Západočeská univerzita v Plzni
Fakulta elektrotechnická

Bakalářská práce

Návrh systému řízení pro simulátor padákového kluzáku

Jakub Vorlíček

květen 2021

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jakub VORLÍČEK**
Osobní číslo:	**E18B0140P**
Studijní program:	**B2644 Aplikovaná elektrotechnika**
Studijní obor:	**Aplikovaná elektrotechnika**
Téma práce:	**Návrh systému řízení pro simulátor padákového kluzáku**
Zadávající katedra:	**Katedra výkonové elektroniky a strojů**

Zásady pro vypracování

1. Prostudujte základní principy ovládání padákového kluzáku a jeho řídícího systému.
2. Navrhněte mechanickou část ovládacích prvků.
3. Navrhněte senzoriku ovládacích prvků.
4. Navrhněte zpětnovazební řízení aktivních členů ovládacích prvků.
5. Navrhněte a otestujte vhodnou konstrukční topologii ovládacích prvků simulátoru pro výcvik pilotů.

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.S.

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

Prof. Ing. Václav Kůs, CSc.
vedoucí katedry

ANOTACE A KLÍČOVÁ SLOVA

Tato bakalářská práce pojednává o návrhu a implementaci systému řízení pro simulátor vlastností padákového kluzáku včetně topologie pohonu. Obsahuje také rozbor principů ovládání a chování skutečného kluzáku. Zároveň se tato práce zaměřuje na problematiku komunikace mezi jednotlivými prvky komunikačního řetězce, konkrétně na kanál mezi počítačem a mikrokontrolérem a návrh vhodného uživatelského rozhraní pro testování této komunikace.

KLÍČOVÁ SLOVA

Parakluzák, vrchlík, aerodynamická síla, vztlak, pohony, servo motory, senzory, Unity, komunikace, uživatelské rozhraní

ANOTATION AND KEYWORDS

This bachelor thesis deals with design and implementation of control system for paraglide properties simulator including drive topology. It also includes analysis of control and behaviour properties of actual paraglide. Simultaneously it focuses on communication problematics between individual elements in communication chain, specifically in channel between PC and microcontroller and in design of suitable user interface for communication testing.

KEYWORDS

Paraglide, canopy, aerodynamic force, buoyancy, drives, servo motors, sensors, Unity, communication, user interface

PODĚKOVÁNÍ

Chtěl bych poděkovat týmu kolem projektu FlyOnVision za jejich dobře odvedenou práci na simulátoru, v první řadě pak svému vedoucímu Ing. Petru Kropíkovi, Ph.D., který byl vždy ochoten nasměrovat mě správným směrem a Petru Staškovi a Radku Spurnému za pomoc při montáži a za motivaci.

PROHLÁŠENÍ

Předkládám tímto k posouzení bakalářskou práci, zpracovanou během mého studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto práci vypracoval samostatně, s použitím uvedené odborné literatury, a pramenů a že veškerý software, použitý při jejím řešení a zpracování, byl využit s respektováním všech jeho licenčních podmínek.

V Plzni, dne 25.05.2021 Jakub Vorlíček

SEZNAM SYMBOLŮ A ZKRATEK

UI	User Interface	Uživatelské rozhraní
VR	Virtual Reality	Virtuální realita

OBSAH

1

ÚVOD

Tato bakalářská práce je zaměřena na návrh vhodné topologie simulátoru padákového kluzáku (Projekt FW01010257 - Simulátor letových vlastností padákového kluzáku), konkrétně jeho řízení. Věnuje se modelování a sestrojení mechanické navíjecí části, umístění senzorů, získávání informací o činnosti pilota a řízení pohonů.

Simulátor padákového kluzáku je projekt, jehož vznik byl iniciován kvůli dosavadní absenci bezpečného a dostupného trenažéru pro začínající piloty padákového kluzáku. Cílem celého projektu je proto vytvořit takové prostředí, které by umožnilo simulování krizových situací v bezpečí na zemi bez rizika zranění a nutnosti vyhledávat tyto nebezpečné situace. Zároveň by tato technologie měla posloužit jako prostředek k zacvičení nových pilotů, kteří ještě neměli možnost získat dostatek zkušeností.

2

JAK FUNGUJE PARAKLUZÁK

2.1 KONSTRUKCE KLUZÁKU

ZÁKLADEM pro pochopení fungování a ovládání paraglidu je jeho konstrukce. V zásadě jej můžeme rozdělit na vrchlík, šňůry a popruhy.

Vrchlík

Nejvýraznější částí kluzáku je vrchlík. Jedná se o žebrovanou konstrukci potaženou neprodyšnou prošívanou látkou, nejčastěji z nylonu natřeného chemickým nátěrem. Žebra jsou tvarovaná do specifického tvaru leteckého profilu. Zároveň jsou vyztužena pevnějším materiálem v místech, kde jsou k žebrům připevněny šňůry, což vede k rovnoměrnému rozložení tahu šňůr. Uvnitř vrchlíku se také nacházejí vyrovnávací otvory pro vyrovnávání tlaku vzduchu. [1]

Šňůry

Hlavní nosné šňůry parakluzáku se pod vrchlíkem větví do takzvané galerie, která následně zajišťuje spojení s vrchlíkem. Řady hlavních šňůr se označují písmeny A až D a v závislosti na tom, komu je kluzák určen, je možné, že některé řady budou spojeny. Šňůry se vyrábějí například z Kevlaru nebo Dyneemy. Tyto materiály se vyznačují malou průtažností, díky čemuž jsou dlouhodobě zachovány letové vlastnosti a geometrie kluzáku. Kevlar využívaný pro závodní kluzáky sice trpí na křehkost a citlivost na UV, avšak kvůli absenci ochranného opletu je jeho aerodynamický odpor mnohem nižší.

Dále na kluzáku najdeme šňůry řídící. Jejich popisu se věnuje kapitola 2.5. [1]

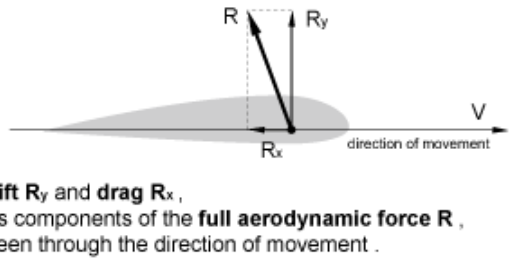
Popruhy

Popruhy spojují sedačku se šňůrami a zároveň k nim je připojeno řízení kluzáku, konkrétně speedsystém a řidičky. [1]

2.2 AERODYNAMIKA

Vrchlík parakluzáku se v aerodynamice jeví jako křídlo. Díky asymetrickému tvaru na něj při pohybu působí vztlak a odpor prostředí (vzduchu). Složením těchto sil získáme výslednou aerodynamickou sílu působící na křídlo kluzáku. Abychom zvýšili vztlak a vytvářeli co nejmenší odpor, využíváme křídlo s asymetrickým tvarem a snažíme se ho situovat asymetricky k proudění vzduchu.

Zaměříme-li se na sílu, která způsobuje pohyb vpřed, zjistíme, že tvar vrchlíku nám umožňuje přeměnit gravitační sílu, která působí na kluzák a pilota, s pomocí vztlaku na dopředný pohyb. Kluzák je tedy schopen pohybu vpřed díky tomu, že má hmotnost a že je umístěn v prostoru tvořeném tekutinou - vzduchem. [2]



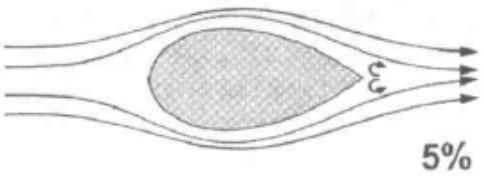
Obr. 1: R - Aerodymanická síla, Rx - odpor vzduchu, Ry - vztlak, V - směr pohybu [2]

Vzduch ovšem klade zároveň tělesu, které se v něm pohybuje, odpor. Tvar křídla tudíž musí být koncipován tak, aby byl tento odpor co nejnižší a kluzák zbytečně neztrácel rychlost.

V případě, že je do proudu vzduchu kolmo umístěna rovná deska, vzniká maximální odpor vzduchu a za deskou zároveň budou vznikat turbulence, jelikož proudění vzduchu není schopno plynule obtékat ostrá zakončení.

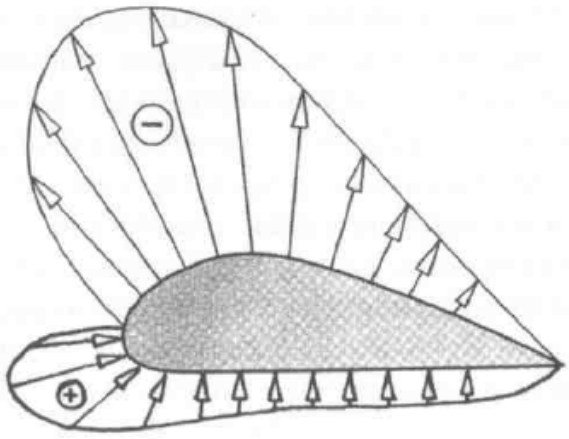
U koule vložené do proudu vzduchu je odpor přibližně čtvrtinový oproti desce, avšak za koulí stále vzniká turbulentní víření.

Při použití tělesa ve tvaru kapky je již odpor vzduchu asi dvacetinou oproti desce. Vzduch značnou část trasy kolem tělesa sleduje jeho tvar a až u konce přechází do turbulencí. Takovéto těleso proudnicového tvaru se sice vyznačuje nízkým odporem vzduchu, avšak vztlak je nulový, tudíž je pro vrchlík kluzáku nevhodný. [1]



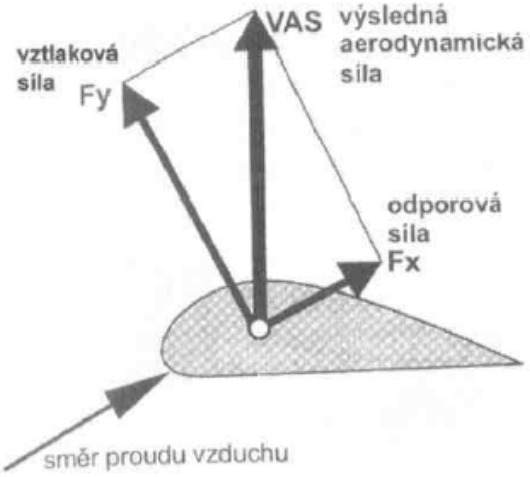
Obr. 2: Proudění vzduchu kolem tělesa proudnicového tvaru [1]

Vztlak vzniká díky rozdílné rychlosti proudění vzduchu pod a nad profilem, konkrétně to znamená, že je nutné, aby nad profilem vzduch proudil rychleji a vytvořil se tak na horní straně profilu podtlak. Využitím leteckého profilu vzniknou potřebné rozdíly tlaku (vztlak), které umožní vznik aerodynamické síly.



