arduinem driver posílá odpověď, typicky jestli se přenos informací podařil.

Komunikační protokol používaný driverem používá datový paket s pevnou délkou 10 bytů.[10] Struktura datového paketu je následující.

Byte 0	Byty 1 až 8	Byte 9
ID	8 bytová data	CHKS

Obr. 3.4: Základní tvar datového paketu driveru [10]

ID značí číslo driveru, což je číslo identifikující daný driver a je defaultně nastaveno na hodnotu „1“. CHKS značí kontrolní součet (checksum), tedy součet všech ostatních bytů paketu. Kontrolní součet je jednoduchou metodou kontroly správnosti přenosu. Jeho výpočet je následovný. [10]

$$CHKS = -\sum(byte0, \dots, byte8) \tag{3.1}$$

Zde uvedu další důležité vlastnosti komunikačního protokolu driveru. [10]

- Přenosová rychlost 38 400 Bd (výchozí hodnota).
- Počet datových bytů 8.
- Počet stop bitů 1.
- Bez kontroly parity.

3.5.2 Datový protokol driveru

Datový protokol driveru má jasně definovanou formu a obsah. Určuje, co přesně má být obsaženo v datovém paketu driveru viz. Obr. 3.4. Jak již bylo ukázáno, jeden datový paket obsahuje 8 datových bytů.

Každý parametr a proměnná driveru (například rychlost otáčení) jsou jasně definovány svým unikátním tzv. indexem a subindexem. Index a subindex jednotlivých proměnných z interního uložiště driveru je možné najít v „Kincoservo+“ softwaru. Zde je možné jej vyčíst ze seznamu objektů (Object dictionary). Index a subindex jsou čísla vyjádřeny ve svých hexadecimálních hodnotách. [10]

Libovolná proměnná v driveru má také svůj definovaný datový typ, který může být jeden až čtyř-bytový a přirozeně může být znaménkový nebo bezznaménkový. U každé proměnné je také definováno, zda ji lze zapisovat i číst nebo jen jedno z toho. [10]

Nyní pro názornost uvedeme příklad komunikace mezi Arduinem a driverem, konktrétněji požadavek na zápis dat do driveru od Arduina a následně odpověď ze strany driveru. Bude zde popsána struktura datového paketu a její jednotlivé části.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CMD	INDEX		SUB	DATA 1 až 4			

Obr. 3.5: Datový paket odeslaný Arduinem [10]

V následujících bodech popíšeme datový paket odeslaný Arduinem z Obr. 3.5. [10]

1. CMD – Specifikuje směr toku dat a jejich velikost. Má celkem čtyři podoby.
   - 0x23 – Arduino zapíše 4 bytová data do driveru. Využijí se tedy byty 4 až 7 datového paketu.
   - 0x2B – Arduino zapíše 2 bytová data do driveru. Využijí se datové byty 4 a 5. Byty 6 a 7 zůstávají nulové.
   - 0x2F – Arduino zapíše 1 bytová data do driveru. Využije se pouze datový byte číslo 4. Ostatní byty zůstávají nulové.
   - 0x40 – Arduino pošle dotaz na hodnotu proměnné driveru.
2. INDEX – Index ze seznamu objektů, tedy adresa proměnné, do které mají být data zapsána.
3. SUB – Subindex ze seznamu objektů, tedy druhá adresa téže samé proměnné.
4. DATA – Žádaná hodnota pro zápis do driveru.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RES	INDEX		SUB	RESERVED			

Obr. 3.6: Datový paket odpovědi driveru [10]

Teď se zaměříme na datový paket odpovědi driveru z Obr. 3.6. [10]

1. RES – Blíže specifikuje správnost přenosu, nebo v případě odpovědi na dotaz.
   - 0x60 – Data byla úspěšně přijata.
   - 0x80 – Chyba, data nebyla přijata.
2. INDEX, SUB – Adresy jsou shodné s adresami paketu na Obr. 3.5.
3. RESERVED – Nevyužívají se, jejich bližší zkoumání nemá význam. Ve skutečnosti driver pošle zpátky stejné datové byty jako byly odeslány.

V případě, že by Arduino nechtělo do driveru data zapsat, ale naopak číst data z libovolné proměnné, by se datové pakety mírně lišily, jak je vidět na Obr. 3.7. Arduino v položce „CMD“ odešle dotaz (0x40) na hodnotu proměnné určenou daným indexem a subindexem a neodesílá žádná další data. Driver v položce „RES“ odpoví velikost odpovědi v bytech nebo chybové hlášení. [10] V položce „DATA“ je poté obsažena hodnota dotazované proměnné.

a)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CMD	INDEX		SUB	RESERVED			

b)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RES	INDEX		SUB	DATA			

Obr. 3.7: a) Dotaz odeslaný Arduinem; b) Odpověď driveru [10]

Důležitým rozdílem v odpovědi oproti předchozímu příkladu je položka „RES“, která specifikuje počet datových bytů obsahujících informace. Flag 0x43 značí 4 příchozí datové byty, 0x4B označuje dva byty a jeden byte je značen 0x4F.

3.6 Ovládání digitálních vstupů a výstupů

Velmi důležitým ovládacím prvkem driveru Kinco jsou digitální vstupy a výstupy. Zatímco digitálních vstupů má driver k dispozici 7, digitálních výstupů má k dispozici pouze 5. [10] Tyto vstupy je možné nakonfigurovat podle potřeby. Existují dva způsoby, jak digitální vstupy a výstupy driveru využívat.

Za prvé mohou být využívány hardwarově, tedy k driveru mohou být připojené zvolené periferie. To se provádí pomocí konektoru driveru „X1“.[10] K tomu je potom možné vyrobit desku plošného spoje na míru a k té připojit kupříkladu čidla mezních poloh, které se po překročení fyzické meze servomotorem sepnou a dají signál driveru, který servomotor zastaví.

Druhý způsob využívání digitálních vstupů a výstupů je způsob softwarový. Driver v sobě má interní proměnnou jednotlivých digitálních vstupů „din_simulate“, pomocí které lze digitální vstupy ovládat pomocí mikrokontroleru. Podobně je tomu u čtení stavů digitálních výstupů, kde lze z proměnné driveru „dout_status“ vyčíst stavy jednotlivých digitálních vstupů.

Z hlediska ovládání servomotorů mikrokontrolerem se budu nyní zaměřovat jen na softwarové ovládání digitální vstupů a výstupů. Pro ovládání servomotorů Arduinem je správné ovládání digitálních vstupů klíčové a bez nich servomotor ovládat nelze.

3.6.1 Digitální vstupy

Z faktu, že pro ovládání digitálních vstupů máme k dispozici pouze jednu proměnnou vyplývá následující: je třeba použít kódování, které zahrne všechny digitální vstupy do této jedné proměnné. Výrobce Kinco ve své dokumentaci označuje toto kódování jako „bitcode“. [10] V podstatě jde o analogii logického kodéru s tím, že má 7 vstupů a 1 výstup. Výstupem tohoto kódování je číslo od 0 do 255, které je součtem vah jednotlivých digitálních vstupů. Váhy jsou rozděleny podle mocnin dvojky. Nyní uvedu stručnou tabulku s příkladem.

Tab. 3: Příklad stavů digitálních vstupů

Digitální vstup *i*	DIN 1	DIN 2	DIN 3	DIN 4	DIN 5	DIN 6	DIN 7
Funkce	Enabled	Reset	Select mode	Activate	Limit +	Limit -	Homing
Váha V	1	2	4	8	16	32	64
Log. Stav L	1	0	0	0	1	1	0

Příklad je následující. Ke každému digitálnímu vstupu je přiřazena jedna z některých funkcí, ale není to pravidlem a digitální vstup mít funkci přiřazenou nemusí. Váha vstupu je vypočítána jako 2^{i-1}, kde i zančí číslo vstupu (číslováno od jedné). Podle logických stavů je poté výsledná hodnota kódování vypočtena následovně:

$$Q = \sum_{i=1}^{n} V_i \cdot L_i \ (-) \tag{3.2}$$

… kde V_i značí váhu příslušného digitálního vstupu a L_i jeho logický stav. Q je pak výsledkem kódování.

Po vypočtení výsledku kódování už jen stačí zapsat požadovanou hodnotu do interní proměnné „din_simulate“. Driver zareaguje úplně stejně, jako kdyby k příslušnému digitálním vstupu byla připojena fyzická periferie. Funkce pro práci s digitálními vstupy, kterou jsem vytvořil a používám ji v programu pro ovládání servomotorů, je uvedena v příloze A.

3.6.2 Digitální výstupy

Digitální výstupy se oproti digitálním vstupům ve standartních situacích neovládají, ale naopak je nutné je pro účely řízení servomotorů číst z driveru. To je možné čtením hodnoty proměnné „dout_status“, která v sobě uchovává logické stavy všech digitálních výstupů.

Po přečtení hodnoty z proměnné „dout_status“ je nutné dekódovat stav jednotlivých výstupů. Postup je analogický ke kódování digitálních vstupů v předchozí kapitole, ale postup je přesně opačný. V rovnici (3.2) bylo vysvětleno, jak se výsledek kódování digitálních vstupů počítá. Nyní ale známe hodnotu Q a zajímají nás jednotlivé stavy digitálních výstupů L_i.

Dekódování jsem realizoval tak, že se z přečtené hodnoty provádí dělení modulo dvěma a hodnota se půlí do doby, než je hodnota rovna nule. Výsledek se vždy uloží do prvku bytového pole, který už představuje konkrétní log. stav daného digitálního výstupu. Algoritmus pro výpočet je následovný.

```
byte x = 0;
while (bitcode > 0) {
   dataToPC.doutput[x] = (bitcode % 2);
   bitcode = (bitcode / 2);
   x++;
}
```

Z bytového pole, do kterého se stavy digitálních výstupů ukládají, je pak možno jednoduše zkontrolovat stav daného výstupu podle potřeby. Digitálních výstupů je dohromady 5, pole má tedy jen pět prvků kde nultý prvek představuje digitální výstup číslo 1, zatímco poslední představuje digitální výstup číslo 5. Celý kód funkce pro čtení digitálních výstupů z driveru je uveden v příloze A.

Funkce digitálních vstupů a výstupů

Na závěr kapitoly uvedeme, jakým způsobem se funkce driveru pro digitální vstupy nastavují. Dohromady totiž pro používaný driver existuje 27 různých funkcí pro digitální vstupy a 15 funkcí pro digitální výstupy, které se dají nastavit.

Každý digitální vstup má v driveru svou proměnnou, ve které je možné nastavit požadovanou funkci. Slouží k tomu proměnné „Din1_Function“ až po „Din7_Function“. Pro digitální výstupy k tomu zase slouží proměnné „Dout1_Function“ až po „Dout5_Function“. Všechny tyto proměnné datového typu bezznaménkového 16-bit integeru, a jejich funkce je možné nastavit daným hexadecimálním číslem. V následujících tabulce uvedu některé z funkcí pro digitální vstupy, které pro naše účely řízení najdou svoje použití.

Tab. 4: Příklady konkrétních funkcí pro digitální vstupy driveru [10]

Funkce	Hex. Číslo	Popis funkce
Driver enable	0x0001	Po nastavení n log. 1 se zapne servomotor.
Driver fault reset	0x0002	Resetuje chybová hlášení driveru.
Homing signal	0x0040	Signál z čidla výchozí polohy servomotoru.
Start homing	0x2000	Servomotor začne hledat výchozí polohu.
Activate command	0x4000	S náběžnou hranou servomotor dojede na cíl. polohu.

3.7 Program pro zapínání driveru mikrokontrolerem

Prvním krokem pro umožnění komunikace mezi Arduinem a driverem je zjištění základních nastavení driveru. Zjištění se provede připojením k driveru přes PC pomocí převodníku USB na RS-232 přes software „KincoServo+“. Všechna nastavení driveru se nacházejí v seznamu objektů. Ze seznamu objektů je nutné si poznamenat ID driveru a přenosovou rychlost. Ve výchozím nastavení jsou proměnné nastaveny na 1 a 38 400 Bd v uvedeném pořadí.

Nyní, po zjištění základních parametrů je možno připojit k driveru Arduino. Pro umožnění komunikace je nutné dodržení následujících podmínek.

- Přenosová rychlost na straně Arduina musí být stejná jako na straně driveru.
- Správně nastavená adresa driveru v Arduinu.
- Správné zapojení převodníku TTL na RS-232.

Po splnění zmíněných podmínek je možné navázat komunikaci. K navázání komunikace mezi Arduinem a driverem je nutno Arduino naprogramovat tak, aby komunikovalo pomocí stejného komunikačního protokolu jako driver. Nejzákladnějšími body programu pro vypnutí a zapnutí driveru mikrokontrolerem jsou:

- Spuštění sériové komunikace.
- Definice paketu pro vypnutí a zapnutí.
- Kontrolní součet.
- Odeslání paketu na příslušný sériový port.

V příloze B jsou tyto body ukázány.

4 Program pro řízení servomotorů

Tato kapitola bude věnována vývoji programu na řízení servomotorů pomocí mikrokontroleru Arduino a vytvoření vyhovující komunikace s řídícím počítačem v programovacím jazyce C++. Budu se zde zabývat svými poznatky, ke kterým jsem při vývoji došel a samotným postupem vytvoření programu. Zmíním taky další limitace, se kterými jsem se musel při psaní programu vypořádat.

Tato kapitola bude rozdělena do osmi částí seřazených podle postupu vývoje programu. Nejprve budeme pojednávat o základních požadavcích na program a o tom, co by měl vykonávat; dále se zaměříme na základní funkce na ovládání servomotoru; poté se přesuneme ke způsobu čtení dat z driveru; dále vysvětlím, jakým způsobem jsem vytvořil stavový stroj; poté bude následovat komunikace s řídícím počítačem; následně představím hlavní smyčku programu určující průběh řízení; jednu podkapitolu věnujeme optimalizaci programu, a nakonec představím funkci pro realizaci nelineárního odporu kladeným řidičkami.

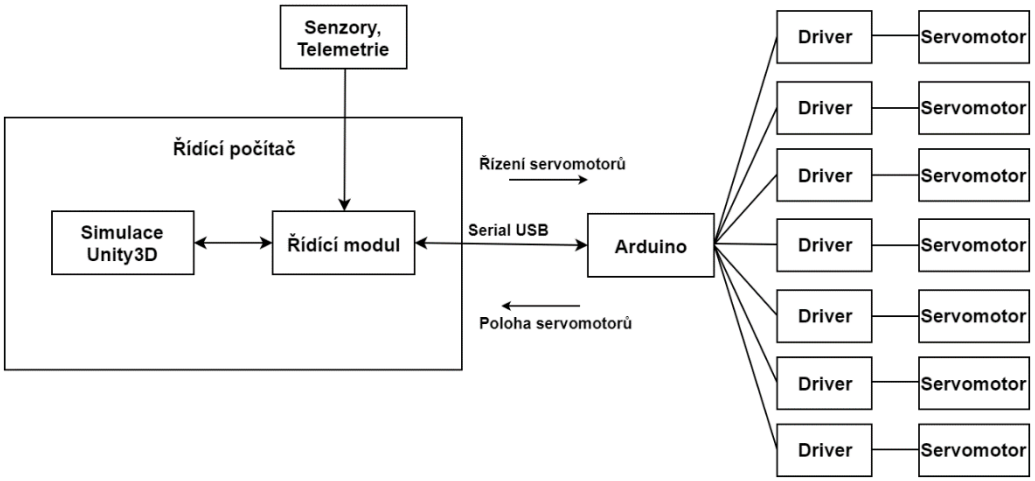
4.1 Stanovení požadavků na řídící program

V této kapitole stanovím, jaké funkce by pro daný účel řízení servomotorů na simulátoru padákového kluzáku měl program mít.

Mikrokontroler Arduino bude přímo komunikovat s driverem řídícím servomotor. Jak z hlediska jednoduchosti hardwaru, tak z hlediska finančního, by mělo jedno Arduino řídit hned několik servomotorů, kterých je na simulátoru sedm. Více servomotorů najednou lze z jednoho Arduina řídit následujícími způsoby:

1. Každý servomotor bude ovládán přes svůj vlastní sériový port (Arduino má čtyři).
2. Přes jeden port se bude ovládat více servomotorů – Toho lze dosáhnout pomocí zapojení vícero driverů do série. Každý z driverů má potom svoje unikátní ID, podle kterého se s daným driverem komunikuje. [10]

V případě využití možnosti číslo dva by Arduino muselo čekat, než jeden z driverů odpoví, než by mohl začít komunikovat s driverem dalším. Z důvodu rychlosti programu jsem tedy zvolil možnost číslo jedna, díky které bude moct Arduino komunikovat s více drivery paralelně. To je obzvláště důležité u aplikace jako je simulátor padákového kluzáku, kde je třeba mnohokrát za sekundu posílat příkazy driverům. Nevýhodou je, že k využití 1. možnosti bude nutné použití vícero mikrokontrolerů Arduino.



Obr. 4.1: Zjednodušený diagram řízení

Jak je z diagramu z Obr. 4.1 vidět, Arduino řídí servomotory na základě dat přicházejících z řídícího počítače. Tyto data jsou vypočítávána na základě výstupů senzorů a dalších telemetrických dat a na základě simulace. Řídící počítač tedy zadá příkaz Arduinu na zapsání dat do driverů a tím řídí servomotory. Arduino pak pošle zpět data o poloze servomotorů, která přečte z jednotlivých driverů, nebo jiná diagnostická data.

Teď, když jsme si představili funkci celého systému, můžeme stanovit požadavky na řídící program. Požadavky jsem rozdělil do následujících bodů:

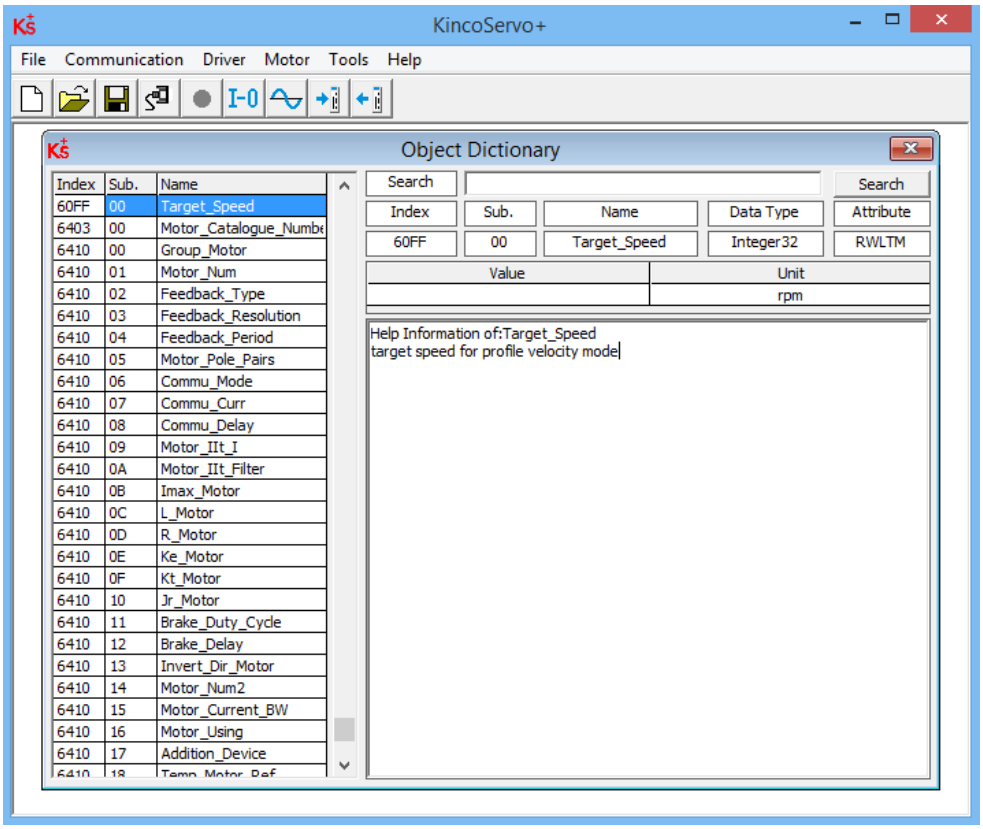
- Ovládat servomotor v libovolném režimu Arduinem.
- Číst libovolná data z driveru Arduinem.
- Možnost dynamicky řídit servomotor pomocí stavového stroje.
- Možnost všechny předešlé operace realizovat z řídícího počítače.
- Zajistit přijatelnou rychlost programu a komunikace.

Program jsem vyvíjel podle uvedeného pořadí požadavků a v následujících kapitolách se zrealizováním těchto požadavků budeme zabývat.

4.2 Vytvoření základních funkcí na ovládání servomotoru

Pro ovládání servomotoru je nezbytně nutné vytvořit funkce, které lze v programu volat a které dokážou servomotor ovládat požadovaným způsobem. Možné způsoby ovládání byly v kapitole 3.3 představeny jako režimy ovládání servomotoru. Mým úkolem tedy bylo pro realizování jednotlivých režimů a změnu jednotlivých parametrů napsat funkce.

Při tvoření funkcí jsem vycházel hlavně z manuálu [10], ve kterém jsou popsány způsoby, jakými lze požadovaného řízení dosáhnout. Dále jsem vycházel ze softwaru „Kincoservo+“, který je též nezbytný, jelikož obsahuje seznam objektů (proměnných). V seznamu objektů má každá proměnná jasně určeny svoje parametry pro přístup, jako je index a subindex, které jsou nezbytně nutné pro vytvoření datového paketu pro danou funkci. Právě vytvořením konkrétních datových paketů pro konkrétní funkce se servomotor ovládá. Pakety pro zapsání do proměnné driveru musí mít formát z Obr. 3.5.



Obr. 4.2: Seznam objektů v softwaru KincoServo+; každá proměnná má takto určené parametry

Na Obr. 4.2 je vidět dvoubytový index, jednobytový subindex a datový typ proměnné, který se musí v tomto datovém typu deklarovat i v řídícím programu. Dále je zde uveden atribut, který určuje, jestli se dá proměnná číst a zapisovat, nebo jen jedno z toho.

Další částí tvoření funkce je realizace přepočtu z inženýrských jednotek do interních jednotek driveru. Řekněme, že máme požadavek na rychlost 3000 *ot/min*. Tuto hodnotu ještě před odesláním do driveru musíme přepočítat podle uvedených přepočtů v dokumentaci [10]. V přepočtech se často objevuje konstanta „ENC_RES“, což je rozlišení enkodéru a je rovno 10 000 PPR. Přepočty jednotek jsou následující:

- Akcelerace a zpomalení: $a\ (rad/s^2)$

$$a_{int} = \frac{(a \cdot 65536 \cdot ENC\_RES)}{4 \cdot 10^6}\ (dec) \tag{4.1}$$

- Rychlost: $v\ (ot/min)$

$$v_{int} = \frac{(v \cdot 512 \cdot ENC\_RES)}{1875}\ (dec) \tag{4.2}$$

- Proud: $I\ (A)$

$$I_{int} = \frac{I \cdot 1.414}{1.050}\ (dec) \tag{4.3}$$

Tento výpočet se pro výpočet v programu ovšem nehodí, v programu je počítáno s celočíselnými proměnnými (z důvodu rychlosti výpočtů) a pro použití tohoto výpočtu by bylo potřeba počítat s proměnnými typu float. Proto je výhodnější počítat s celými čísly, jak je ukázáno v následující rovnici. Na základě testování byly z důvodu přesnosti ještě upraveny dané koeficienty.

$$I_{int} = \frac{I \cdot 1365}{10000}\ (dec) \tag{4.4}$$

Problém, se kterým jsem se v přepočtech setkal, bylo přetečení integeru. Většina proměnných (např. požadovaná rychlost, zrychlení atd.) jsou datového typu integer bez znaménka (na Arduinu „unsigned long“ 32bitový). Zde byl problém v tom, že při přepočtu rychlosti byla hodnota v čitateli v jednu chvíli větší, než je celková velikost tohoto datového typu. Při zadané rychlosti větší než 838 *ot/min* byla hodnota tedy větší než 2^{32} a výsledek vyšel špatně. Pro vyřešení tohoto problému jsem zavedl konstantu „SRF“, která je rovna výsledku všech konstant v rovnici. Její hodnota je následující:

$$SRF = \frac{512 \cdot ENC\_RES}{1875} = \frac{512 \cdot 10^4}{1875} = 2730 \tag{4.5}$$

Výsledný přepočet pro rychlost se tedy pak rovná „SRF“ krát rychlost. Totéž jsem musel udělat pro přepočet zrychlení, kde byl výpočet proveden analogicky. Konstantu pro zrychlení jsem nazval „ARF“.

Pro příklad uvedu následující funkci, která nastavuje požadovanou rychlost servomotoru.

```
void * target_speed(uint8_t adr, int32_t rpm){
      uint8_t packet[PL] = {adr, 0x23, 0xFF, 0x60, 0x00, 0x00, 0x00, 0x00,
                            0x00, 0x00}; // Definice paketu jakožto pole bytů

      rpm_recalc = (int32_t) (rpm * (int32_t)(SRF)); //Přepočet rychlosti

      int rpm_pos = 5;   // Pozice prvního datového bytu v paketu
      for (int pos = 0; pos < 4; pos++) {
           packet[rpm_pos + pos] = (rpm_recalc >> (8 * pos))
                                   & ((int32_t) 0xFF); // Bitové maskování
      }

      packet[PL-1] = eval_chksum(packet);  // Kontrolní součet
      send_packet();                       // Zápis do driveru
}
```

V uvedené funkci pro změnu rychlosti je velmi důležitý cyklus „for“, ve kterém se provádí tzv. bitové maskování. Provádí se kvůli rozdělení proměnné typu long do pole bytů.

4.3 Čtení dat z driveru

Druhou důležitou částí je blok programu, který zajišťuje rychlé a spolehlivé čtení dat z driveru, jako je například poloha z enkodéru. Dalším požadavkem kromě rychlosti a spolehlivosti čtení může být také možnost výběru, jestli data z driveru číst nebo ne. Čtení dat z driveru je totiž časově nejnáročnější operací programu.

Předtím než se zmíním o samotném postupu čtení dat z driveru je nutné definovat vstupní proměnné této funkce, které jsou celkem tři. První je adresa driveru, druhou je index proměnné a třetí je sériový port Arduina, ke kterému je driver připojen. Adresa a indexy jsou uvedeny v seznamu objektů softwaru „KincoServo+“ nebo v dokumentaci driveru [10].

Formát paketu pro dotaz na data z driveru musí dodržovat formát, který je znázorněn na Obr. 3.7. Průběh dotazu na data z driveru probíhá následovně:

1. Arduino odešle paket, který se dotazuje na danou proměnnou.
2. Posléze driver pošle zpět paket, který už obsahuje dotazovaná data.

Přijmutím datového paketu z driveru ještě funkce nekončí. Zaprvé je naprosto nutné zajistit, aby funkce při přijmutí jiného než dotazovaného paketu, paket ignorovala a čekala na ten správný.

Příkladem může být situace, kdy Arduino do driveru zapsalo informace o cílové pozici a hned poté se dotázalo na stav aktuální pozice. Ve zmíněné situaci driver nejprve prvním paketem odpoví, že se data o cílové pozici správně/nesprávně zapsala do driveru a druhým paketem bude odpovídat na dotaz aktuální pozice. Proto je tedy nutné implementovat i funkci se sérií podmínek „if“, který bude požadovaný paket rozlišovat a číst. Pozor, i při čtení dat z driveru může dojít k chybě, v takovém případě je nutné data číst znovu.