Obr. 3: Letecký profil se znázorněným přetlakem a podtlakem [1]

Jak již bylo zmíněno, složením vztlaku a odporu vzduchu vznikne výsledná aerodynamické síla, na obrázku 4 je oproti obrázku 1 znázorněn směr proudu vzduchu pod úhlem, což lépe koresponduje s nasazením kluzáku v letu.



Obr. 4: Skládání sil ve výslednou aerodynamickou sílu [1]

2.3 ÚHEL NÁBĚHU

Úhel, pod kterým je křídlo v prostoru vůči proudu větru nakloněno, se nazývá úhel náběhu. Během letu tento úhel určuje styl letu a pilot jej může zásahem do řízení měnit.

Mění se však nejen přičiněním pilota, ale je ovlivněn i podmínkami okolí. Při vletu do proudu vzduchu dojde k sumaci rychlosti vrchlíku s rychlostí proudění a tím i ke změně úhlu náběhu. [1]

Minimální klesání

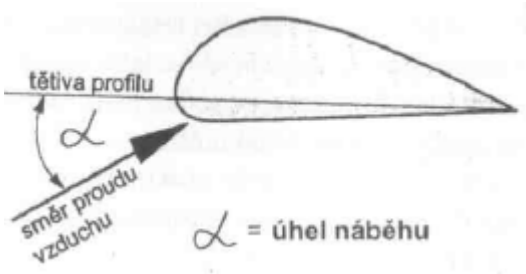
Díky velkému úhlu náběhu se křídlo pohybuje pomalu vpřed a jen mírně klesá, proto poskytuje tento styl nejdelší dobu letu. Pokud by byl úhel náběhu příliš velký může dojít k přetažení.

Nejlepší klouzavost

Jedná se o vyvážený styl, kde je poměr rychlosti pohybu vpřed a klesání nejvyšší. Díky tomu je trasa letu nejdelší, jaká může být.

Maximální rychlost

Křídlo rychle klesá a trajektorie letu je strmá. Dosahujeme nejvyšší možné rychlosti. [2]



Obr. 5: Vyznačení úhlu náběhu [1]

2.4 KLOPIVÝ MOMENT

Za rovnovážného stavu je výsledná aerodynamická síla v ose s tíhou, kterou působí pilot s kluzákem. Pokud se však tyto dvě síly dostanou mimo společnou osu, vznikne klopivý moment, který způsobuje, že se vrchlík svým pohybem snaží tyto síly opět dostat na společnou osu.

Při změně úhlu náběhu dochází zároveň ke změně polohy působiště aerodynamické síly a tím i ke vzniku klopivého momentu. Tato skutečnost například umožňuje startování kluzáku, kdy vrchlík stojí kolmo k proudění vzduchu, tím pádem je působiště aerodynamické síly v jeho zadní části a klopivý moment nutí vrchlík vystoupat nad pilota. [1]

2.5 ŘÍZENÍ PARAKLUZÁKU

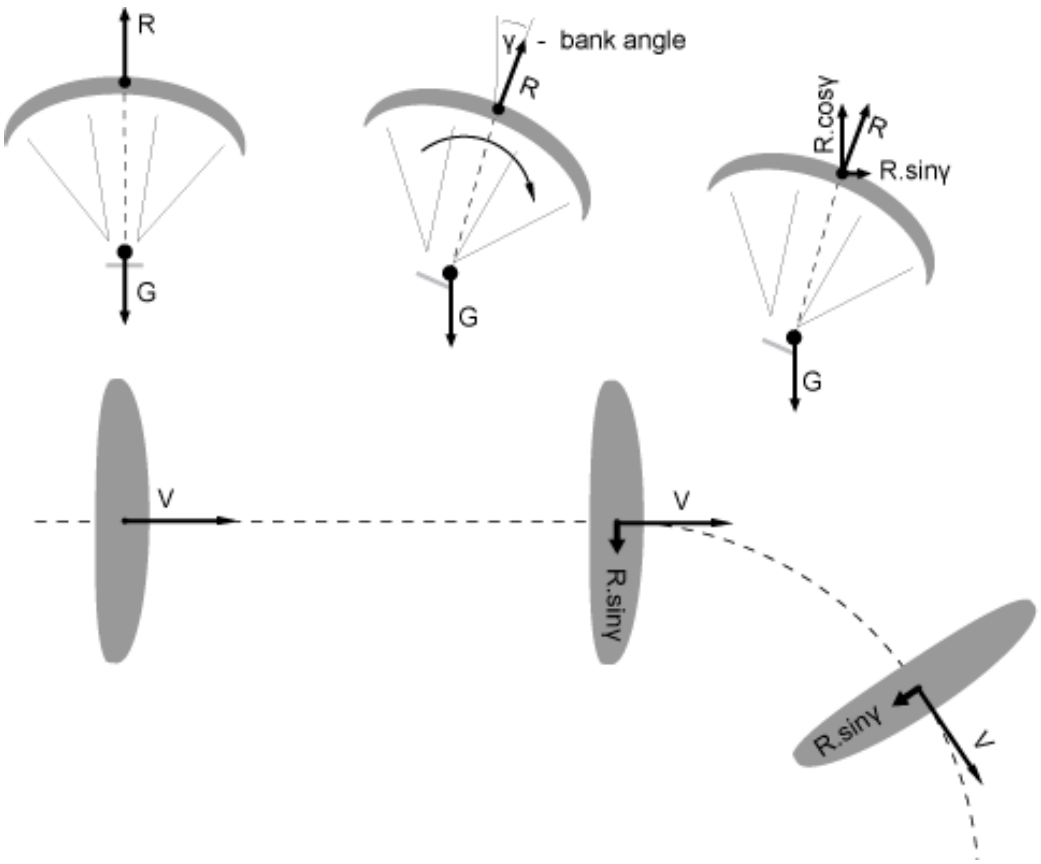
Řízení parakluzáku lze rozdělit do dvou částí: těžištěm a aerodynamicky.

Těžištěm

Změníme-li pozici těžiště parakluzáku (např. náklonem pilota), dojde k rozstředění gravitační a aerodynamické síly.

Přesune-li pilot těžiště svého těla do strany, způsobí tím nerovnoměrné zatížení křídla a to se nakloní do strany. Díky tomu dojde i ke změně úhlu působení aerodynamické síly, jejíž horizontální složka ovlivní původní pohyb vpřed a parakluzák zatočí ve směru náklonu.

Nevýhodou je, že nakloněné křídlo se jeví jako přetížené. Dochází k rychlejšímu klesání, které je způsobeno tím, že čím větší je náklon kluzáku, tím menší část aerodynamické síly působí přímo vertikálně. [2]



Obr. 6: Průběh zatočení pomocí posunu těžiště [2]

Aerodynamické řízení

Jedná se o změnu stavu různých částí křídla (např. ohýbáním), typicky pomocí řidiček nebo speedu.

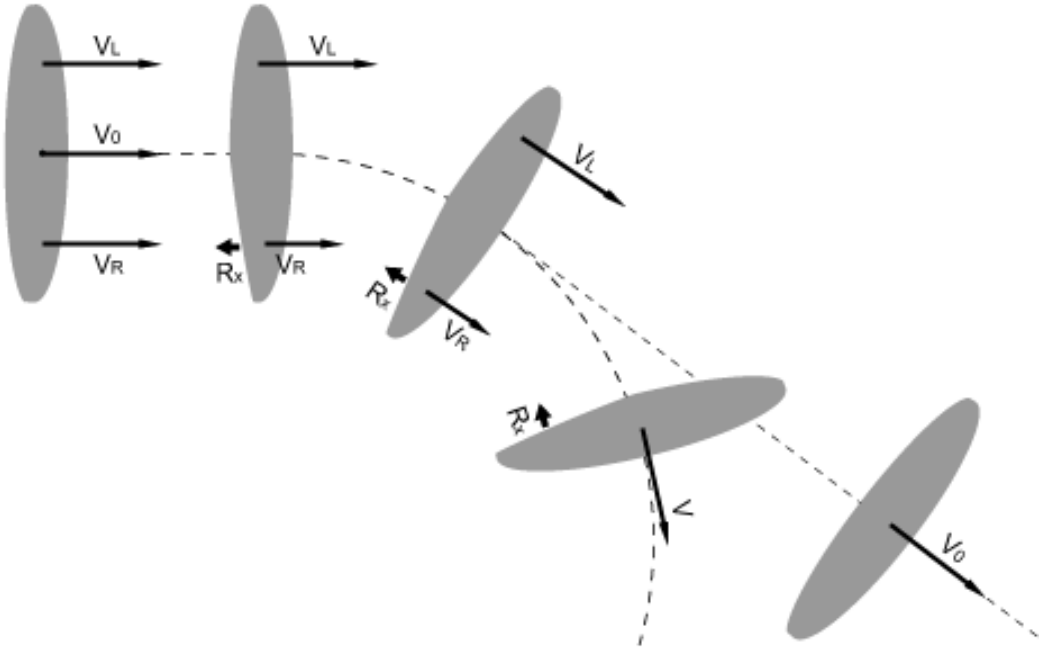
Řidičky

Zatáhnutí za obě řidičky způsobí deformaci zadní části křídla, což zvýší odpor vzduchu. Díky tomuto jevu se křídlo zpomalí a zvýší se úhel náběhu. To dočasně zvýší vztlak na úkor rychlosti a umožní pilotovi vstoupit to nového stylu letu s vyšším úhlem náběhu, strmější trajektorií a nižší rychlostí.

Další úlohou řidiček je brzdění nežádoucích zrychlení způsobených termálními proudy a poryvy větru nebo vlivy po zotavení z přetažení. Tato nežádoucí zrychlení způsobují, že křídlo předbíhá pilota a může i zkolabovat. Tvrdé brzdění řidičkami by nemělo probíhat déle než 1 vteřinu, aby nedošlo k přetažení. Křídlu musí být umožněno vrátit se k normální letové rychlosti.

Zatáhnutí za pouze jednu řidičku zdeformuje půlku křídla, která následně zpomalí, zatímco druhá půlka pokračuje normální rychlostí. Parakluzák zatočí směrem k odpovídající řidičce a až po uvolnění řidičky poletí rovně.

Prudké zatažení za jednu řidičku může způsobit obdboný efekt jako zatáčení pomocí náklonu pilota. [2]



Obr. 7: Průběh zatočení pomocí řidičky [2]

Speed

Speedbar (též speed systém nebo šlapák) je část řízení, která po zatažení nohama za lano pod sedačkou, zatáhne zároveň za 2 přední šňůry spojující sedačku s křídlem. Při plném napnutí speed systému se dostává parakluzák do stylu letu s maximální rychlostí. [2]

2.6 ŘEŠENÍ LETOVÝCH SITUACÍ

Tato kapitola vysvětluje a popisuje situace, do kterých se pilot během letu může dostat. Jelikož pro začínající piloty mohou být stresující a pouze teoretická znalost nemusí nutně stačit k jejich vyřešení, budou právě tyto stavy předmětem simulací na hotovém simulátoru letových vlastností padákového kluzáku.

Klesavé proudy

Při vletu do klesavého proudu se vlivem snížení náběžného úhlu vrchlík krátkodobě zrychlí. Pokud pilot vletí do silného klesavého proudu a nezbrzdí vrchlík, může dojít i k zaklopení náběžné hrany. [1]

Stoupavé proudy

Ve stoupavém proudu vrchlík mírně zvýší svůj úhel náběhu a vzniká kladný klopivý moment. Snižuje se opadání kluzáku a může nastat i stoupání kluzáku. [1]

Turbulence

Při turbulenci může docházet k nesymetrickým zaklopením a deformacím vrchlíku nebo náklonům na stranu, což jsou jevy, na které musí pilot intuitivně reagovat a eliminovat je. Turbulentní poryvy jsou většinou jen krátkodobé a proto kluzák velmi rychle přechází mezi přechodovými režimy. Při letu ve vysoce turbulentních podmínkách mohou však být změny v chování vrchlíku výraznější, tudíž se můžou stát pro začínajícího pilota nezvladatelnými. [1]

Přistání

Parakluzák je díky své konstrukci schopen samovolného přistání, obtížnější však už pro pilota může být přistát na přesně určeném místě v prostoru. Aby pilot dosáhl přesného přistání, musí si správně připravit takzvaný rozpočet na přistání. Důležitá je informace o vzdálenosti, kterou chce uletět, a také jaké má k dispozici převýšení. Z těchto dat je možné přibližně určit, jakou klouzavost musí kluzák mít, avšak je nutné mít na paměti, že v průběhu letu bude docházet ke změnám režimů, což povede k dočasnému zhoršení klouzavosti. Značně dolet ovlivní i protivítr nebo například klesavé a stoupavé proudy.

Jelikož správný přistávací manévr se provádí proti větru, musí si pilot správně naplánovat z jakého směru bude přistávat. Zohlednit se samozřejmě musí také terén v okolí místa přistání.

Těsně před přistáním pilot vysedá z postroje sedačky a s nohama mírně pokrčenýma u sebe se připravuje k dopadu. Je nutné sledovat rychlost protivětru a v závislosti na jeho síle případně přibrzdit. Kousek nad povrchem pak pilot stahuje řidičky podél těla a dopadá na obě nohy zároveň. [1]

Boční kývání

Dostane-li se pilot vlivem turbulence nebo chybou při řízení do stavu bočního kývání, hrozí například asymetrické zaklopení vrchlíku.

Optimální řešení situace je sledovat svoji polohu oproti prostoru a ve chvíli, kdy je pilot uprostřed zhoupnutí na druhou stranu, to je přímo pod vrchlíkem, stahuje řidičku na straně, ze které se právě zhoupl. Tento úkon spolehlivě kývání zastaví. Pokud pilot zatáhne řidičku na druhé straně, způsobí, že se kývání ještě umocní. Pomalejším řešením je kluzák rovnoměrně zbrzdit a nehybně čekat na uklidnění kyvů. [1]

Sackflug

Sackflug nebo-li padákový režim je stav, kdy kluzák přechází do pádu s odtrženým řízením. Dojít k němu může při řešení jiných krizových situací, případně při pomalém letu s malým opadáním.

Prvním krokem k jeho eliminaci je uvolnění řízení, což ve většině případů stačí k nápravě. Pokud tato činnost nepomůže, je nutné provést prudký a symetrický zásah do řízení, aby vrchlík přešel do přechodného režimu. Po obou těchto akcích je nutné s dalším manévrováním počkat, než kluzák opět nabere rychlost. [1]

Frontstall

Zaklopení náběžné hrany je způsobeno přílišným snížením úhlu náběhu. Častými viníky kromě pilota můžou být turbulence, silné klesavé proudy a předbíhání vrchlíku před pilota.

Tato situace se obvykle vyřeší samovolně, vrchlík se zbrzdí a kvůli zvýšenému opadání se pilot pomalu vrací zpět pod vrchlík a dochází k opětovnému dofouknutí vrchlíku. Pokud se vrchlík sám nedofoukne, musí pilot zatáhnout za obě řidičky a počkat na regeneraci vrchlíku. Následně je nutné nenechat se překvapit zrychlením vrchlíku a plynule jej po návratu do rovnovážné polohy odbrzdit. [1]

Asymetrické zaklopení vrchlíku

Jedná se v podstatě o částečný frontstall, kdy dojde k zaklopení pouze části vrchlíku. Kluzák v této situaci začíná zatáčet a pilot cítí uvolněné řízení na postižené straně.

Aby kluzák nezačal rotovat, musí pilot zbrzdit nezborcenou stranu, čímž se srovnají rychlosti obou stran kluzáku a zborcená strana se může samovolně dofouknout. Urychlit dofukování je možno pumpováním řízení. [1]

Boční zaklopení

Boční zaklopení připomíná to asymetrické, avšak zde dochází k zaklopení okraje s osou kluzáku. Chování kluzáku se také změní obdobně jako při asymetrickém zaklopení. Problém je mnohem větší postižení řízení na zaklopené straně.

Většinou se toto zaklopení vyřeší samovolně díky srovnávání tlaku uvnitř vrchlíku. Zotavení je oproti předchozímu případu delší a pilot jej může urychlit zatáhnutím za zadní popruh. [1]

Zachycení vrchlíku o šňůry

Tento defekt může ve vzácných případech doprovázet zaklopení. Pokud se část vrchlíku zachytí o šňůry, je znemožněno jeho dofouknutí.

Jedním řešením může být přechod do Frontstallu, při němž dojde k nárazovému dofukování a pravděpodobně i uvolnění vrchlíku. Jestliže k uvolnění nedojde a deformace vrchlíku je příliš velká a nebezpečná, je pilot nucen využít záložní padák.

Šňůra zároveň může zachytit vrchlík ještě před startem kvůli nedůsledné kontrole. Takto zamotaná šňůra se během letu nedá uvolnit a tak s ohledem na řiditelnost deformovaného vrchlíku se pilot buď pokusí přistát nebo využije záložní padák. [1]

Fullstall

Přetažení vrchlíku je stav, který může způsobit jedině pilot necitlivým řízením. Pokud pilot příliš intenzivně brzdí a stahuje řízení moc nízko. V takovém případě začne kluzák zpomalovat a následně se uvolňuje řízení, až do okamžiku, kdy se vrchlík vyfoukne a dochází k pádu. [1]

Vývrtka

Negativní zatáčka je defekt, který vzniká například při prudkém řízení při zatáčení nebo při pokusu o spirálu. Vnitřní strana vrchlíku je postavena pod větším úhlem náběhu než ta vnější, tudíž hrozí její přetažení. Při pokusu o vývrtku je varováním, že stahování řízení nezpůsobuje vzrůstání rychlosti a náklonu kluzáku. Pokud pilot nerespektuje tyto znaky, dojde ke změknutí vnitřní strany vrchlíku, která začne oproti vnější straně couvat a kluzák začíná rotovat kolem osy, která protíná vrchlík.

Pokud nedojde k zamotání šňůr, musí pilot okamžitě přerušit manévr, uvolnit řízení a připravit se na brzdění vrchlíku. Jinak už nezbývá nic jiného, než použít záložní padák.

Aby se pilot negativní zatáčce vyhl, nikdy intenzivně dlouhodobě nebrzdí a věnuje pozornost stavu šňůr, jelikož staré polyesterové šňůry trpí na protahování a tím se zvyšuje i riziko vývrtky. [1]

2.7 MANÉVRY

Pokud pilot potřebuje například z důvodu klimatických změn rychle klesnout může použít některý z následujících manévrů.

B stall

Tento manévr se provádí zatažením za B řadu šňůr, což způsobuje podélnou deformaci vrchlíku. Tato deformace znemožní let, proto začne celý kluzák prudce padat a zpomaluje jej pouze deformovaný vrchlík.

Při uvolňování šňůr je nutné nejprve povolit přibližně o půl stažené dráhy a následně úplně pustit. Při nedodržení tohoto postupu nebo při nedostatečně pevném držení šňůr hrozí přechod do vývrtky. Manévr se doporučuje pouze s velkou výškovou rezervou. [1]

Zaklopení stabilizátorů

Pro zaklopení uší uchopí pilot jednu nebo dvě přední řady šňůr a stáhne je. Tím se zdeformují konce vrchlíku a dochází ke klesání. Pokud v průběhu manévru pilot zároveň zatáhne za speed, klesání se ještě urychlí.

Ukončení manévru se provádí přibrzděním kluzáku téměř na polovinu rychlosti a následně se čeká na dofouknutí konců vrchlíku. I tento manévr se doporučuje provádět pouze s dostatečnou výškovou rezervou. [1]

Spirála

Aby pilot mohl spirálu provést, musí nejprve nabrat dostatečnou rychlost. Následně plynule stahuje řízení a sleduje, zda rychlost a přetížení narůstá. Správná spirála zajistí klesání 15 m/s s velkým náklonem.

Provedení spirály může být značně zkomplikováno turbulencemi, při nichž hrozí zaklopení vnější strany vrchlíku a pád do vývrtky.

Při správném ukončování spirály pilot postupně uvolňuje řízení a dochází k postupnému zpomalování rychlosti kluzáku a zvětšování průměru otáčení. Je nutno myslet na dostatečnou výškovou rezervu. [1]

3

MOTORY

Pro zajištění polohování pilota a simulace odezvy řidiček je využita sada několika servo motorů. Jedná se o výrobky firmy Kinco a to konkrétně:

- Dvě serva SMH60S-0040-30AAK-3LKH (314B/K1) pro řidičky
- Čtyři serva SMH80S-0100-30AAK-3LKH (334B/K3) pro polohování pilota ve vodorovném směru (dále jako boční pohony)
- Jedno servo SMH110D-0157-30ABK-4HKC (384B/K8) pro polohování pilota ve svislém směru (dále jako horní pohon)

Motory řidiček a bočních pohonů slouží jako navijáky, u horního pohonu pak motor otáčí šroubem posuvného mechanismu, na kterém je zavěšena sedačka s pilotem a protizávaží.