Funkci pro rozpoznání správného paketu jsem realizoval následovně. Zpočátku ještě dodávám, že uvedená funkce je volána ve funkci pro čtení dat z driveru a je opětovně volána ve smyčce „while“, dokud následující funkce nevrátí hodnotu pravda (true).

```
bool correct_packet (uint8_t packetReceived[PL], uint8_t packetSent[PL],
                    serial_port port, IndexesForRead index) {

    uint32_t index_received = 0;
    index_received = packetReceived[2];
    index_received += (packetReceived[3] << 8);
    index_received += (packetReceived[4] << 16);

    if(index_received == index) {              // Kontrola žádaného paketu
         if((packetReceived[1] == 0x43) || (packetReceived[1] == 0x4B)
         || (packetReceived[1] == 0x4F))       // Správný paket
             return true;
         else if (packetReceived[1] == 0x60)  // Potvrzovací paket nechceme
             return false;
         else {  // Zbývá pouze chybové čtení; dotaz se pošle znovu
             send_packet_to_port(packetSent, port);
             delayMicroseconds(500);
             return false; }
   }
   else
       return false;
}
```

Popis funkce je následující. Nejprve je 3 bytová adresa proměnné, obsáhlá ve třech separátních bytech paketu, který přišel z driveru, bitovými posuny přesunuta do proměnné „index_received“. Poté následuje základní podmínka zjišťující, zdali se adresa přečtená shoduje s adresou požadovanou. To zaručí odfiltrování ne právě požadovaných dat, které mohli mezitím ze strany driveru přijít do bufferu Arduina.

Dále přichází na řadu tři podmínky určující charakter přijaté zprávy. Charakter je určen druhým bytem paketu, tedy bytem na pozici „1“. Zaprvé jsem vytvořil podmínku určující, zdali paket obsahuje 4, 2 nebo 1 bytovou informaci. V takovém případě je to ten správný

paket a úkol této funkce je tímto splněn. Druhá podmínka odfiltruje potvrzovací pakety (flag 0x60). Třetí a poslední podmínkou je zjištění chyby (flag 0x80). To je schováno v podmínce „else“, ve které dochází k opětovnému odeslání dotazu.

Uvedená funkce „correct_packet“ tedy počítá se všemi možnými scénáři a zajišťuje, aby byla z driveru přečtena ta správná data.

Dalším úkolem pro mě v této funkci bylo v případě čtení více bytových proměnných tyto jednotlivé byty z paketu umístit do jedné proměnné. K tomu slouží opět série tří podmínek „if“ které mají za úkol rozlišit počet příchozích bytů obsahujících data. Postup je proveden bitovými posuvy a je proveden analogicky k bitovým posuvům na začátku uvedené funkce „correct_packet“.

Posledním úkolem bylo ve specifických případech přepočítání z interních jednotek driveru (dec) do inženýrských (fyzikálních) jednotek. Tyto přepočty jsou uvedeny v kapitole 4.2 a zpětné přepočtení je pouhým vyjádřením fyzikálních jednotek z uvedených vzorců. Rozlišení, zdali je přepočet třeba, jsem provedl několika podmínkami „if“ určujícími, zdali se právě četla proměnná vyžadující zpětný přepočet.

Celá funkce pro čtení dat z driveru nakonec vrátí přijatou a případně přepočtenou hodnotu požadované proměnné.

4.4 Vytvoření stavového stroje

Při tvoření programu pro řízení servomotorů jsem měl v tomto stádiu hotové základní funkce pro ovládání servomotoru a funkci pro čtení základních dat o jeho běhu. Byl jsem schopen tedy odzkoušet jednotlivé funkce a různě ovládat motor vždy drobným přepsáním programu a jeho opětovným nahráním. To je ale zcela vyloučené pro finální provoz a nyní jsem měl za úkol vytvořit tzv. stavový stroj (angl. „state machine“).

Myšlenka stavového stroje je následovná: funkce obsahuje několik režimů ovládání servomotoru tak, že pro změnu daného režimu stačí například stisknout tlačítko. Může se například jednat o změnu režimu ze stavu „vypnuto“ na „zapnuto“. V praxi se poté budou příslušné stavy měnit na základě dat příchozích z řídícího počítače, ale to až v další kapitole.

Nyní je třeba si nadefinovat, které režimy (stavy) budou na aplikaci servomotorů pro simulátor padákového kluzáku potřebné. Vycházím z dostupných režimů servomotoru Kinco popsaných v kapitole 3.3.

1. Servomotor zapnutý.
2. Servomotor vypnutý.
3. Režim točivého momentu – rozdělený na dva směry otáčení.
4. Režim dojetí na cílovou polohu.
5. Režim dojetí na výchozí polohu.
6. Zastavení servomotoru.
7. Okamžité zastavení v případě nouze.
8. Vynulování polohy v enkodéru.
9. Resetování chybových hlášení driveru.
10. Restartování Arduina.

Toto jsou základní stavy pro stavový stroj, které jsem postupem vývoje programu definoval. Režim rychlosti jsem ve stavovém stroji záměrně nepoužil, protože pro simulátor padákového kluzáku postrádá význam. Při běhu simulátoru bude mít význam pouze režim točivého momentu, kterým se bude nastavovat moment pro řidičky a režim dojetí na cílovou polohu, který bude používán polohovacími servomotory, které natáčejí pilota v prostoru. Nesmím také opomenout důležitý režim dojetí na výchozí polohu, který bude volán vždy po zapnutí simulátoru proto, aby se všechny servomotory dostaly do své výchozí polohy a tam nastavily svou nulu.

Dalším stavem je například okamžité zastavení v případě nouze, který by se mohl využít například při překročení dané krajní pozice. V takovém případě je ohrožen pilot na simulátoru a je prioritou ihned vypnout všechny servomotory.

Dalším užitečným stavem je resetování chybových hlášení driveru a restartování Arduina. Driver má svoje interní hlídání chyb, a i když by se při běžném provozu servomotorů chyba objevit neměla, stále je to možné. V takovém případě se chyby jednoduše resetují.

Posledním, ale neméně důležitým stavem je také zastavení servomotorů. Tento stav servomotory jednoduše uvede do ustáleného stavu bez tahu a servomotory pak drží svoji pozici. To se například hodí v situaci, kdy jeden pilot ze simulátoru odchází a druhý přichází, nebo když je pilot sice na simulátoru posazen, ale ještě se nachází v menu simulátoru. Realizaci tohoto stavu jsem provedl jednoduchým způsobem, a to sice nastavením nulové rychlosti.

Struktura stavového stroje

Nedílnou součástí tvoření stavového stroje bylo nadefinování, jakým způsobem se bude daný stav rozlišovat. Víme, že volba stavu se bude realizovat z řídícího počítače, takže Arduino bude stav vybírat na základě velikosti dané celočíselné proměnné příchozí z počítače. Strukturu stavového stroje jsem tedy nadefinoval následovně.

```
void state_machine (uint8_t mode_command) {
    set_state(mode_command);

    switch (state) {
     case position:
     …
     break;
    …
    }
}
```

Vstupní proměnná určující režim motoru „mode_command“ se předá funkci „set_state“, která pomocí podmínek „if else if“ určí daný stav pro konkrétní motor na základě této proměnné. Proměnnou „state“ jsem definoval pro přehlednost programu jako výčtový typ (enum). Nakonec se ve stavovém stroji na základě vybraného stavu vybere stav pomocí větvení programu způsobem „switch case“. Kód stavového stroje pro jednu řidičku se nachází v příloze C.

4.5 Komunikace s řídícím počítačem

Teď, když jsme schopni servomotor ovládat pomocí stavového stroje a číst z něj data je na řadě vytvoření vyhovující komunikace s řídícím počítačem. Na komunikaci se vtahuje pár základních nároků, kterými jsou: co nejvyšší rychlost, spolehlivost a možnost měnit množství předávaných dat z počítače.

Komunikace mezi počítačem a Arduinem probíhá přes USB. Tento typ komunikace je bez problému schopen dosahovat rychlosti 115 200 Bd. Nyní je nutné definovat data pro přenos.

```
typedef struct pc_data {
    // Datový blok pro driver 1
    int32_t position;
    uint16_t milli_amps;
    uint32_t rpm_profile;
    uint32_t acceleration;
    int16_t torque;
    uint32_t max_rpm;
```

```
    uint8_t mode_command; // Proměnná určující požadovaný režim motoru
    // Další datové bloky
    …
};
```

Pro data jsem vytvořil datovou strukturu. Výhodou takové struktury je přehlednost, data se nachází na jednom místě programu a další data je možné jednoduše přidat.

Vytvořená datová struktura obsahuje datové bloky pro jednotlivé servomotory, ve kterých jsou obsaženy proměnné například pro cílovou pozici, velikost proudu v miliampérech nebo požadovaný točivý moment.

Důležité pro pozdější představování je také zmínit, jak je struktura velká. Velikost se udává v bytech. Zjistit se dá pomocí funkce „sizeof“. Jeden uvedený datový blok obsahuje 21 bytů, celkově struktura obsahuje datové bloky pro drivery čtyři. Velikost celé datové struktury je tedy 84 bytů.

Přijímání dat Arduinem z počítače

Po vytvoření datové struktury jsem měl za úkol vybrat vhodný způsob přijímání těchto dat z počítače Arduinem. Jak jsem již zmínil, přenos informací musí být naprosto spolehlivý a běžně by nemělo při komunikaci docházet k žádným přeslechům. Měl jsem dvě možnosti, jak přijímání informací z hlediska softwaru realizovat:

- Přijímat data po jednom bytu do daného pole, ze kterého by bylo možné po dokončení příjmu nakopírovat data do struktury pomocí funkce pro kopírování paměti z jednoho místa na druhé „memcpy“.
- Použít funkci Arduina „readbytes“, která přečte všech 84 bytů dat najednou a umístí je rovnou do cílové datové struktury.

Po zvážení obou možností jsem dospěl k závěru, že použití druhé možnosti bude mít nespornou výhodu oproti možnosti první, a to ve vypuštění funkce „memcpy“. Funkce „readBytes“ se volá následovně.

```
Serial.readBytes((uint8_t*)&dataToDriver, sizeof(dataToDriver));
```

Funkce má dva argumenty. První určuje, do jaké proměnné se mají přečtené byty zapsat. Tam se ale proměnná datové struktury „dataFromPC“ nedá napsat přímo, protože funkce „readBytes“ zde přijímá jenom proměnné typu char nebo byte. Kompilátor při přímém odkázání se na proměnnou struktury hlásí chybu. Správným využitím ukazatelů jsem problém vyřešil. Druhým argumentem je počet bytů datové struktury určený pomocí funkce „sizeof“.

Následně jsem program začal testovat a do Arduina posílat 84 bytů dat přes terminál. Při zkouškách jsem zjistil, že Arduino přijímá pouze prvních 64 bytů a zbylých dvacet se kdesi ztratilo.

Po krátkém hledání jsem v referenci pro Arduino Mega 2560 zjistil, že Arduino má výchozí velikost bufferu pro sériovou komunikaci jenom pouhých 64 bytů [11]. Jeho velikost jsem tedy potřeboval zvětšit. O velikost sériového bufferu se stará interní knihovna Arduina „HardwareSerial.h“. Bohužel, tato knihovna neobsahuje jediný příkaz, kterým by se velikost bufferu dala zvětšit. Otevřením zdrojového kódu knihovny a přepsáním hodnoty „RX_BUFFER_SIZE“ z 64 na 256 bytů jsem problém vyřešil.

Příchozí data z počítače se nyní čtou správně a jako plus je i velké množství dodatečného místa v bufferu, což se může hodit pro budoucí úpravy. Požadavkem je, aby byly jednotlivé proměnné z počítače posílané v pořadí, v jakém jsou za sebou deklarovány v datové struktuře.

Odesílání dat z Arduina do počítače

Nyní se budeme zabývat opačným případem a tím je odesílání dat z Arduina do počítače. To je pro ovládání servomotorů pro simulátor padákového kluzáku velice důležité, neboť simulace v unity potřebuje pro správné polohování servomotorů i zpětná data o poloze přečtená z driveru. Ve smyčce programu, ve které Arduino neustále běží, se bude totiž volat funkce pro čtení dat z driveru „read_packet“ z kapitoly 4.3 a následně se budou přečtená data posílat do počítače. Postup bude vypadat následovně.

```
position = read_packet(controller_adr, CURRENT_POS, portSerial1);
torque   = read_packet(controller_adr, CURRENT_TORQUE, portSerial1);

Serial.print(ARD_NUM); // Číslo Arduina
Serial.print(";");
Serial.print(position);
Serial.print(";");
Serial.print(torque);
Serial.println();
```

Pro krátkost příkladu jsem uvedl pouze čtení dvou proměnných: aktuální pozice servomotoru a jeho aktuálního točivého momentu na hřídeli. Data jsou tedy přečtena pomocí funkce „read_packet“, kde jsou vstupními proměnnými adresa driveru, index proměnné neboli její umístění v paměti driveru a port Arduina, ke kterému je driver připojený. Následně jsou přečtená data odeslána do počítače pomocí funkce „print“.

Protože budou k počítači připojena dvě Arduina řídící celkově 7 servomotorů, je třeba data od dvou Arduin nějakým způsobem rozlišit. To jsem realizoval odesláním čísla Arduina před samotnými daty, což je konstanta zvolená pro konkrétní Arduino. Všechna odeslaná data jsou separována středníkem.

4.6 Hlavní smyčka řídícího programu

U mikrokontrolerů je většinou zvykem, že hlavní běh programu se provádí v nekonečné smyčce. Tato nekonečná smyčka je většinou umístěna v hlavní funkci programu „main“, ale u Arduina je to uděláno jinak. Arduino obsahuje funkci „setup“ a funkci „loop“. Jejich funkce jsou následující:

- Setup – Funkce, která se spouští pouze jednou po spuštění Arduina. Slouží k inicializování proměnných a spuštění sériové komunikace na začátku programu.
- Loop – Funkce spouštějící se po funkci setup, jedná se o nekonečnou smyčku, ve které je obsažen samotný běh programu a řízení.

V následujících dvou podkapitolách představím obsah těchto dvou funkcí, abych přiblížil běh programu.

4.6.1 Funkce pro inicializaci funkcí

První na řadu tedy přichází funkce setup, která obsahuje inicializaci sériové komunikace a vymazání obsahu sériového bufferu.

```
void setup() {
    Serial.begin(115200); // Inicializace pro rychlost 115 200 Bd
    while(!Serial)
        Serial.begin(115200, SERIAL_8N1);

    Serial.setTimeout(4);

    init_serial1(); // Inicializace komunikace pro driver 1
    init_serial2(); // Inicializace komunikace pro driver 2
    init_serial3(); // Inicializace komunikace pro driver 3

    clear_buffer(); // Vymazání obsahu bufferu
}
```

První čtyři příkazy mají za úkol inicializaci sériové komunikace na portu „0“, tedy na portu, který komunikuje s řídícím počítačem. Příkaz „setTimeout(4)“ nastavuje, jak dlouho má program čekat na příchod dat z počítače. Tím se omezí případné prodlevy při chybách v komunikaci.

Další tři příkazy inicializují komunikaci s drivery. Postup je shodný s inicializací komunikace pro řídící počítač.

Posledním příkazem je vymazání obsahu bufferu. To se po inicializaci provádí z důvodu, že driver po inicializaci komunikace posílá potvrzovací pakety do bufferu Arduina. Pro běh programu jsou tyto informace naprosto nepodstatné, a proto je vhodné je vymazat. K vymazání jsem se uchýlil z důvodu, aby nebylo dále v programu zdržováno čtení dat pomocí funkce „read_packet“.

4.6.2 Nekonečná smyčka programu

Jak již bylo zmíněno v úvodu kapitoly, funkce pro nekonečnou smyčku „loop“ je spuštěna hned po funkci „setup“. Jakmile se smyčka spustí, už se do konce běhu programu nezastaví. V jejím obsahu se běžně nacházejí funkce, které definují funkci celého programu. Její zjednodušená struktura je následující:

```
void loop() {
    newData = false;
    newData = read_data_from_pc();

    if (newData) {
        event_state_machine_ridicky(controller_adr);
        clear_buffer();

        read_data_from_drivers();
        send_data_to_pc();
    }
}
```

Smyčka má za úkol přečíst příchozí data z řídícího počítače. Jen tehdy, pokud nějaká data přišla, má za úkol tyto data zapsat do driveru. Poté se přečtou požadovaná data z driveru, jako například momentální rychlost a pozice, a pošlou se zpět řídícímu počítači na zpracování. Smyčka se tedy opakuje pořád dokola a Arduino tedy s driverem komunikuje jen tehdy, když si to přeje řídící počítač. Tím je realizován tzv. eventový stavový stroj.

Při přijmutí dat se vykonají následující úkony v pořadí uvedeném v útržku kódu:

1. Vykonání stavového stroje podle přijatých dat z řídícího počítače.
2. Vymazání obsahu bufferu, ve kterém se mezitím nahromadily potvrzovací pakety vlivem vykonání stavového stroje, tedy zapsáním požadovaných hodnot do driveru.
3. Přečtení požadovaných dat z driveru.
4. Odeslání přečtených dat zpět do řídícího počítače.

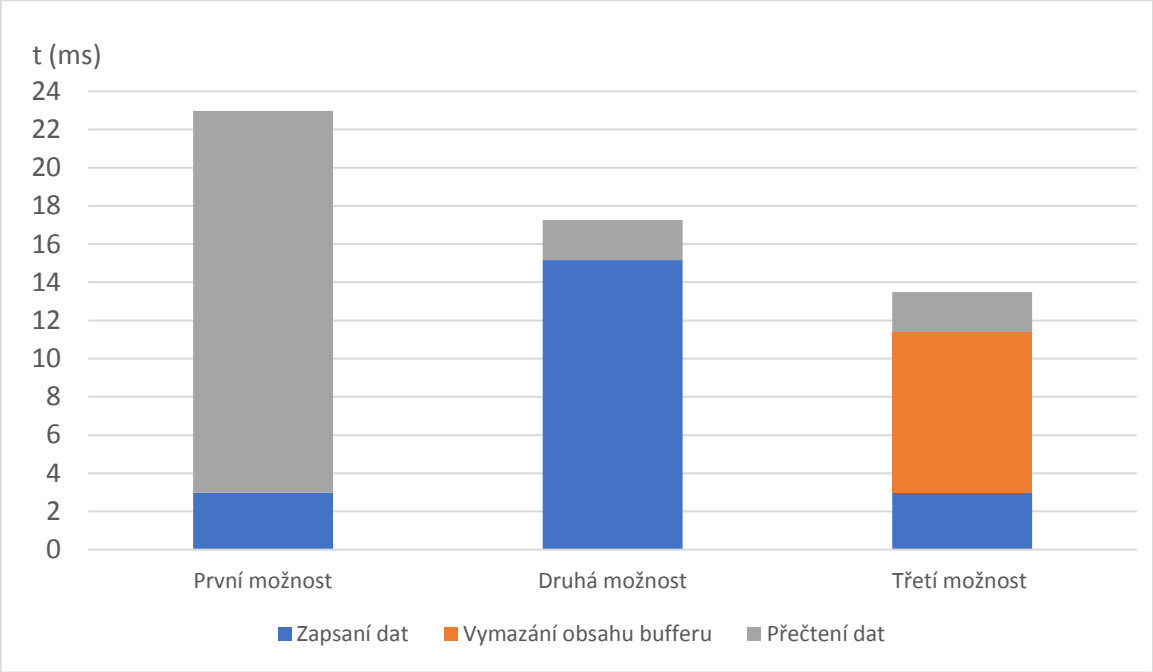
Rád bych ještě upřesnil, proč je tedy nutné vždy vymazávat obsah bufferu. Důvodem je zamezení úplnému naplnění bufferu. Víme, že při vykonávání stavového stroje se zapisují data do driveru. Po každé do driveru zapsané proměnné driver automaticky odpoví, že se zapsání podařilo, a to sice potvrzovacím desetibytovým paketem. Pokud se tedy obsah bufferu po vykonání stavového stroje nevymaže, buffer se časem celý naplní. To působí vážné problémy, protože po vykonání stavového stroje je nutné data z driveru naopak číst. Buffer pro příchozí data má 64 bytů. V následujícím příkladu popíšu, co by se při jeho naplnění a následném čtení dat stalo.

1. Ve stavovém stroji se zapsalo sedm proměnných do driveru.
2. Do bufferu přišlo sedm potvrzovacích paketů, každý pro jednu proměnnou. Potvrzovací paket má deset bytů, celkově je to tedy 70. Buffer má 64 bytů, takže je nyní naplněný a zbylých 6 bytů se ztratilo.
3. Nyní čteme data z driveru. Při čtení jsme nejprve odeslali paket se žádostí o daná data. Jelikož je buffer plný, došlo ke ztrátě žádaných dat z driveru.
4. Dále se ve funkci pro čtení dat z driveru začnou data načítat po deseti bytech z bufferu. Buffer je ale plný potvrzovacích paketů a funkce „correct_packet“ z kapitoly 4.3 všechna tato data postupně vymaže.
5. Nakonec funkce žádaná data v bufferu nenajde a data se z driveru nepřečetla.

Tento postup by se samozřejmě dal ošetřit, ale jak je z celého příkladu vidět, už takto je postupné načítání jednotlivých redundantních dat z bufferu časově náročné a zpomaluje program. Druhou možností zamezení naplnění bufferu by bylo vždy po zapsání dat do driveru čekání na potvrzovací paket a jeho vymazání. To je ale opět časově náročné a nyní uvedu konkrétní časy, které jsem při tvoření programu naměřil funkcí Arduina pro měření času. Na obrázku Obr. 4.3 jsou časy, které jsem pro tyto tři možnosti naměřil.

- První možnost – Číst data z driveru bez vymazání obsahu bufferu.
- Druhá možnost – Po odesílání dat do driveru jednotlivě na potvrzovací pakety čekat a následně je mazat ještě před čtením dat z driveru.
- Třetí možnost – Po současném zapsání vícero dat do driveru vymazat celý obsah bufferu a až poté číst jednotlivá data.

Ještě pro úplnost uvádím, že časy byly naměřeny po zapsání 7 proměnných do driveru a následném přečtení 1 proměnné. Měření časů bylo provedeno v mikrosekundách pomocí funkce „micros()“ a výsledné časy jsou průměrem z deseti měření.



Obr. 4.3: Čas uplynutý od zapsání 7 proměnných do driveru po následné přečtení jedné proměnné

Jak je na obrázku vidět, třetí možnost je zdaleka nejrychlejší. Celý proces od zapsaní sedmi proměnných do driveru po přečtení jedné proměnné z driveru vyšel na 13,5 ms. Naproti tomu druhá možnost byla z hlediska čekání na potvrzovací pakety velmi časově náročná. Čekání na potvrzovací pakety, započtené v sekci grafu „zapsání dat“, trvalo 15,2 ms. Celkový čas i s přečtením dat je 17,3 ms a v porovnání se třetí možností je pomalejší o 3,8 ms. Nakonec první možnost zabrala celých 22,9 ms.

Rozdíly jsou velké, zatímco třetí možnost umožňuje zmíněné operace provést 74 krát za sekundu, druhá už pouze 58 krát a první dokonce jen 44 krát.

Co se vývoje programu pro řízení servomotorů týče, v tuto chvíli společně s vytvořením všech funkcí z předchozích kapitol je program funkční.

4.7 Optimalizace rychlosti programu

V této kapitole se budu zabývat způsoby, kterými jsem běh programu pro Arduino zrychlil. Arduino musí být schopno zapisovat a číst data z driveru co nejvíce krát za sekundu. Zavedeme pojem „jeden takt“, kterým budeme dále označovat souhrn následujících úkonů, které se budou provádět v nekonečné smyčce:

- Příjem dat pro řízení servomotoru ze řídícího počítače.
- Zapsání požadovaných dat pro řízení do driveru (provedení stavového stroje).
- Vymazání obsahu bufferu.
- Čtení požadovaných dat z driveru.
- Odeslány přečtených dat zpět do řídícího počítače.

Nyní je pro účely simulátoru nezbytně nutné zajistit co nejvíce provedených taktů za sekundu. Tuto veličinu nazveme „obnovovací frekvence“.

Je nutné tedy zajistit tak vysokou obnovovací frekvenci, aby uživatel nepoznal, že se servomotory na simulátoru padákového kluzáku neřídí spojitě. Zároveň ale také není třeba zvyšovat obnovovací frekvenci na co nejvyšší hodnotu, protože simulace a VR brýle budou mít obnovovací frekvenci standartně maximálně kolem 90 Hz.

Pro stanovení minimální spodní hranice obnovovací frekvence stanovíme frekvenci 7,30 Hz. Tuto frekvenci jsem velmi hrubě stanovil na základě výsledků studie, ve které byly měřeny jednoduché vizuálně reakční doby u basketbalových hráčů. Průměrná reakční doba právě pro jednoduchou vizuální reakční dobu byla 136,9 ms. [15] Tudíž čas potřebný pro provedení jednoho taktu musí být menší než 136,9 ms, ideálně aspoň dvakrát.

Rychlost programu jsem zvyšoval různými způsoby, jako například co největším zjednodušováním funkcí při zachování čitelnosti programu, nebo také definováním konstant pro vynechání některých výpočtů. V následující podkapitole popíšu způsob, kterým se mi podařilo program zrychlit nejvíce.

4.7.1 Zvýšení rychlosti komunikace

V době optimalizování programu jsem používal rychlost komunikace mezi Arduinem a driverem 38400 Bd. Tato rychlost se ukázala být pro komunikaci velmi pomalá, jenom zapsáním deseti proměnných do driveru jsem naměřil 12ms (průměr z deseti měření). Dalším časově náročným úkonem bylo čtení dat z driveru, kde byly časy nekonzistentní, při čtení deseti proměnných se ale pohybovaly až nad 50 ms.

Víme, že Arduino dokáže s rezervou komunikovat rychlostí 115 200 Bd. Použitý převodník MAX3232 má garantovanou rychlost 120 kb za sekundu, což bylo uvedeno v kapitole 3.4. To odpovídá rychlosti 120 kBd. Nakonec Kinco nestanovuje, jakou maximální rychlost je sériová linka driveru schopna zvládnou, v softwaru „KincoServo+“ je ale maximální zmíněná komunikační rychlost rovna 115 200 Bd.

Tyto informace byly dobrou zprávou, nastavil jsem tedy v Arduinu komunikaci na 115 200 Bd a stejnou rychlost jsem nastavil i v driveru pomocí přepsání proměnné „U2BRG“. Komunikaci se mi takto ale nedařilo zprovoznit, zatímco driver s počítačem pomocí softwaru při vyšší rychlosti komunikace komunikoval správně.

Usoudil jsem, že Arduino nemá přesný zdroj hodin nebo že převodník nemá dostatečné dynamické vlastnosti. Jelikož jsem kvůli pandemické situaci neměl k dispozici osciloskop, abych změřil frekvenci výstupního signálu, tak jsem zvolil metodu pokus omyl. Napsal jsem jednoduchý program, který měl za úkol najít správnou přenosovou rychlost. Měl zapínat driver pomocí komunikace o dané rychlosti, kterou v dalším taktu zvětšil o 100 Bd. Zkoušel jsem to takhle pro frekvence od 105 000 do 120 000 Bd. Komunikaci se nakonec podařilo navázat na rychlosti 110 000 Bd. Pro jistotu jsem vyzkoušel ještě čtení několika proměnných z driveru a kontroloval jsem, jestli nastavené hodnoty čte Arduino správně a bez chyb. Tento postup byl úspěšný a vedl ke stanovení správné rychlosti komunikace.