Tab. 1: Jmenovité hodnoty motorů [3]

motor	výkon [W]	proud [A]	kroutící moment [Nm]	rychlost [ot/min]	napájecí napětí [V]
314B/K1	400	3,1	1,27	3000	220
334B/K3	1000	6,3	3,18	3000	220
384B/K8	1570	5,9	5,00	3000	380

Tab. 2: Další parametry motorů [3]

motor	setrvačnost	mechanická brzda	počet pólpárů
314B/K1	nízká	ne	3
334B/K3	nízká	ne	3
384B/K8	střední	ano	4

Z bezpečnostních důvodů byl vybrán pro horní pohon motor s brzdou. V případě nouzového odpojení od napájení pilot setrvá v aktuální výšce, což značně snižuje riziko úrazu.

3.1 DRIVERY

Každý motor je vybaven vlastním driverem, který umožňuje přesně řídit jeho chování a kontrolovat skutečný stav motoru.

Driver přijímá příkazy od nadřazeného systému a v závislosti na nastavených parametrech ovládá chování připojeného motoru. Díky zpětné vazbě od integrovaných senzorů v motoru je driver schopen upravovat vlastnosti napájení tak, aby chování motoru odpovídalo požadavkům i při změně zatížení. Zároveň je driver schopen veškeré informace o motoru předat zpět nadřazené jednotce, takže informace od více driverů může uživatel komplexně využívat.

Drivery využité pro tento simulátor mají řadu výhod. Například není nutné upravovat řídící parametry typické pro daný druh motoru, jelikož driver při připojení sám rozpozná, připojený motor a parametry si optimálně nastaví. Uživateli stačí jen zkontrolovat, zda kód motoru, který je nyní v paměti driveru uložen, souhlasí s hodnotou na štítku. Pokud je tato hodnota v pořádku, je celkem jisté, že parametry pro regulaci budou optimální.

Pro využití informací z driveru si musí nadřazená jednotka vyžádat požadovaná data z paměti driveru. Data jsou ukládána na pevných adresách v paměti, takže pokud uživatele zajímá například aktuální poloha, zjistí si nejprve v datasheetu nebo pomocí softwaru od výrobce na jaké adrese se nachází a poté už může tyto informace o motoru vyčítat.

4

KONSTRUKCE ŘIDIČEK

V rámci celé konstrukce simulátoru byly využity různé materiály. Pro největší konstrukční části slouží hliníkové profily s vysokou modulovatelností. Pro jejich spojování se vyrábí rozmanité spojovací prvky, což umožňuje sestavení široké škály konstrukcí.

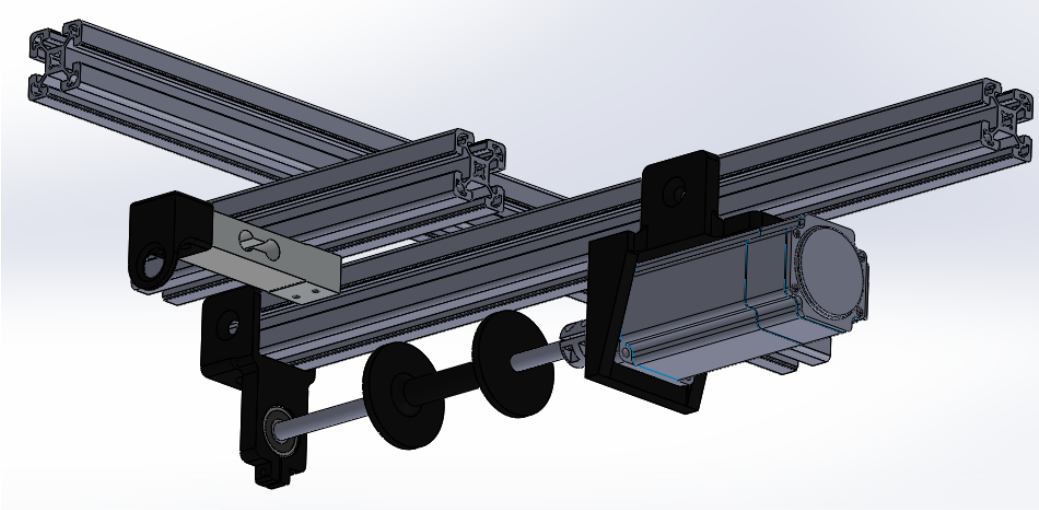
Menší specifické prvky byly vytištěny, díky čemuž mohly vzniknout spojovací a speciální díly, které přesně odpovídaly našim požadavkům. Pokud by tyto díly byly vyrobeny z jiných materiálů, bylo by nutné pro jejich poskytnutí kontaktovat externí firmy, což by několikanásobně zvyšovalo jejich cenu. Takto s využitím 3D tisku jsou díly levné a v případě jejich zničení, popřípadě při nevhodném návrhu, je náprava možná v rámci hodin maximálně dní. Hřídele, použité u navíjecích mechanismů, nejen u řidiček, jsou pak ocelové.

V průběhu vývoje vzniklo mnoho návrhů konstrukce mechanismu řidiček. Kromě mechanické odolnosti celého systému je důležité, aby zvolené tenzometry poskytovaly přesná data. Některé varianty jsou:

4.1 PŮVODNÍ NÁVRH

Jako první a nejjednodušší varianta vznikla soustava s jednou navíjecí cívkou a okem na tenzometru, které řídící lanko směrovalo na cívku.

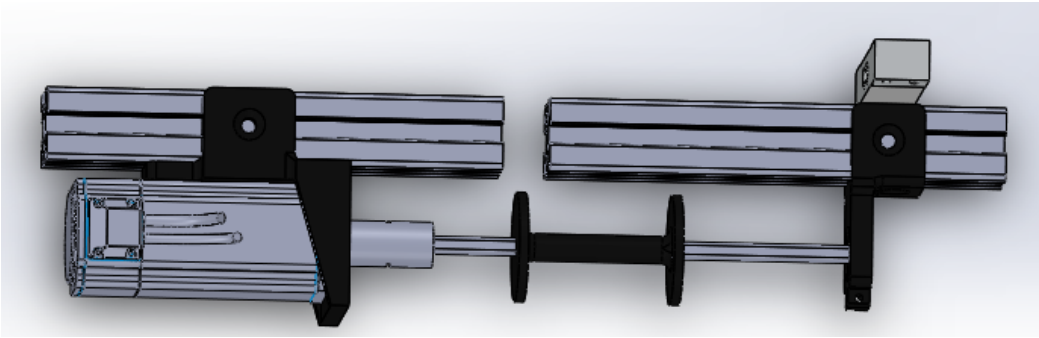
Toto řešení vyniká svou jednoduchostí, proto bylo při prvotním testování čtení dat z tenzometrů využito, avšak z hlediska tření lanka o plastové oko při chodu motoru bylo nutné toto řešení upravit.



Obr. 8: První varianta mechanické části

4.2 ROZDĚLENÝ NAVÍJECÍ MECHANISMUS

U této varianty byl rozdělen nosný profil napůl. Senzor by pak zaznamenával sílu, kterou pilot vyvíjí, přes páku složenou z osy cívky a pevného uchycení na motoru.

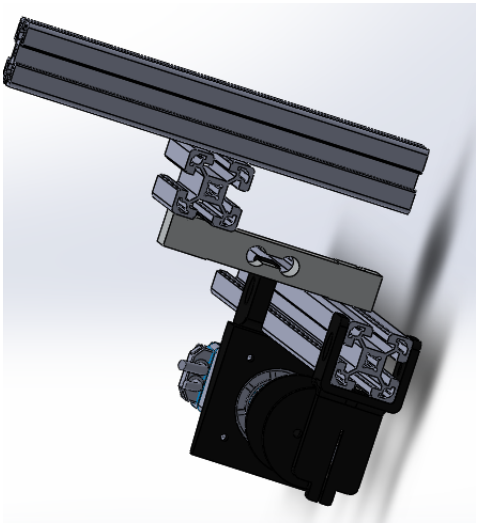


Obr. 9: Varianta s rozděleným navíjecím mechanismem

U této varianty by kvalita čtených dat silně závisela na materiálu hřídele. Pro získání kvalitních dat by bylo nutné zvolit hřídel z měkkého materiálu, což by však snížilo životnost celého mechanismu. Docházelo by k ohýbání hřídele a ničení ložiska motoru. Naopak pokud by byla hřídel vyrobena z tvrdého, nepoddajného materiálu, nedokázal by pilot pravděpodobně vyvinout dostatečnou sílu, aby mechanismus vychýlil tak, aby tenzometr zaznamenal kvalitní data, proto byla tato verze zavržena.

4.3 CELÝ MECHANISMUS ZAVĚŠENÝ NA TENZOMETRU

U další sestavy byl využit původní návrh a celý zavěšen na senzor.



Obr. 10: Varianta zavěšeného mechanismu

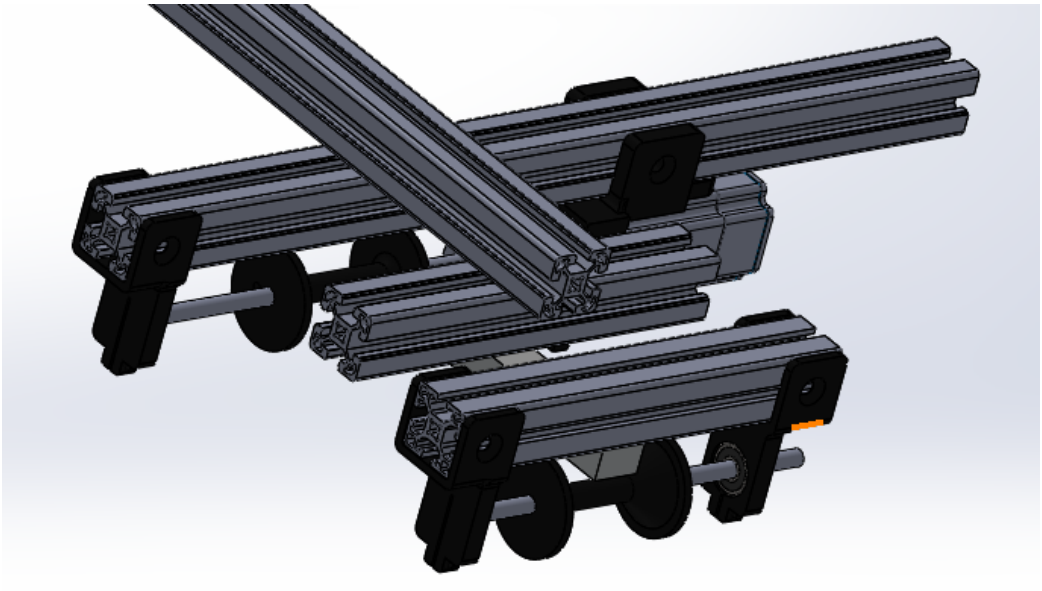
Z konstrukčního hlediska velmi jednoduché řešení, avšak je velmi pravděpodobné, že tenzometr by zaznamenával rázy způsobené motorem, což by vedlo ke zkreslení dat. Zároveň by byl senzor trvale zatížen hmotností motoru, tudíž by došlo ke snížení rozsahu.