Nyní už bylo třeba pouze vytvořit funkci pro inicializaci komunikace s driverem. Důležité je, že při spuštění se musí na Arduinu nastavit nejprve výchozí rychlost komunikace pro driver, tedy 38 400 Bd. Poté se do proměnné „U2BRG“ zapíše nová rychlost komunikace pro driver a až poté se na tuto novou rychlost může nastavit i Arduino. Kód je následující. Celá funkce „init_serial1“ je volána ve funkci „setup“, která je popsána v kapitole 4.6.1.

```
void init_serial1() {
    port = portSerial1;
    Serial1.begin(38400, SERIAL_8N1);
    set_baudrate(controller_adr, BAUDRATE, port); // Změna rychlosti U2BRG
    Serial1.begin(110000, SERIAL_8N1);
    Serial1.setTimeout(2);
}
```

4.8 Realizace nelineárního odporu kladeným řidičkami

Poslední částí tvoření programu pro řízení servomotorů bude realizace funkce, která bude mít za úkol přiblížit uvěřitelnost pocitu při stahování řidiček na maximum.

Jak bylo zmíněno v kapitole 1.2, řidičky při stahování kladou odpor, který není v průběhu jejich stahování lineární. Tento odpor se při stahování progresivně zvětšuje. Tento odpor je závislý na konstrukci konkrétního kluzáku. Z praxe, za pomoci testování piloty reálných kluzáků, bylo stanoveno, že se odpor řidiček při stahování zvětšuje přibližně se druhou mocninou. Pro tento průběh odporu řidiček je tedy nyní nutné vytvořit funkci, která bude na

simulátoru na základě jejich stažení zvětšovat či zmenšovat nastavený točivý moment servomotoru. Před vytvořením funkce je zapotřebí si stanovit proměnné, se kterými bude funkce pracovat:

- Pro zjišťování aktuální pozice je nutné číst polohu hřídele servomotoru.
- Je nutné stanovit velikost točivého momentu ve stavu s nulovým stažením řidiček.
- Dále je nutné stanovit velikost točivého momentu při maximálním stažení řidiček.
- Musíme zjistit, na jaké poloze se horní mez stažení řidiček nachází (dolní je 0).

Nyní je na čase vytvořit funkci a stanovit vzorec pro přepočet momentu. Úvaha je taková, že před přepočtem momentu je nutné přečíst požadovaný točivý moment z řídícího počítače a přečíst aktuální polohu X_a z driveru. Horní mez řidiček a maximální přírůstek momentu budou dány konstantami.

Pro příklad řekněme, že maximálnímu stažení řidiček bude odpovídat poloha 100 000 inc (odpovídá deseti otáčkám servomotoru). Dále řekněme, zatímco povel pro točivý moment z řídícího počítače bude točivý moment základní M_z 10 % jmenovitého, maximální hodnota momentu při stažení řidiček bude 30 %. Maximální přírůstek M_p ze základního točivého momentu M_z je tedy 20 %. Výpočet bude následovný:

$$M = M_z + M_p \ (\%) \tag{4.6}$$

Moment přírůstku dále rozepíšeme tak, aby se celkový točivý moment měnil s druhou mocninou.

$$M = M_z + \left(M_p' \cdot \frac{X_a}{X_{max}}\right)^2 \ (\%) \tag{4.7}$$

Z této rovnice jasné, že podíl poloh může nabýt hodnoty od 0 do 1. Se zvětšujícím stažením řidiček se aktuální poloha (čitatel) zvětšuje, a přírůstek momentu se začíná zvětšovat. Důležitým prvkem přepočtu je M_p', který znázorňuje přepočtenou hodnotu přírustku pro počítání s druhou mocninou. Chceme totiž, aby celý člen po mocnině na druhou dosáhl maximálně hodnoty 20 %. M_p' získáme operací inverzní, tedy druhou odmocninou. M_p' se tedy rovná přibližně 4,47.

$$M = M_z + \left(4{,}47 \cdot \frac{X_a}{100\,000}\right)^2 \ (\%) \tag{4.8}$$

Teď už stačí rovnici pro přepočet přepsat do funkce pro Arduino.

```
float prirustek = 0;
prirustek = ((float) position / (float) MAX_POS)
          * (float) MAX_TORQUE_PRIRUSTEK;
prirustek *= prirustek;
torque_recalculated = torque_zakl + (int16_t) prirustek;
target_torque(controller_adr, torque_recalculated);
```

Nejprve je spočtena vnitřní hodnota závorky z rovnice (4.8), poté je realizována druhá mocnina, následovně je přírůstek připočten k základní hodnotě točivého momentu z rovnice (4.6) a přepočet je hotový. Nakonec se po přepočtu zapíše výsledný točivý moment do driveru a servomotor má nyní nový nastavený točivý moment, velký přesně podle míry stažení řidiček. Celá funkce pro přepočet točivého momentu je uvedena v příloze D.

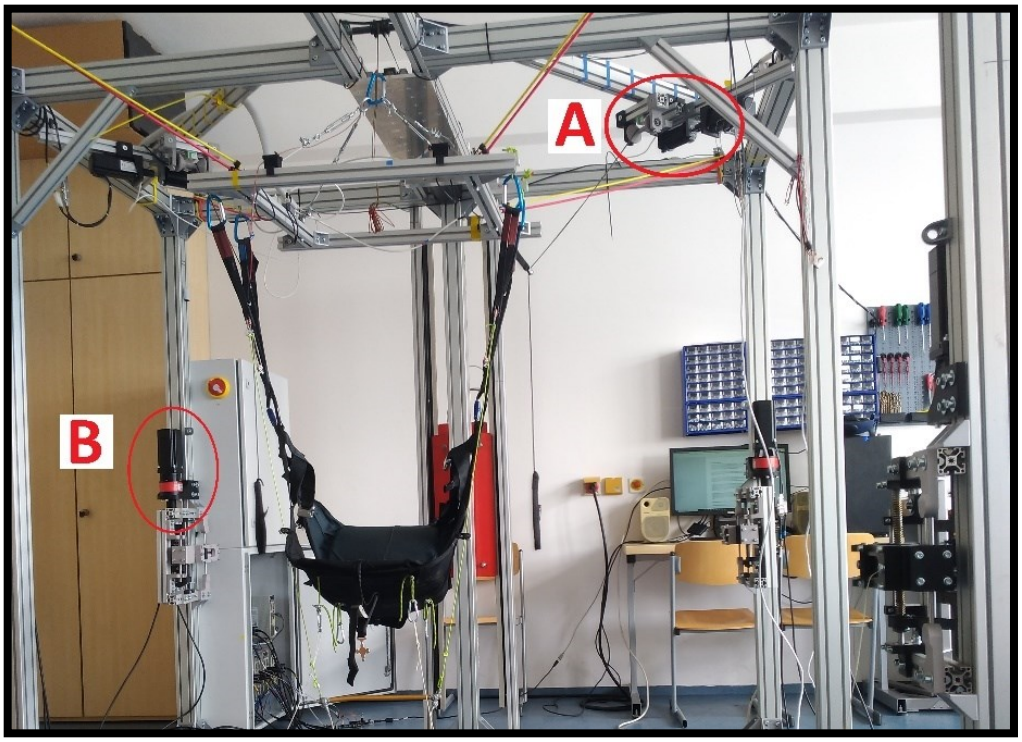
5 Implementace řízení na simulátoru a výsledky

Předmětem této kapitoly bude zhodnocení výsledků praktických zkoušek na simulátoru a prezentace výsledků řízení, tedy jak dobře celý systém řízení servomotorů přes mikrokontroler funguje. Kapitola bude zaměřená zejména na ovládání servomotorů řidiček. Pro polohovací servomotory program funguje stejně a principy jsou identické. Důraz při zkouškách byl kladen zejména na pružnost ovládání servomotorů a uvěřitelnost pocitu řidiček ovládaných servomotory v porovnání oproti skutečným řidičkám.

V následujících kapitolách bude stručně popsána funkce a konstrukce systému pro účely prezentace výsledků. Následně budou představeny výsledné hodnoty veličin pro řízení řidiček. Dále budou zhodnoceny výsledky řízení servomotorů a realističnost pocitu při stahování řidiček, budou ve stručnosti ukázány výsledky řízení polohovacích servomotorů, a nakonec budou představena možná rozšíření do budoucna.

5.1 Funkce a konstrukce systému

Ještě před prezentací výsledků definujeme, jak celý systém funguje. Z hlediska řízení se servomotory ovládají z řídícího počítače pomocí grafického rozhraní v softwaru „Unity“ viz. Obr. 5.3. V tomto rozhraní se dají nastavit režimy servomotorů a data pro ovládání. Dále se v uživatelském rozhraní zobrazují základní údaje o stavu servomotoru, jako například jeho aktuální poloha, rychlost a odebíraný proud.



Obr. 5.1: Konstrukce simulátoru; nad sedačkou jsou vidět servomotory řidiček A; v rozích konstrukce se nacházejí servomotory pro polohování pilota B

Z hlediska konstrukce jsou servomotory řidiček zavěšeny nad pilotem a jsou pevně uchyceny ke konstrukci simulátoru. Pilot je posazen do sedačky, která je na konstrukci zavěšena, a stahuje řidičky směrem dolů. Řidičky jsou navíjeny na rotor servomotoru. Polohovací servomotory přitahují pilota navíjením lan připevněných k sedačce.

5.2 Výsledné hodnoty veličin

Obsahem této kapitoly bude prezentace výsledných hodnot veličin, které budou nastavovány při řízení servomotorů řidiček a pro režim dojetí na výchozí polohu. Výsledky byly odečítány při zkouškách ze softwaru „KincoServo+“.

Tab. 5: Hodnoty veličin pro řízení servomotorů řidiček při průměru rotoru navíjecí cívky 5,9 cm

Zákl. toč. moment	Přírůstek momentu	Max. proud	Max. rychlost	Max. poloha
M_z	M_p	I_{maxcmd}	v_{max}	X_{max}
0,14 Nm; 11 % z M_N	0,11 Nm; 9 % z M_N	0,95 A	1500 ot/min	37 000 inc

Tab. 6: Hodnoty veličin pro režim dojetí na výchozí polohu po spuštění simulátoru

Použitá metoda	Proud pro dojetí	Vychýlení (offset)	Rychlost
-17	400 mA	5000 inc	150 ot/min

Jak již bylo zmíněno, řidičky na simulátoru budou řízeny v režimu točivého momentu, který bude nastavován v řídícím počítači. Prvním úkolem bylo zjistit, jaké parametry bude servomotorům nutné nastavit pro věrnou reprezentaci reálné situace. Zjišťování parametrů probíhá tak, že jedna osoba sedí u počítače, ovládá servomotory a odečítá hodnoty ze softwaru. Druhá osoba sedí na pozici pilota a podle pokynů osoby u počítače stahuje řidičky a hodnotí, jestli je dané nastavení vhodné. Je nutné, aby osoba hodnotící věrohodnost nastavení měla předchozí zkušenosti s létáním na skutečném padákovém kluzáku.

Druhý odečet veličin týkající se Tab. 6 byl proveden pouze jednou osobou u počítače, která opět přes software nastavovala parametry pro režim dojetí na výchozí polohu po spuštění simulátoru. Byly zkoušeny některé z potenciálně vhodných metod (-17, -18), které spočívají v hledání mechanického limitu. Na pozici limitu nastavují výchozí polohu. Metody hledání se liší jen ve směru otáčení při hledání výchozí polohy. Vhodná metoda se volila podle vhodného směru navíjení lana řidiček na navíjející cívku, tedy zespoda.

Výsledné hodnoty pro řízení servomotorů řidiček

V Tab. 5 jsou vidět výsledné velikosti veličin klíčové pro řízení servomotorů řidiček. Řidičky jsou řízený v režimu točivého momentu, jsou zde tedy klíčové hlavně mezní momenty, které kladou odpor pilotovi při jejich stažení. Je nutno dodat, že M_z je pouze základní nejnižší hodnota, která se může dynamicky podle požadavků z řídícího počítače měnit. Točivý moment M_p je zase maximální přírůstek točivého momentu pro daný M_z. Ten je důležitý z hlediska realizace nelineárního odporu, které řidičky kladou při svém stažení. Detailní popsání realizace nelineárního točivého momentu řidiček se nachází v kapitole 4.8.

Maximální proud I_{maxcmd} je proud, který omezuje maximální moment, kterého je servomotor schopen dosáhnout. Hodnota 0,95 A odpovídá zhruba 30 % z momentu jmenovitého M_N. Nad tuto hodnotu momentu nelze servomotor nastavit. To je z důvodu bezpečnosti a zajištění, že silové účinky servomotoru nebudou moct být natolik velké, že by nějakým způsobem poškodily simulátor nebo zranily pilota.

Velikost max. rychlosti určuje rychlost, které servomotor v režimu točivého momentu dosáhne bez zátěžného momentu. Tato rychlost je důležitá pro situaci, ve které má pilot staženou řidičku a rychle ji pustí nahoru. Je nutné, aby servomotor ihned navinul lano řidiček. Dostačující rychlostí se ukázala být hodnota okolo 1500 *ot/min*.

Max. poloha řidiček je dána samotnou délkou lana řidiček a poloměrem cívky lano navíjející. Při průměru navíjející cívky 5,9 cm navíjel servomotor 18,5 cm lana jednou otáčkou, otáček pro danou délku lana po úplném stažení řidiček udělal 3,7. Maximální poloha je tedy rovna 37 000 inc.

Výsledné hodnoty pro režim dojetí na výchozí polohu

Výsledky z Tab. 6 jsou důležité pro spouštění simulátoru. Po spuštění je poloha řidiček neznámá, proto je nutné nechat motory dojet na výchozí polohu. Na to je v driveru připraven režim „Homing_mode“, jehož parametry lze jednoduše nakonfigurovat.

Nejdůležitějším parametrem je použitá metoda „-17“, která začne se servomotorem točit na jednu stranu, dokud nenarazí na mechanický limit. V tomto limitě podle velikosti proudu pro dojetí „ztratí“ točivý moment, motor zde zastaví a nastaví zde výchozí polohu mínus velikost vychýlení. Vychýlení je nastaveno proto, že na pozici mechanického limitu na simulátoru jsou řidičky vysoko. Nulová poloha se tedy díky vychýlení nachází na mírně nižší poloze, než kde se nachází mechanický limit (v tomto případě o 5000 inc neboli půl otáčky servomotoru). Rychlost udává rychlost otáčení servomotoru při hledání výchozí polohy.

5.3 Výsledky řízení

V této kapitole budou uvedeny poznatky získané z praktických zkoušek na simulátoru. Tyto zkoušky už byly zaměřeny pouze na ovládání servomotorů z řídícího počítače skrz mikrokontroler.

Jako první budou uvedeny výsledky řízení řidiček v režimu točivého momentu. Za druhé budou uvedeny výsledky řízení servomotorů v režimu dojetí na výchozí polohu, který bude využíván po zapnutí simulátoru. Nakonec budou velmi stručně uvedeny výsledky řízení polohovacích servomotorů.

5.3.1 Výsledky řízení řidiček

Po zjištění výsledných hodnot veličin z kapitoly 5.2 nastala chvíle otestovat řízení servomotorů řidiček pomocí mikrokontroleru. Prvním úkolem bylo vyzkoušet komunikaci mezi Arduinem a drivery řidiček. Zkouška proběhla úspěšně, drivery se dařilo vypínat a zapínat.

V režimu točivého momentu se ale servomotor neroztočil. Na vině byla chyba v programu, které jsem se dopustil při definování datových typů. Předpokladem pro funkční ovládání servomotoru mikrokontrolerem je, že datové typy daných proměnných v programu jsou shodné s datovými typy v driveru. Problém jsem vyřešil kontrolou všech proměnných v programu.

Dalším úkolem bylo zajistit, aby oba servomotory navíjely lana řidiček ze spodní strany navíjecí cívky rotoru. Jelikož se nyní oba servomotory točily stejným směrem, jeden navíjel lano na spodní stranu navíjecí cívky a druhý na horní. Řešením bylo změnit směr otáčení jednoho servomotoru invertováním směru otáčení proměnnou „Invert_dir“.

Posledním úkolem byla kontrola funkce pro nelineární odpor kladený řidičkami. Funkce zpočátku nefungovala. Nefunkčnost byla způsobena špatným umístěním přepočtu odporu řidiček v programu. Přepočet byl umístění ve stavovém stroji, ale tam se provedl jen jednou při povelu z řídícího počítače a v dalších taktech smyčky „loop“ už se odpor nepřepočítával. Stavový stroj je totiž volán jen v taktu, ve kterém přišel nový povel z počítače, v dalších taktech už nikoliv. Celá funkce pro přepočet tedy musí být volána v každém taktu nekonečné smyčky „loop“, ať už přišel povel z řídícího počítače nebo ne.

```
void loop() {
    newData = false;
    newData = read_data_from_pc();

    if (newData) {
        event_state_machine_ridicky(controller_adr);
        clear_buffer();
    }

    if ((loops % 100) == 0) {     // Provedení přepočtu jednou za 100 ms
          read_data_from_drivers();
          send_data_to_pc();
          recalc_torque_demand(); // Přepočet odporu řidiček
    }

    delay(1);
    loops++;
}
```

Novou smyčku „loop“ můžeme porovnat s původní smyčkou z kapitoly 4.6.2. Zde se nyní funkce přepočtu volá přibližně každých 100 ms. Tím je tedy zajištěno, že se přibližně desetkrát za sekundu na řidičkách nastaví nový točivý moment odpovídající aktuálnímu stažení řidiček v daném okamžiku. Před přepočtem je samozřejmě nutné přečíst aktuální pozici řidiček funkcí „read_data_from_drivers“. Funkce „send_data_to_pc“ pro odesílání aktuálních dat do počítače se nyní též volá každých 100 ms, tím se zajistí neustálá aktualizace čtených dat z driveru v počítači pro lepší přehled o průběhu ovládání servomotorů. To se ukázalo být velmi užitečné z hlediska testování řízení.

Výsledkem všech zmíněných úprav v programu je progresivní řízení řidiček, které reaguje na jejich stažení pilotem a úměrně tomu zvyšuje jejich točivý moment (a tím odpor) s druhou mocninou. Na změnu základního točivého momentu M_z řídícím počítačem reaguje mikrokontroler správně a maximální odpor řidiček při plném stažení zase posune o maximální přírůstek momentu M_p dál. Dynamika je tedy vyhovující. Pokud je vyžadováno více přepočtů odporu řidiček za jednu sekundu pro větší plynulost změn točivého momentu, je možné upravit číslo „loops % 100“ v podmínce „if“ na menší hodnotu.

Pro účely názorného porovnání odporů kladených řidičkami jsem pomocí siloměru změřil síly přitahující řidičky na počátku stahování a na konci. Jak je z Obr. 5.2 vidět, při plném stažení řidiček servomotor kladl vyšší odpor než na začátku stahování. Síly na fotografiích jsou v jednotkách *N*. Síly byly změřeny pouze pro demonstraci rozdílu v nastavených momentech servomotorů a jsou spíše indikační.

Obr. 5.2: Porovnání odporu řidiček při jejich různém stažení; velké fotky pouze ilustrují pozici řidičky, síly na nich nesouhlasí

5.3.2 Výsledky režimu dojetí na výchozí polohu

Způsob dojetí servomotorů řidiček na výchozí polohu byl popsán v kapitole 5.2. Nyní bylo nutné režim odzkoušet a zajistit jeho správnou funkci. Následující funkce vychází z hodnot uvedených v Tab. 6. Tato funkce je součástí stavového stroje.

```
case startHoming:
    homing_method(controller_adr, MECH_NEGATIVE, 1, portSerial1);
    homing_current(controller_adr, HOMING_I_MA, portSerial1);
    set_homing_offset(adr, OFFSET1, portSerial1);
    start_homing(adr, portSerial1);
    break;
```

Nejprve je nutné nastavit jednu z metod dojetí na výchozí polohu, a to funkcí „homing_method“. V tomto případě je nastavena zvolená metoda „-17“, která roztočí servomotor daným směrem a nastaví výchozí polohu na poloze svého mechanického limitu. Tento způsob je reprezentován konstantou „MECH_NEGATIVE“. Jednička poté říká, že se má jednat o dojetí na pozici s vychýlením, tedy že výchozí poloha bude poloha limitu mínus velikost vychýlení (OFFSET1).

Další částí funkce je nastavení maximálního proudu odebíraného servomotorem při provádění tohoto úkonu, a to na změřených 400 mA. Proud se nastavuje z důvodu omezení maximálního momentu servomotoru, jelikož moment je úměrný proudu. Podle toho ostatně driver pozná, že je servomotor na svém mechanickém limitu.

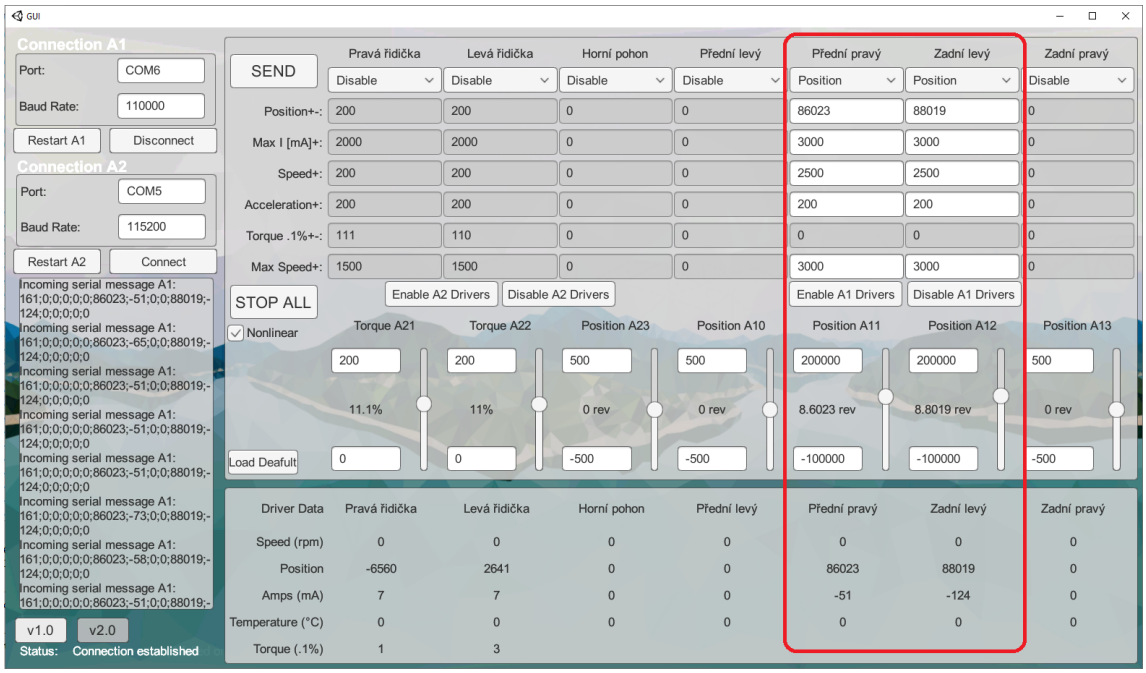
Nakonec se danou funkcí nastaví velikost vychýlení pro případ dojetí na výchozí polohu s vychýlením a poslední funkcí se celý proces spustí. Spuštění procesu funkcí „start_homing“ se provádí ovládáním softwarových digitálních vstupů viz. kapitola 3.6.

Po spuštění tohoto režimu z řídícího počítače režim fungoval tak jak bylo zamýšleno. Po správném nastavení hodnot proměnných z Tab. 6 stačí režim dojetí na výchozí polohu spustit a servomotor už sám na výchozí polohu najede.

5.3.3 Výsledky řízení polohovacích servomotorů

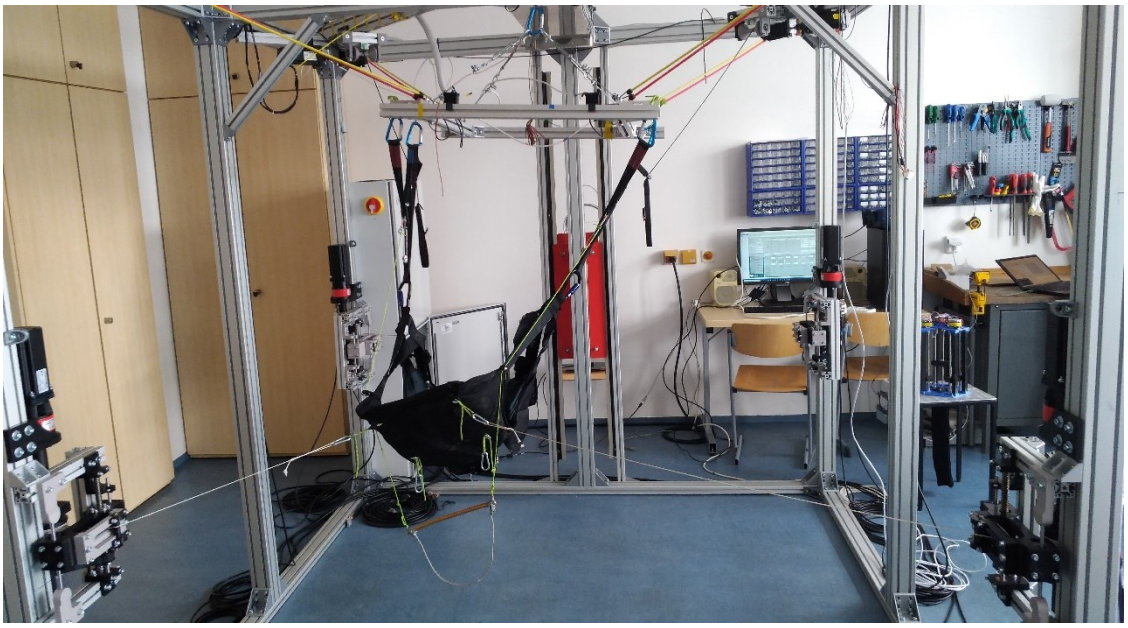
I když tato práce byla zaměřena na servomotory řidiček, povedlo se zprovoznit i ostatní servomotory. Základy programu jsou pro řidičky a polohovací servomotory naprosto identické. Drobnými úpravami ve stavovém stroji a rozšířením programu o další dva servomotory bylo možné velmi rychle přivést i polohovací servomotory k životu.

Polohovací servomotory budou na simulátoru fungovat v režimu dojetí na cílovou polohu, jejich funkci jsem tedy ověřil zadáváním různých cílových poloh z řídícího počítače pomocí grafického rozhraní Unity viz. Obr. 5.3. Tvorba grafického rozhraní byla součástí jiné práce viz. [16].