4.4 FINÁLNÍ VARIANTA

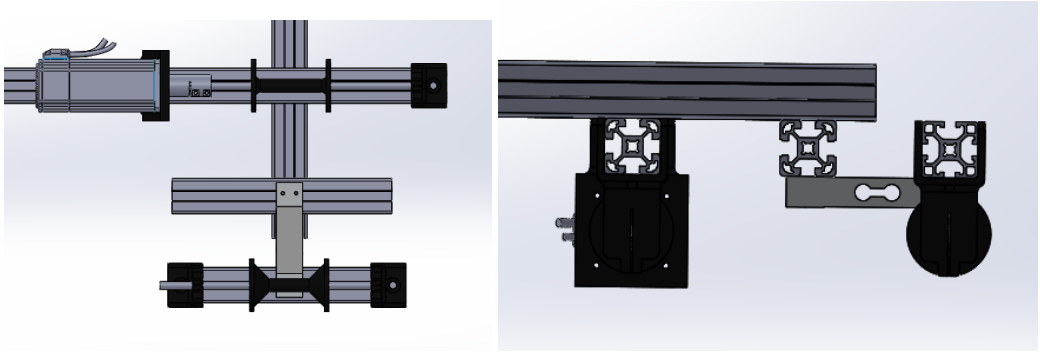
U posledního návrhu je využita soustava dvou cívek, z nichž první (směrovací) směruje provázek na druhou (navíjecí) cívku. První cívka je zároveň zavěšena na tenzometru, což umožňuje čtení dat prakticky bez vlivu vibrací motoru.

Zkosení na první cívce je velmi dobrým řešením pro pokrytí výkyvů pilota v sedačce a změn polohy sedačky v prostoru. Lanko je stabilně směrováno ke středu druhé cívky. Toto řešení z velké části vychází z původního návrhu, jedinou markantní změnou je doplnění o směrovací cívku a její uchycení.

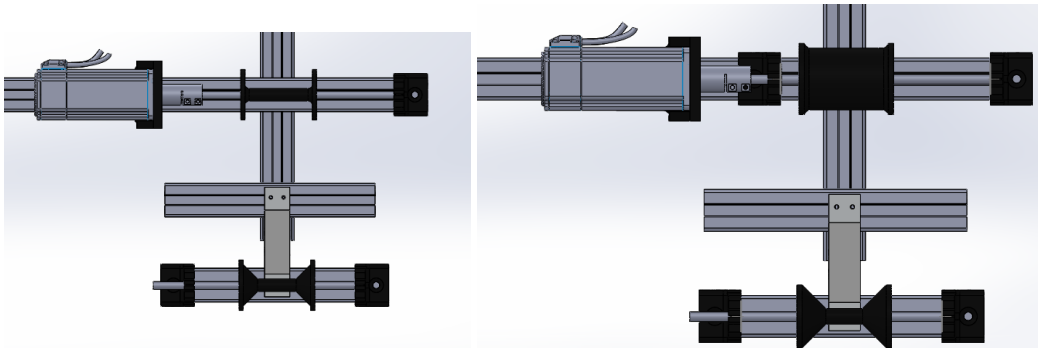
Během prvotního testování finální varianty došlo ke zjištění, že hodnota kroutícího momentu působícího na řidičkách je příliš vysoká. Zkušenému pilotovi se řidička jevila příliš tuhá, proto došlo ke zvětšení průměru navíjecí cívky z 15 mm na 59 mm, a tím i k téměř čtyřnásobnému snížení výsledného přenášeného momentu. Pokud by k tomuto řešení nedošlo, bylo by nutné nastavovat motoru tak nízké hodnoty momentu, že by nebyl vůbec využit jeho potenciál.



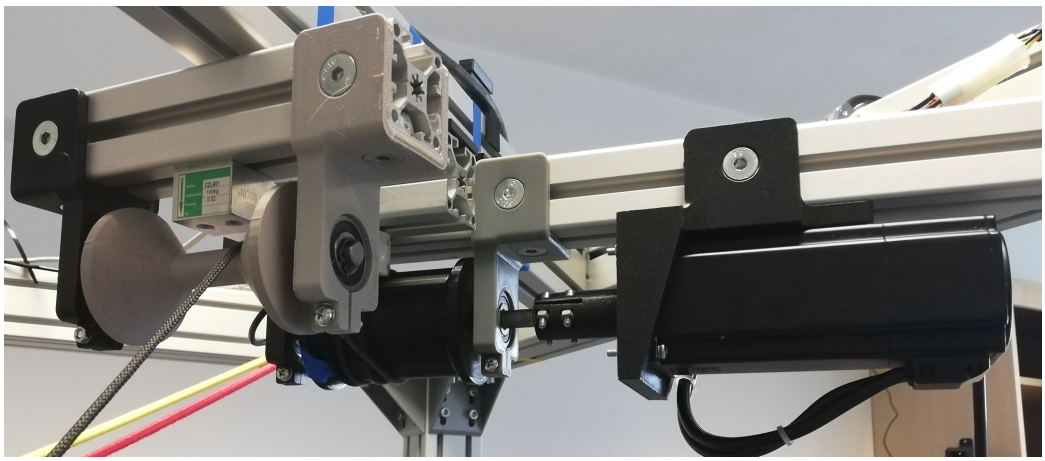
Obr. 11: Finální verze



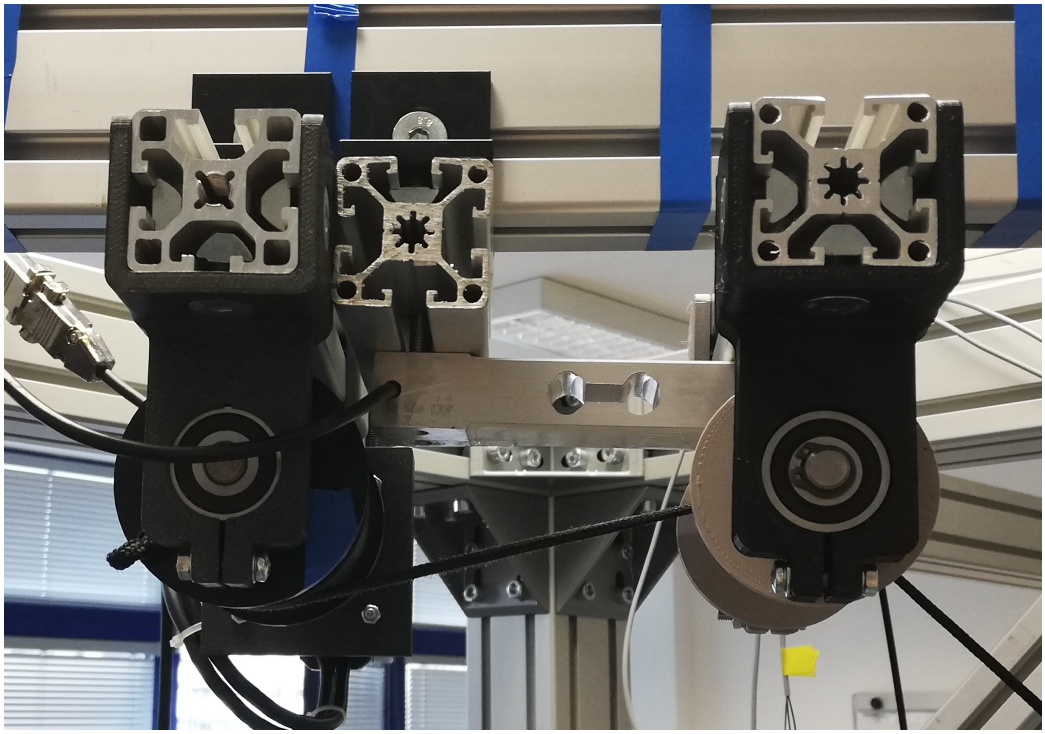
Obr. 12: Půdorys a bokorys finální verze



Obr. 13: Porovnání změn



Obr. 14: Foto finálního provedení



Obr. 15: Foto - Bokorys

5

SENZORIKA

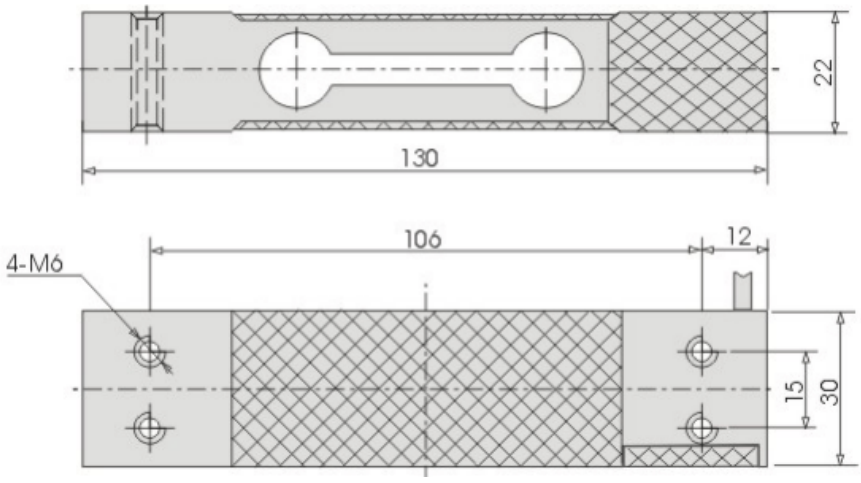
Pro přesné určení toho, co pilot v sedačce provádí za úkony, je nutné vybrat vhodné senzory a metody získávání dat o jeho poloze v prostoru.

5.1 TENZOMETRY

Jako první došlo k implementaci dvou tenzometrů typu CZL601, jejichž úkolem je monitorovat sílu, kterou pilot vyvíjí na řidičky. V kapitole Konstrukce řidiček již bylo vyobrazeno, kde a jakým způsobem jsou senzory upevněny.

Data získaná z tenzometrů jsou přes převodníky HX711 zasílány do Arduina, které následně data interpretuje jako pohyb joystickem a zasílá je do počítače. Zde již jsou tato data využita k ovládání kluzáku v počítačové simulaci.

Použité tenzometry CZL601 jsou koncipovány na hmotnost 100 kg s přesností 0,02%. Ačkoli se může zdát, že jsou senzory pro řidičky předimenzované, je nutné si uvědomit, že pilot se může za řidičky pověsit, zatáhnout je až pod sebe a při plném napnutí lanek se i nadzvednout, čímž celou svou váhu přenese právě na řidičky.



Obr. 16: Rozměry tenzometru [4]

24 bitové A/D převodníky HX711 jsou určeny pro přesné čtení analogového signálu z váhových senzorů. Převodník nevyžaduje žádné další úpravy v kódu, ovládání nastavitelných parametrů se provádí přivedením signálu na korespondující pin. [5]

Arduino již také sbírá data ze čtyř dalších tenzometrů, na kterých je zavěšena sedačka. Tato data zatím nejsou využita v simulaci, avšak v budoucnu mohou posloužit k určení náklonu pilota a rozložení jeho váhy, což jsou parametry, které jednoznačně určují chování parakluzáku.

5.2 DALŠÍ SENZORIKA

Pro další zjišťování polohy pilota se do budoucna počítá s použitím gyroskopického senzoru do samotné sedačky, který bude přesně informovat o náklonu pilota, popřípadě v rámci využití VR headsetu a ovladačů je možné zmíněné ovladače připevnit k sedačce.

VR

Aby byla simulace co nejvěrnější, bude pro zobrazování terénu pilotovi využito virtuální reality. Během posledních let došlo k rozmachu této technologie nejen v herním průmyslu. Nepřekonatelnou výhodou VR je pohyblivé zobrazení okolního prostředí, kdy na rozdíl od monitoru se zobrazený obraz posouvá v závislosti na poloze hlavy uživatele a dochází tak k naprostému ponoření do virtuálního prostředí. Většina VR headsetů je kombinována se sadou ovladačů pro interakci s prostředím.

Pro přesně určení polohy a natočení ovladačů a headsetu v prostoru se používá set stanic, které se umístí za okraj prostoru vyhrazeného pro pohyb uživatele. Tyto stanice pak monitorují VR headset a ovladače.

Data z encodérů

Motor a driver také poskytují zajímavé informace o stavu pohonu. Arduino do Unity zatím předává k budoucímu využití informace o aktuálních skutečných otáčkách, pozici, odebíraném proudu a teplotě. U obou pohonů řidiček také získáváme informaci o aktuálním momentu. Zvýšená hodnota proudu odebíraného pohonem indikuje vyšší zatížení daného pohonu, tím pádem je patrné, že pilot svou činností působí proti efektu vyvolanému daným pohybem. Z hlediska řízení tento údaj spolu s dalšími daty zajistí upřesnění informace o poloze, náklonu a činnosti pilota.

Arduino by mohlo posílat i mnoho dalších informací, např. chybové hlášky nebo aktuální rychlost pohonu, avšak pro ně zatím v projektu nebylo patrné využití. Není však vyloučeno, že v budoucnu budou pro simulaci některá další data nutná.

6

ŘÍZENÍ

Pro řízení motorů využíváme sériovou komunikaci mezi počítačem a Arduinem, které následně předává data v paketech driveru motoru. Tato práce je zaměřena na komunikaci mezi PC a Arduinem prostřednictvím Unity. Hlavním prostředkem pro usnadnění komunikace je pro nás projekt Ardity, který je přímo zaměřen na komunikaci Unity <-> Arduino. Autorem je vývojář Daniel Wilches.

6.1 ARDITY

Ardity vytváří vlákna, která sbírají data z daného COM portu do fronty. Tato data může pak uživatel dle libosti využít. Zároveň má uživatel možnost data odesílat. [6]

Obr. 17: Logo projektu Ardity

6.2 KOMUNIKACE

Pro naši aplikaci jsem využil funkce a metody implementované projektem Ardity. Hlavně funkci SendSerialMessage(), která předává zprávu do fronty, ze které je následně vytažena a prostřednictvím sériového vlákna odeslána na příslušný port. Velmi užitečná je také metoda OnMessageArrived(), která bez přičinění uživatele čte poslední přijatou zprávu umístěnou ve frontě s přijatými zprávami.

Důležitým parametrem komunikace mezi Unity a Arduinem je struktura zasílaných dat. Kód Arduina u naší aplikace vyžaduje, aby data, která obdrží, přišla jako řetězec bytů s přesně definovaným umístěním jednotlivých proměnných, proto v Unity existuje struktura obsahující všechny odesílané proměnné v předem domluveném pořadí.

Unity dostane od uživatele data v textové podobě. Ta jsou nejprve uložena v domluveném formátu jako číslo do struktury "PcData"a následně před odesláním dat musí dojít k převodu na pole bytů. Pokud by data byla odesílána jako text, resp. řetězec znaků, docházelo by před odesláním k překladu znaků podle kódování systému. Tudíž by ještě před odesláním dat došlo k jejich znehodnocení. Zasílání pole bytů tento problém dokonale řeší.

Na přijímaná data již nejsou kladeny takové nároky. Arduino zasílá hodnoty jako znaky a po odeslání celé zprávy umístí nakonec symbol konce řádku, který Unity chápe jako konec zprávy. Po přijetí zprávy dojde v Unity k rozdělení zprávy podle středníků a následně k parsování jednotlivých kusů zprávy do určených proměnných ve struktuře "DriverData". Tato data budou v budoucnu sloužit jako zpětná vazba pro simulaci.

MessageListener

Skripty MessageListener a MessageListener2 tvoří páteř celého projektu v Unity. V podstatě se jedná o téměř totožné skripty, každý z nich však spravuje jeden komunikační port, to znamená, že obsluhuje jedno Arduino. Protože každé z Arduin pracuje s jinými drivery, liší se skripty hlavně použitými proměnnými a obsluhovanými grafickými prvky. Co se metod týče jsou až na počet proměnných principiálně stejné. Zde následuje popis skriptu MessageListener2.

Při spuštění aplikace Unity volá metodu Start(). Ta obsahuje kromě inicializace také pokus o nahrání posledních uložených dat - metoda LoadLatestData(). V případě, že taková data neexistují, nahrají se data základní (InitializeDefaultData()).

Aby mohl uživatel začít ovládat pohony je nutné nejprve připojit Arduino/Arduina. Po zadání názvu portu a modulační rychlosti uživatel klikne na "Connect", což vyvolá metodu Connect2Clicked(). Unity použije zadané hodnoty a pokusí se připojit. Pokus o připojení volá metodu OnConnectionEvent(bool), která v závislosti na úspěšnosti připojení vypíše odpovídající hlášku.

Po připojení uživatel zadá pomocí grafických prvků InputField, Dropdown a Slider hodnoty, které si přeje poslat Arduinu, a klikne na "SEND". To spustí řetězec reakcí začínající metodou SendClicked() a následně SendA2Clicked(), která je pozůstatkem starší varianty. V metodě SendA2Clicked() se několikanásobným zavoláním metody DropownTranslate() vytvoří proměnné obsahující příkazy/režimy pro ovládané motory na základě výběru v daných Dropdwnech a ty jsou následně předány metodě StructUpdate(byte, byte, byte).

StructUpdate(byte, byte, byte) sesbírá hodnoty relevantní pro strukturu obsahující data k odeslání a naplní ji. Po aktualizaci hodnot ve struktuře je již možné přejít k jejich

odeslání - metoda MessageOutCombineAndSend(). Ta vytváří zprávu typu pole bytů a to za pomoci metod GetByteArray() a AppendArrays(). GetByteArray na základě datového typu převede číselnou hodnotu na pole bytů o vhodné velikosti a AppendArrays() následně tato pole nalepuje za sebe do finální zprávy.

Když je zpráva vytvořena již nic nebrání ji odeslat metodou z projektu Ardity - SendSerialMessage().

Uživatelské rozhraní obsahuje několik dalších tlačítek:

- "Enable A2 Drivers"a "Disable A2 Drivers"pro zapnutí a vypnutí všech driverů připojených k danému Arduinu
- "Restart A2"pro restart Arduina
- "STOP ALL"pro nouzové zastavení všech motorů

Tato tlačítka mají přiřazené vlastní metody *Clicked(), které pouze volají StructUpdate(byte, byte, byte), avšak příkazy/režimy jsou již předem nastaveny přímo v kódu.

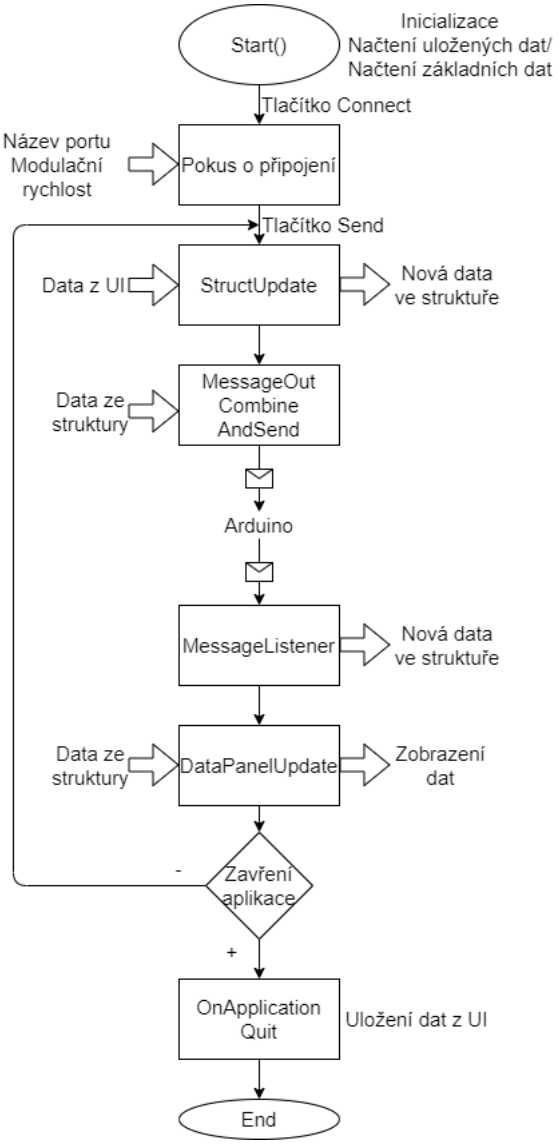
Arduino po té, co obdrží zprávu, obratem vrací zprávu s daty z driverů. Na to reaguje metoda OnMessageArrived(string), která příchozí zprávu rozdělí podle středníků a zkontroluje, zda první část zprávy obsahuje specifický podpis pro rozlišení, který MessageListener se má zprávě věnovat. Následně se v metodě OnMessageArrived() jednotlivé části zprávy ukládají do struktury s příchozími daty. Po uložení všech hodnot je zavolaná metoda DataPanelUpdate(), která hodnoty ze struktury zobrazí ve spodní části UI.