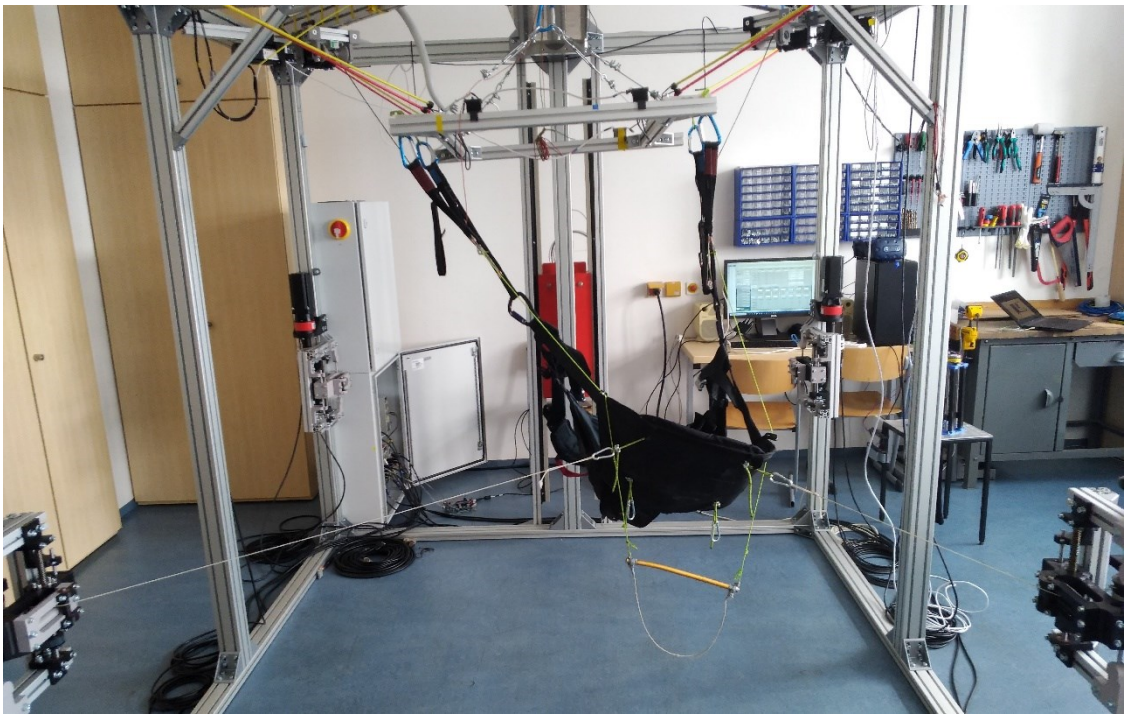


Obr. 5.3: Grafické rozhraní Unity řídícího počítače pro ovládání servomotorů; ovládané servomotory označeny červeně; popisky ovládaných motorů nesouhlasí [16]

Důležité bylo přibližné stanovení velikostí základních veličin pro řízení, které bylo provedeno analogicky k řidičkám z kapitoly 5.2. Po nastavení základních hodnot, které jsou vidět na Obr. 5.3, už stačilo měnit polohu sedačky pomocí posuvníků ve střední části vyznačené plochy. Byl jsem tak schopen měnit polohu sedačky dvěma servomotory v přední části konstrukce. Zadní servomotory nebyly ovládány z důvodu chybějících šňůr pro přitahování sedačky. Při daných parametrech a při rychlosti 2500 *ot/min* byla sedačka schopna měnit svoji pozici velmi dynamicky.



Obr. 5.4: Polohování sedačky doprava



Obr. 5.5: Polohování sedačky doleva

5.4 Zvýšení komunikační rychlosti

V této kapitole představím krátký náhled na zvýšení rychlosti komunikace mezi mikrokontrolerem a driverem, jaké přínosy bude rychlejší komunikace mít a jak jsem se ji pokusil zvýšit.

Zvýšení komunikační rychlosti mezi mikrokontrolerem a driverem přináší své výhody, ale i některé nevýhody. Hlavní výhodou je menší čas potřebný k vykonání jednoho taktu (jedné smyčky „loop“). Kratší doba taktu umožňuje vícekrát za sekundu:

- Přijímat nová data pro řízení z řídícího počítače.
- Přepočítávat točivý moment pro řidičky.
- Číst a zapisovat kterákoliv data z driveru.

Zvýšení komunikační rychlosti by tedy mělo přispět k větší plynulosti chodu celého simulátoru, ale také například více času pro případné výpočty, které se v budoucnu budou na mikrokontroleru realizovat.

Vyšší komunikační rychlosti s sebou přinášejí bohužel také vyšší požadavky na hardware a po překročení určité rychlosti způsobují chyby v přenosu. Nyní nastíním, jakým způsobem jsem se pokusil zvýšit komunikační rychlost a proč je to v dané hardwarové konfiguraci problematické. Odkazuji se tímto také na kapitolu 3.4, ve které je zmíněn použitý hardware.

Cílem bylo zvýšit komunikaci ze 115 200 Bd na 230 400 Bd. Tato rychlost je pro čip ATmega 2560 vyhovující, i když při této rychlosti dochází k mírné 3 % chybě při přenosu [11]. Další částí hardwaru je převodník TTL na RS-232 MAX3232, který má garantovanou rychlost komunikace pouze 120 000 Bd. Výrobce uvádí, že je převodník teoreticky schopen rychlosti až 1 MBd [12].

Velkou neznámou je ve zvyšování rychlosti ale driver, v jehož dokumentaci není ani zmínka o nejvyšší rychlosti pro sériovou komunikaci. V dokumentaci je uvedena pouze rychlost 115 200 Bd, o vyšší rychlosti se nezmiňuje [10].

Byl tedy čas nastavit na driveru požadovanou rychlosti. Přenosová rychlost v Arduinu se nastavuje velmi jednoduše příkazem „Serial.begin(230400);“. U driveru se nastavuje pomocí proměnné „U2BRG“, detailní postup je popsán v kapitole 4.7.1. Důležité je, že hodnota proměnné pro požadovanou rychlost je neznámá. Rychlosti 115 200 Bd odpovídá „U2BRG“ rovné 90 [10]. Rychlosti 38 400 Bd, což je rychlost 3x menší, odpovídá zase „U2BRG“ rovné 270 [10]. Vyskytuje se zde tedy lineární závislost.

Pro zjištění velikosti proměnné „U2BRG“ pro rychlost 230 400 Bd jsem tedy vydělil tuto rychlost 115 200 Bd, abych zjistil, jakým násobkem chci komunikaci zrychlit. Tímto násobkem jsem dále vydělil „U2BRG“ rovno 90, výsledkem je velikost „U2BRG“ pro požadovanou vyšší rychlost. Pro dvojnásobek rychlosti je výsledek jasný, ovšem pro jiné rychlosti, jako je 250 000 Bd může být výpočet užitečný. Rychlost 250 000 Bd by v budoucnu mohla být lepší alternativou, jelikož se při ní čip ATmega 2560 nedopouští žádné chyby [11].

$$U2BRG_{230,4k} = U2BRG_{115,2k} \div \frac{230\,400}{115\,200} = 90 \div 2 = 45 \tag{5.1}$$

Hodnotu 45 proměnné „U2BRG jsem tedy zapsal do driveru a spustil test, ve kterém se měl driver zapínat a vypínat. Stejně jako při zvyšování rychlosti v kapitole 4.7.1, při nastavení rychlosti na straně mikrokontroleru na přesnou hodnotu 230 400 Bd komunikace nefungovala. Napsal jsem tedy jednoduchý program pro automatické testování komunikačních rychlostí od 200 do 260 kBd.

Výsledná hodnota, při které komunikace funguje je 215 000 Bd. Opět musí být rychlost komunikace na straně mikrokontroleru nižší než na straně driveru, stejně jako v kap. 4.7.1, aby komunikace fungovala. Musím také dodat, že spolehlivost komunikace je při této rychlosti velmi malá. Driver na povely pro zapnutí zareaguje jen zřídka. Při použití současného hardwaru na simulátoru je tedy komunikace při této rychlosti naprosto nevyhovující.

Na vině se zdá být převodník MAX3232, který má garantovanou rychlost pouhých 120 000 Bd. Pro verifikaci této teorie jsem připojil osciloskop k jeho výstupům. Už při nastavené rychlosti 110 000 Bd byl jeho výstup téměř o 5000 Bd vyšší. Zároveň jsem na výstupu Arduina ve stejné chvíli naměřil téměř přesných 110 000 Bd, Arduino tedy nastavuje rychlost komunikace správně.

Pro potenciální zrychlení komunikace bude do budoucna potřeba minimálně volba kvalitnějšího převodníku, MAX3232 způsobuje při vyšších rychlostech nezanedbatelné chyby. Je také faktem, že použitý převodník je jednou z nejlevnějších možností na trhu a dražší alternativy by mohly být schopny o poznání vyšších komunikačních rychlostí. Další otázkou bude, jestli vyšší rychlosti nepůsobí problémy i driveru. Ovšem i při rychlosti 115 200 Bd je komunikace z hlediska obnovovací frekvence viz. 4.7 dostatečně rychlá a optimalizací programu se dá vyřešit většina problémů s rychlostí.

5.5 Možná budoucí rozšíření programu

Program pro řízení servomotorů byl navržen modulárně, a proto bude vždy prostor pro jeho rozšíření. Tato kapitola obsahuje možná rozšíření programu do budoucna. V této práci bylo pojednáváno zejména o programu pro řízení řidiček simulátoru a pro ostatní servomotory simulátoru platí stejné základní principy řízení. Kromě řidiček se na simulátoru nacházejí další čtyři výkonnější polohovací servomotory a velký 3f servomotor se nachází nad pilotem a je schopen s ním pohybovat nahoru a dolu, viz. Obr. 5.1.

Částečné zprovoznění polohovacích servomotorů bylo provedeno už v kapitole 5.3.3. K jejich úplnému zprovoznění bude zapotřebí pouze změření hodnot analogicky ke kapitole 5.2 a následná zkouška pro ověření funkčnosti. Program byl navržen modulárně takovým způsobem, aby bylo možné přidat další ovládaný servomotor jednoduše připojením na nový sériový port. Poté stačí už jen rozšířit přijímanou datovou strukturu z řídícího počítače a rozšířit stavový stroj o nový servomotor s příslušnými parametry. Modulárností programu je dosaženo optimálního řešení pro budoucí rozšíření. Výsledky jednoduchosti rozšíření o další servomotory byly demonstrovány v kapitole 5.3.3.

Další rozšíření přijde na řadu, až bude software simulace schopen zpracovávat data z mikrokontroleru a v reálném čase dávat povely servomotorům. Bude to úprava přepočtu točivého momentu na základě stažení řidiček z kapitoly 0. Protože bude pilot polohován do různých poloh, bude se měnit i stažení řidiček. Předpokládejme, že pilot bude předními servomotory přitažen směrem dopředu o 50 cm. Tím se přirozeně stáhnou řidičky, které jsou přimontovány pevně ke konstrukci simulátoru a nehýbou se společně s pilotem. I přes stažení řidiček vlivem natočení pilota je ale nutné zajistit, aby se jejich toč. moment kvůli tomuto stažení nezvýšil. Bylo by tedy možné realizovat výpočet aktuální pozice pilota v řídícím počítači, který by mikrokontroleru řídícím řidičky předal informaci o jejich stažení vlivem natočení pilota. Říkejme tomuto stažení posun řidiček X_{pos}. Mikrokontroler by tedy dostal přesnou hodnotu posunu v jednotkách inc, kterou by nyní pokládal za výchozí polohu daného servomotoru. Přepočet točivého momentu z rovnice (4.7) by se tedy změnil na:

$$M = M_z + \left(M'_p \cdot \frac{X_a - X_{pos}}{X_{max}}\right)^2 \ (\%) \tag{5.2}$$

Smysl proměnných zůstává stejný jako v původní rovnici (4.7) z kapitoly 0. Přepočet se bude realizovat tedy jen tehdy, když pilot i v nakloněné pozici řidičku stáhne. V takovém případě už bude hodnota čitatele ve zlomku nenulová a zvýší se řidičky odpor.

V neposlední řadě by se dal program zrychlit, ať už optimalizací samotného kódu, nebo zrychlením komunikace jak mezi mikrokontrolerem a driverem, tak mezi mikrokontrolerem a řídícím počítačem. Zrychlení může být v budoucnu například tak jednoduché, jako je přidání konstant u známých naměřených hodnot. Pokud bude například jasné, že maximální proud pro servomotory řidiček bude roven velikosti 1 A, nebude už nutné tuto informaci přijímat z řídícího počítače. Pokud nebude nezbytně nutné znát aktuální teplotu servomotoru, proč ji z driveru za běhu programu číst. Podobnými úpravami bude možné program zrychlit.

Další dodatečná rozšíření už v současném stavu programu budou velmi jednoduchá, neboť program obsahuje všechny funkce, které jsou pro ovládání servomotoru zapotřebí.

Závěr

Cílem této práce bylo vytvoření programu na pokročilé řízení servomotorů Kinco pro simulátor padákového kluzáku pomocí mikrokontroleru Arduino. Tento úkol byl nezbytnou částí pro vytvoření realistického pohybu pilota v prostoru, jež má simulovat skutečné síly působící na pilota při letu padákovým kluzákem. Mikrokontroler je jakýmsi mezistupněm mezi řídícím počítačem a servomotory, jeho úlohou je tedy řízení servomotoru na základě řídících požadavků z počítače. Práce byla zaměřena zejména na servomotory řidiček, ale navržený program je funkční i pro všechny ostatní používané servomotory.

Prvním úkolem bylo seznámit se s používaným hardwarem a jakým způsobem se dané servomotory ovládají. Poté byl definován způsob komunikace s drivery pomocí RS-232 a byl popsán komunikační protokol driverů pro tento způsob komunikace. Následujícím logickým krokem, když už bylo známo, jak se servomotory ovládají, bylo samotné vytvoření programu, tak aby vyhovoval z hlediska rychlosti a rozšiřitelnosti. Při psaní programu byl kladen důraz také na správnost všech požadovaných funkcí. Takovými funkcemi jsou například zápis dat do driveru pro ovládání servomotoru, vyčítání dat o běhu servomotoru z driveru a možnost ovládání servomotorů z řídícího počítače.

Po vytvoření základních funkcí byla na řadě definice stavového stroje, ve kterém byly vytvořeny požadované režimy řízení servomotorů. Mezi tyto režimy patří například režim točivého momentu, dojetí na cílovou polohu nebo dojetí na výchozí polohu. Výsledkem byly funkce, které dané servomotory dokážou ovládat ve všech simulátorem požadovaných režimech. Posledním hlavním krokem při tvoření základního programu bylo použití definovaných funkcí v inicializační a běhové funkci mikrokontroleru. Přitom byl kladen důraz hlavně na správné pořadí, ve kterém se funkce při běhu programu volají. Funkce navíc, která už neměla na fundamentální funkčnost ovládání vliv, byla funkce pro realizaci nelineárního odporu řidiček. Funkce byla přidána za účelem vytvoření realističtějšího pocitu při stahování řidiček.

S funkčním řídícím programem už zbývalo realizovat praktické zkoušky, vyčíst všechny důležité hodnoty pro řízení a posoudit správnost funkce. Po představení dané konstrukce simulátoru, na kterém byly experimenty realizovány, bylo prvním krokem vyčíst velikosti klíčových veličin pro řízení ze softwaru „KincoServo+“. To bylo důležité, protože konkrétní vlastnosti a rozměry simulátoru určují, jak velkými silami budou muset řidičky proti pilotovi působit. Po zjištění všech potřebných veličin byly v programu definovány výchozí velikosti

proměnných pro řízení servomotorů. Pro proměnné, které už budou po celou dobu běhu simulátoru neměnné, byly definovány příslušné konstanty.