Chce-li uživatel vynulovat hodnoty zadané do rozhraní je nejjednodušším způsobem kliknout na "Load Default", které volá metodu LoadDefaultClicked() a ta už jen zavolá na začátku zmíněnou InitializeDefaultData().

Při zavření okna Unity volá funkci OnApplicationQuit(), která zajistí uložení posledních zadaných hodnot.

V rámci rozšiřování a aktualizacích uživatelského rozhraní zůstaly v kódu také metody ButtonEnableClicked() až HomingClicked(), které do LogTextu přidají text přiřazený funkci, kterou zastávájí (LogTextUpdate()) a následně aktualizují data ve struktuře (StructUpdate(string)). StructUpdate(string) kromě uložení nových dat v závislosti na zvoleném pohonu rovnou také volá metodu MessageOutCombineAndSend().

Další nevyužitou metodou je je DriverDataToLog(), která jen vypisovala poslední přijatá data z Arduina do Logu.



Obr. 18: Zjednodušený vývojový diagram normálního chodu aplikace

Datové struktury a další části kódu

Aby odesílaná a přijímaná data vyhovovala komunikaci a nedocházelo k problémům s jejich formátem, obsahuje jak Unity tak kód pro Arduino totožné struktury. Struktury PcData v sobě uchovávají data odesílaná z Unity, struktury DriverData pak data přijímaná. Každá struktura má dvě varianty, což odpovídá použití dvou Arduin s odlišným počtem připojených driverů.

Aby aplikaci mohl využívat i uživatel, který není dokonale zasvěcen do problematiky, musí být uživatelské rozhraní pokud možno naprosto intuitivní a musí uživatele navést ke správnému postupu, popř. mu dát najevo, co dělá špatně.

Pro správu prvků UI existuje v aplikaci několik skriptů, v první řadě DropdownValueChanged, která upravuje aktivní prvky UI v závislosti na uživatelem zvoleném režimu v rolovací nabídce. U režimu StayStill například není nutné, aby uživatel zadával jakékoli hodnoty, proto jsou všechna okna pro zadávání textu deaktivována a uživatel nemá přístup k úpravě hodnot.

Pro uživatele méně viditelné změny provádí SlidersBehaviour, který obstarává chování sliderů. S využitím metody Update(), která je volána každý snímek, aktualizuje hodnoty mezí pro jednotlivé hodnoty sliderů a zároveň hlídá hodnotu nastavenou uživatelem. Ta je zobrazena vedle slideru a zároveň zapsána do zadávacího okna.

Poslední součástí aplikace je SaveLoad, jehož název je samovysvětlující. Stará se o to, aby data z UI byla při ukončení aplikace správně uložena a při opětovném zapnutí zase načtena.

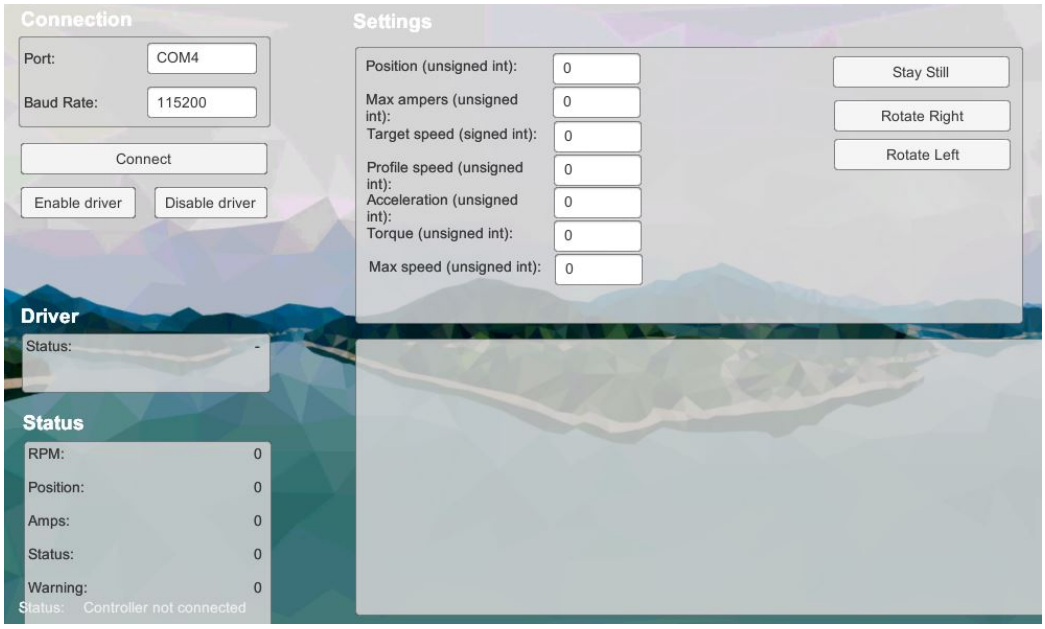
6.3 UŽIVATELSKÉ ROZHRANÍ

Jelikož v rámci projektu bylo nutné otestovat jednotlivé funkcionality pohonů, vzniklo uživatelské rozhraní, které zjednodušuje zasílání zpráv pohonu a zobrazování přijímaných dat.

V průběhu testování bylo nutné uživatelské rozhraní v Unity upravovat, doplňovat a redukovat. První varianta viz obrázek 19 obsahovala pouze tři příkazy - rotaci vlevo, rotaci vpravo a bez rotace. Důležitou součástí je také panel vlevo nahoře, který zprostředkuje připojení Arduina včetně nastavení modulační rychlosti a výběru portu, ke kterému je připojeno.

Při stisku tlačítka s příkazem dojde ke složení dat do zprávy včetně přiřazení příkazu na základě stisknutého tlačítka a následně k jejímu odeslání.

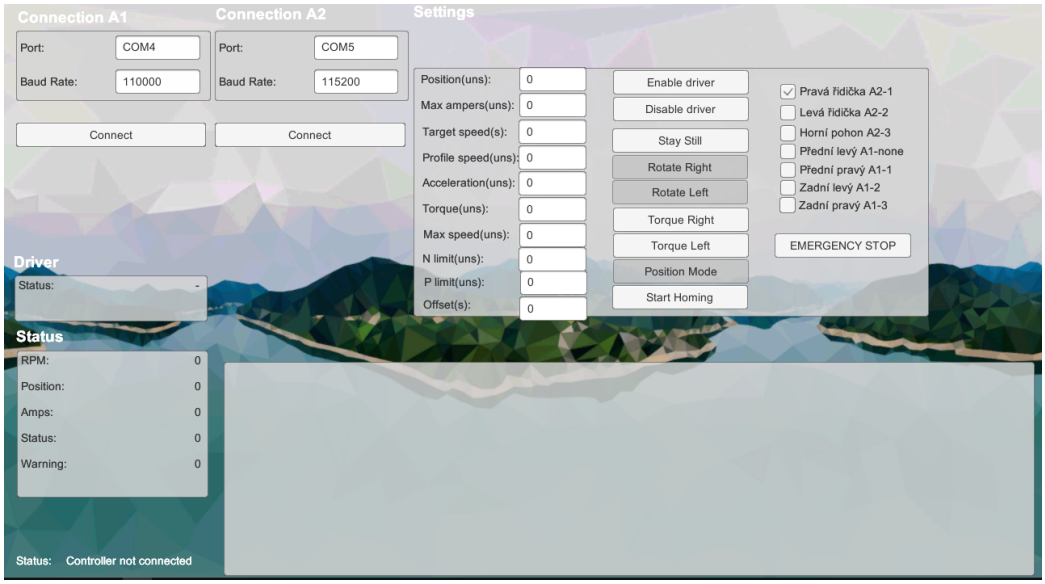
Tato varianta umožňuje testování komunikace s jedním Arduinem, ke kterému je připojen právě jeden driver, což poskytlo první zpětnou vazbu v ohledu konzistence a čitelnosti posílaných dat.



Obr. 19: První verze uživatelského rozhraní pro testování komunikace Arduino Unity

Mezi prvními úpravami byla možnost připojení dvou Arduin viz obrázek 20. Rozšířil se také seznam zasílaných dat a přibyly nové režimy, z hlediska řidiček jsou nejzajímavější ty momentové (Torque).

Do rozhraní přibyla možnost přepínači vybírat řízený motor, což umožnilo dále testovat konzistenci přenášených dat i při připojení více driverů k Arduinu.

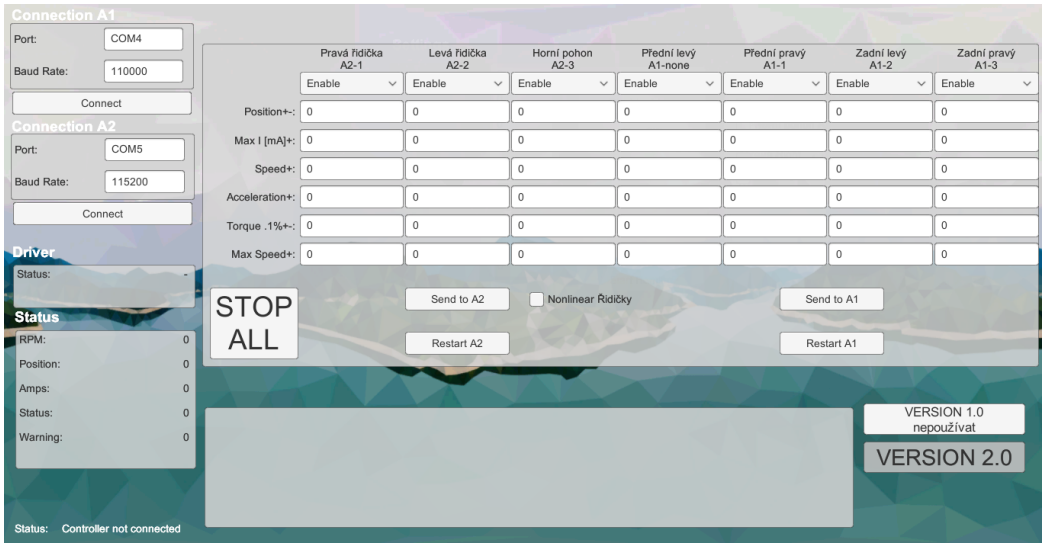


Obr. 20: Upravená první verze UI pro testování komunikace Arduino Unity

Nevýhodou poslední verze bylo to, že umožňuje ovládání právě jednoho motoru, pro další testování však bylo nezbytné ovládat více motorů zároveň. Proto je druhá verze (viz obrázek 21) doplněna o velké množství zadávacích kolonek, do kterých se zadávají hodnoty pro všechny motory zároveň. Na rozdíl od předchozí verze se již režim běhu motoru nevybírá tlačítkem, ale je nutné jej zvolit v posuvné nabídce nad hodnotami (viz obrázek 22). K samotnému zaslání dat pak dojde stiskem tlačítka "Send to"korespondujícím s připojeným Arduinem.