Samotné výsledky řízení byly velmi uspokojivé. Opravením několika drobných chyb a přidáním přepočtu točivého momentu řidiček do hlavní smyčky programu byl vytvořen solidní základ řízení řidiček. Program je nyní schopen progresivně zvyšovat točivý moment řidiček na základě zpětné vazby z enkodéru. Řidičky jsou také schopny po spuštění dojet na svoji výchozí polohu. Všechny další stavy stavového stroje, jako například resetování chybových hlášení driveru a režim dojetí na cílovou polohu taktéž správně fungují.

Zadáním také bylo experimentálně ověřit řízení se zpětnou vazbou nejenom na základě dat z enkodéru, ale také ze senzorů. Momentálně je řízení navrženo tak, že data ze senzorů se budou zpracovávat v řídícím počítači viz. Obr. 4.1. Mikrokontroler bude přijímat povely na řízení servomotorů už přepočítané s ohledem na jejich výstupy.

Na závěr bylo pojednáno o limitacích nyní používaného hardwaru, konkrétně převodníku TTL na RS-232, který neumožňuje spolehlivou komunikaci na vyšší rychlosti než 115 200 Bd. Byla zmíněna i potenciální budoucí rozšíření programu.

Finálně navržený program je schopen bez obtíží řídit všechny servomotory Kinco používané na simulátoru. Program je schopen naprosto přesného ovládání servomotorů řidiček. Další variace programu, která byla vytvořena pro ovládání polohovacích servomotorů, je schopna precizně polohovat pozici svého rotoru v řádu setin stupňů. Tyto dva navržené programy pro Arduino, které jsou jen velmi málo odlišné, jsou dohromady schopné dynamicky polohovat pilota do jakékoliv polohy ve stanovených mezích a poskytnout mu téměř realistický pocit odporu na řidičkách.

Seznam literatury a dalších informačních zdrojů

[1] PLOS, Richard. ***Paragliding.*** Cheb: Svět křídel, 2008. ISBN 978-80-86808-47-5

[2] Aviation Supplies & Academics. ***Glider Flying Handbook.*** Washington: FAA, 2013. ISBN 978-1619541047

[3] OBERGRUBER, Julian a MEHNEN, Lars. ***Development of a Paraglide Control System for Automatic Pitch Stabilization to Increase the Passive Safety*** [online]. Science Direct, 2016 [cit. 22.3.2021]. ISSN 1877-7058. Dostupné prostřednictvím Science Direct. DOI: https://doi.org/10.1016/j.proeng.2016.06.184

[4] ŠULC, Dalibor, 2011. ***Návrh Softwarového modulu trenažéru přistání na padákovém kluzáku.*** Brno. Diplomová práce. Vysoké učení technické v Brně. Fakulta strojního inženýrství. Ústav mechaniky těles, mechatroniky a biomechaniky.

[5] El Speedo. ***Výuka k testům test*** [online]. El Speedo s.r.o., © 2017 [cit. 23.3.2021]. Dostupné z: https://elspeedo.cz/prakticke-rady/

[6] Flybubble. ***Flight Skills*** [online]. Flybubble Paragliding, © [cit. 23.3.2021]. Dostupné z: https://flybubble.com/blog/flightskills

[7] SKALICKÝ, Jiří. ***Elektrické servopohony***. Brno: VUT FEKT, 2002. ISBN 80-214-1978-4 Dostupné z: https://www.vutbr.cz/www_base/priloha.php?dpid=33400

[8] Kinco automation. Servo systems [online]. Anaheim Automation Inc., [cit. 26.3.2021]. Dostupné z: https://www.kincoautomation.com/marketing/servo/

[9] COLLINS, Danielle. ***What is quadrature encoding?*** [online]. WTWH Media LLC, © 2021 [cit. 30.3.2021]. Dostupné z: https://www.linearmotiontips.com/what-is-quadrature-encoding/

[10] Kinco Electric（Shenzhen） Ltd. [online dokumentace]. ***Kinco FD Series Servo User Manual.*** [cit. 20.2.2021]. Dostupné z: https://kamelectro.com/pdf/Kinco/Servo/Servo_FD_User_Manual_EN.pdf

[11] Arduino. ***Arduino MEGA 2560 & Genuino MEGA 2560*** [online]. Arduino, © 2021 [cit. 8.4.2021]. Dostupné z: https://www.Arduino.cc/en/Main/ArduinoBoardMega2560/

[12] Maxim Integrated [online dokumentace]. ***MAX3232ESE Datasheet***. Maxim Integrated, © 1999 [cit. 8.4.2021]. Dostupné z: https://www.alldatasheet.com/datasheet-pdf/pdf/73150/MAXIM/MAX3232ESE.html

[13] STRANGIO, Christopher. ***The RS232 STANDARD*** [online]. CAMI Research Inc., © 2015 [cit. 8.4.2021]. Dostupné z: https://www.camiresearch.com/Data_Com_Basics/RS232_standard.html

[14] OLMR, Vít. ***HW server představuje - Sériová linka RS-232*** [online]. HW server s.r.o., © 2014 [cit. 9.4.2021]. Dostupné z: https://vyvoj.hw.cz//rozhrani/hw-server-predstavuje-seriova-linka-rs-232.html

[15] Ghuntla, T. et al. ***A Comparative Study of Visual Reaction Time in Basketball Players and Healthy Controls*** [online]. National Journal of Integrated Research in Medicine, 2016. [cit. 12.5.2021]. ISSN: 0975-9840. Dostupné prostřednictvím EBSCOhost. DOI: http://search.ebscohost.com/login.aspx?direct=true&db=asn&AN=71533626&lang=cs&site=ehost-live

[16] VORLÍČEK, Jakub 2021. ***Návrh systému řízení pro simulátor padákového kluzáku.*** Plzeň. Bakalářská práce. Západočeská univerzita v Plzni. Fakulta elektrotechnická.

[17] SIVERA, Richard. ***Control*** [online]. GitLab Repository, 2021. Dostupné z: https://gitlab.fel.zcu.cz/parasim/control

Seznam obrázků

Seznam tabulek

Přílohy

Příloha A – Funkce pro práci s digitálními vstupy/výstupy a konstanty

```
// Definice konstant pro funkci “din_simulate”. Konstanty jsou
// výsledkem kódování uvedeného v kapitole 3.6. Jsou zahrnuty
// různé funkce.
typedef uint8_t DinSimulate;
const DinSimulate EN_LIMS = 49;
const DinSimulate RST_ERROR = 2;
const DinSimulate EN_LIMS_AC = 57;
const DinSimulate EN_LIMS_SHOM = 53;
const DinSimulate EN_LIMS_RST_QUICKSTOP = 55;
const DinSimulate DISABLE = 0;
const DinSimulate EN_PLIM = 33;
const DinSimulate EN_NLIM = 17;

// Funkce pro ovládání digitálních vstupů. Bytová proměnná “bitcode”
// je zapsána do proměnné driveru “din_simulate”. Proměnná může být
// hodnoty od 0 do 255, nebo může být použita jedna z definovaných
// konstant uvedených výše.
void * din_simulate (uint8_t adr, DinSimulate bitcode, serial_port port) {
    uint8_t packet [PL] = {adr, 0x2F, 0x10, 0x20, 0x02, 0x00, 0x00,
                           0x00, 0x00, 0x00};

    packet [DATA_POS] = bitcode & 0xFF;

    packet [PL-1] = eval_chksum(packet);
    send_packet_to_port(packet, port);
}

// Funkce pro čtení digitálních výstupů. Přečte dekadickou hodnotu
// z proměnné driver “dout_status” a převede ji na binární kód,
// který uloží do bytového pole viz. kapitola 3.6.2.
void * dout_status (uint8_t adr, serial_port port) {
    uint8_t bitcode = 0;

    // Konstanta “DOUT_STAT” se rovná adrese proměnné “dout_status”
    bitcode = read_packet(adr, DOUT_STAT, port);
    for(int i = 0; i < 5; i++) {
        dataToPC.doutput[i] = 0;
    }
    int x = 0;
    while(bitcode > 0) {
        dataToPC.doutput[x] = (bitcode % 2);
        bitcode = (bitcode / 2);
        x++;
    }
}
```

Příloha B – Program pro zapínání driveru mikrokontrolerem

```
#define PL 10 // Definice délky datového paketu v bytech

uint16_t suma = 0;
uint8_t driver_ID;

uint8_t eval_chksum (byte packet[PL]) {
    suma = 0x00;

    for (int i = 0; i < (PL - 1); i++) {
        suma = (packet[i]) + suma;
    }
    // Kontrolní součet v návratové hodnotě funkce
    return ((((~suma) & ((uint16_t)(0xff))) + 0x01) & ((uint16_t)(0xff)));
}

void * enabled_on (uint8_t adr) {
uint8_t packet [PL] = {adr, 0x2B, 0x10, 0x20, 0x02, 51, 0x00, 0x00,
                       0x00, 0x00}; // Definice paketu pro zapnutí

    packet [PL-1]= eval_chksum(packet);
    Serial1.write(packet, PL); // Odeslání paketu
}

void * enabled_off (uint8_t adr) {
uint8_t packet [PL] = {adr, 0x2B, 0x10, 0x20, 0x02, 50, 0x00, 0x00,
                       0x00, 0x00}; // Definice paketu pro vypnutí

    packet [PL-1] = eval_chksum(packet);
    Serial1.write(packet, PL); // Odeslání paketu
}

void setup() {
    driver_ID = 0x01;

    // Spuštění sériové komunikace
    Serial1.begin(38400, SERIAL_8N1);
        while (!Serial1)
            Serial1.begin(38400, SERIAL_8N1);
}

void loop() {
  enabled_on(driver_ID);
  delay(2000);
  enabled_off(driver_ID);
  delay(2000);     }
```

Příloha C – Stavový stroj pro řidičky

```
set_state();

switch (state) {
    case enabled: // Zapnuto
        enabled_on(adr, portSerial1);
        din0(adr, PULSE_MODE, portSerial1);
        profile_acc(adr, ACCEL_RIDICKY, portSerial1);
        profile_dec(adr, DECCEL_RIDICKY, portSerial1);
        max_speed_rpm(adr, dataToDriver.max_rpm1, portSerial1);
        soft_nlimit(adr, N_LIMIT1, portSerial1);
        soft_plimit(adr, P_LIMIT1, portSerial1);
        max_current(adr, dataToDriver.milli_amps1, portSerial1)
        break;
    case disabled: // Vypnuto
        din0(adr, PULSE_MODE, portSerial1);
        enabled_off(adr, portSerial1);
        break;
    case setZeroPosition: // Nastavení nulové pozice
        set_homing_offset(controller_adr, 0, portSerial1);
        homing_method(controller_adr, SET_POS_HERE, 0, portSerial1);
        start_homing(adr, portSerial1);
        break;
    case torqueRight: // Točivý moment jeden směr
        target_torque(adr, dataToDriver.torque1, portSerial1);
        max_speed_rpm(adr, dataToDriver.max_rpm1, portSerial1);
        max_current(adr, dataToDriver.milli_amps1, portSerial1);
        invert_direction(adr, false, portSerial1);
        din0(adr, TORQUE_MODE, portSerial1);
        break;
    case torqueLeft: // Točivý moment druhý směr
        target_torque(adr, dataToDriver.torque1, portSerial1);
        max_speed_rpm(adr, dataToDriver.max_rpm1, portSerial1);
        max_current(adr, dataToDriver.milli_amps1, portSerial1);
        invert_direction(adr, true, portSerial1);
        din0(adr, TORQUE_MODE, portSerial1);
        break;
    case emergencyStop: // Nouzové zastavení
        quick_stop(adr, portSerial1);
        break;
    case startHoming: // Dojetí na výchozí polohu
        invert_direction(adr, false, portSerial1);
        homing_method(controller_adr, MECH_NEGATIVE, 0, portSerial1);
        homing_current(controller_adr, HOMING_I_MA, portSerial1);
        set_homing_offset(adr, OFFSET1, portSerial1);
        start_homing(adr, portSerial1);
        break;
```

```
    case stayStill: // Zastavení servomotoru
        profile_speed(adr, 0, portSerial1);
        din0(adr, PULSE_MODE, portSerial1);
        goto_home_position(portSerial1);
        break;
    case resetDriverErrors: // Resetování chybových hlášení
        reset_errors(controller_adr, portSerial1);
        break;
}
```

Příloha D – Funkce pro přepočet točivého momentu obou řidiček

```
void * recalc_torque_demand() {
   float prirustek = 0;

   // Přečtení aktuální pozice; zpětná vazba na polohu enkodéru
   dataToPC.current_position1 = -read_packet(adr, CURRENT_POS, portSerial1);

   if((dataToPC.current_position1 < MAX_POS_RIDICKY) { // Meze polohy
        if(dataToPC.current_position1 < 0)
            dataToPC.current_position1 = 0;
        prirustek = (float) (((float) dataToPC.current_position1 /
                  (float) MAX_POS_RIDICKY) * (float) MAX_TORQUE_PRIRUSTEK);
        prirustek *= prirustek;
        torque_recalculated = dataToDriver.torque1 + (int16_t) prirustek;
        // Nastavení přepočteného točivého momentu řidičce
        target_torque(adr, torque_recalculated, portSerial1);
   }

   dataToPC.current_position2 = -read_packet(adr, CURRENT_POS, portSerial2);

   if ((dataToPC.current_position2 < MAX_POS_RIDICKY) {
        if(dataToPC.current_position2 < 0)
            dataToPC.current_position2 = 0;
        prirustek = (float) (((float) dataToPC.current_position2 /
                  (float) MAX_POS_RIDICKY) * (float) MAX_TORQUE_PRIRUSTEK);
        prirustek *= prirustek;
        torque_recalculated = dataToDriver.torque2 + (int16_t) prirustek;
        target_torque(adr, torque_recalculated, portSerial2);
   }
}
```