Ve verzi 2 došlo také k redukci zasílaných dat. Dospěli jsme k závěru, že není nutné některé hodnoty posílat při každé změně hodnot, proto byly limity a offset odstraněny a jejich hodnoty se budou měnit přímo v kódu pro Arduino. Hodnota Target speed, která je určena pro rychlostní režim, byla úplně vyřazena a hodnota Profile speed byla přejmenována na Speed. Změnil se také počet režimů, jelikož pro naši aplikaci se rychlostní režim nehodil. Přibyl pak režim pro nastavení konkrétní pozice jako nulové.

Kvůli jednoduššímu řešení chybových stavů Arduina a driverů byla přidána tlačítka pro restartování Arduin a možnost smazat chybové hlášky v jednotlivých driverech.



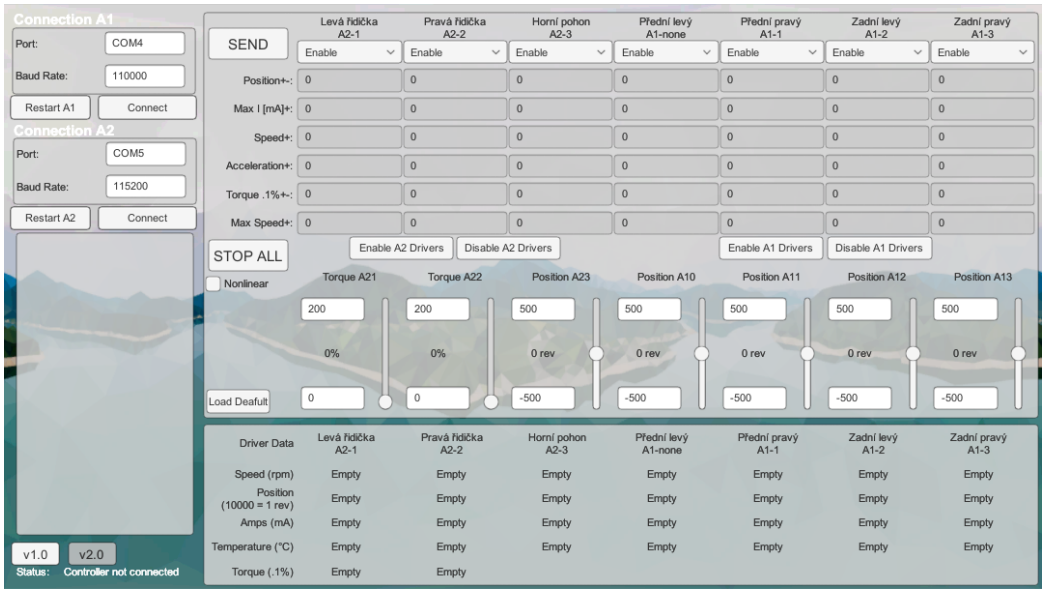
Obr. 21: Druhá verze UI

Obr. 22: Druhá verze UI - posuvná nabídka

Ani tato verze se však nevyhnula úpravám. Bylo nutno přidat okno pro zobrazení přijímaných dat a jezdců pro zadávání vybraných hodnot s proměnnými mezemi. Tyto úpravy si vyžádaly ustoupení okna pro Log na levou stranu a přemístění některých ovládacích prvků. Zároveň přibyla tlačítka pro zapnutí a vypnutí všech driverů připojených ke konkrétnímu Arduinu najednou. V rámci testování a ladění bylo zapínání a vypínání driverů ponecháno i v rolovacích nabídkách.

Tato verze také disponuje ukládáním posledních zadaných hodnot, tudíž při spuštění aplikace se zobrazí hodnoty, které uživatel zadával před vypnutím. Pokud by uživatel chtěl okno dostat do stavu na obrázku 23, je mu k dispozici tlačítko "Load Default".

V rozhraní jsou již také aktivní pouze ta zadávací okna, která jsou relevantní pro zvolený režim daného pohonu.



Obr. 23: Upravená druhá verze UI

7

ZÁVĚR

Prostudoval jsem systém ovládání padákového kluzáku a krizové situace, do kterých se jeho pilot může dostat. V závislosti na těchto informacích jsem navrhoval a upravoval mechanický systém řízení tak, aby byl pocit z manipulace řidičkami co nejblíže skutečnosti. V rámci testování se mnou vybraná topologie ukázala jako efektivní a jednoduché řešení.

Data z některých namontovaných senzorů jsou již úspěšně využívána pro ovládání kluzáku v jednoduché simulaci letu.

Zároveň se podařilo zprovoznit komunikaci mezi počítačem a Arduinem, které již předává přesné pokyny driverům jednotlivých pohonů. Tato komunikace funguje oběma směry, což umožňuje zpřesnit ovládání v simulaci a kontrolu chybových stavů pohonů.

Navržené uživatelské rozhraní pravděpodobně poslouží pouze k testování a ladění řízení pohonů, avšak funkce, které tvoří jeho jádro, v budoucnosti jistě využije firma navrhující komplexní simulaci - Skeleton Software s.r.o.

A

PŘÍLOHA

A.1 MESSAGELISTENER2.CS

Páteř komunikace a základní obsluha prvků UI; dvojče MessageListeneru - obdobný kód, pouze využívá jiné prvky UI a struktur + některé metody se liší počtem vstupních a výstupních proměnných

```
using System;
using UnityEngine;
using UnityEngine.UI;
using static SaveLoad;

//Arduino2 - ridicky a horni pohon

public class MessageListener2 : MonoBehaviour
{
    private Text logText;
    private Text connectionText;

    public Button connectButton2;
    public SerialController serialController2;
    public InputField baudRateInputField2;
    public InputField portInputField2;

    public Button driverEnableButton;
    public Button driverDisableButton;
    public Button stayStillButton;
    public Button rRightButton;
    public Button rLeftButton;
    public Button torqueRightButton;
    public Button torqueLeftButton;
    public Button positionButton;
    public Button emergencyStopButton;
    public Button homingButton;
    //version2.0
    public Button sendToA2Button;
    public Button restartA2Button;
    public Button stopAllButton;
    public Button sendButton;
    public Button loadDefaultButton;
    public Button enableA2Button;
    public Button disableA2Button;

    public InputField targetSpeedField;
```

```
public InputField profileSpeedField;
public InputField profileAccelerationField;
public InputField torqueField;
public InputField maxSpeedField;
public InputField maxAmpersField;
public InputField positionField;
public InputField nLimitField;
public InputField pLimitField;
public InputField offsetField;
//version2.0
public InputField positionA21Field;
public InputField positionA22Field;
public InputField positionA23Field;
public InputField maxMAmpersA21Field;
public InputField maxMAmpersA22Field;
public InputField maxMAmpersA23Field;
public InputField speedA21Field;
public InputField speedA22Field;
public InputField speedA23Field;
public InputField accelerationA21Field;
public InputField accelerationA22Field;
public InputField accelerationA23Field;
public InputField torqueA21Field;
public InputField torqueA22Field;
public InputField torqueA23Field;
public InputField maxSpeedA21Field;
public InputField maxSpeedA22Field;
public InputField maxSpeedA23Field;

public Toggle rightHandToggle;
public Toggle leftHandToggle;
public Toggle upperServoToggle;
//version2.0
public Toggle nonlinearRidicky;

//version2.0
public Dropdown dropDownA21;
public Dropdown dropDownA22;
public Dropdown dropDownA23;

//version2.0+
public Text rpmA21Text;
public Text positionA21Text;
public Text ampsA21Text;
public Text temperatureA21Text;
public Text torqueA21Text;
public Text rpmA22Text;
public Text positionA22Text;
public Text ampsA22Text;
public Text temperatureA22Text;
public Text torqueA22Text;
public Text rpmA23Text;
public Text positionA23Text;
public Text ampsA23Text;
public Text temperatureA23Text;

public PcData.PcData2 pcDataForA2;
public DriverData.DriverData2 fromA2;
```

```
void Start()
{
    connectionText =
      GameObject.Find("ConnectionText").GetComponent<Text>();
    logText = GameObject.Find("LogText").GetComponent<Text>();

    targetSpeedField.text = profileSpeedField.text =
      profileAccelerationField.text = torqueField.text =
      maxSpeedField.text = positionField.text =
      maxAmpersField.text = nLimitField.text =
      pLimitField.text =  offsetField.text = "0";

    LoadLatestData();

    connectButton2.onClick.AddListener(Connect2Clicked);
    driverEnableButton.onClick.AddListener(ButtonEnableClicked);
    driverDisableButton.onClick.AddListener(ButtonDisableClicked);
    stayStillButton.onClick.AddListener(StayStillClicked);
    rRightButton.onClick.AddListener(RotateRightClicked);
    rLeftButton.onClick.AddListener(RotateLeftClicked);
    torqueRightButton.onClick.AddListener(TorqueRightClicked);
    torqueLeftButton.onClick.AddListener(TorqueLeftClicked);
    positionButton.onClick.AddListener(PositionClicked);
    emergencyStopButton.onClick.AddListener(EmergencyStopClicked);
    homingButton.onClick.AddListener(HomingClicked);

    //version2.0
    sendToA2Button.onClick.AddListener(SendA2Clicked);
    restartA2Button.onClick.AddListener(ResetA2Clicked);
    stopAllButton.onClick.AddListener(StopAllClicked);
    sendButton.onClick.AddListener(SendClicked);
    loadDefaultButton.onClick.AddListener(LoadDefaultClicked);
    enableA2Button.onClick.AddListener(EnableA2Clicked);
    disableA2Button.onClick.AddListener(DisableA2Clicked);
}

void LoadLatestData()
{
    string[] newData = LoadData2();
    if (newData == null)
    {
        InitializeDefaultData();
    }
    else
    {
        int i = 0;
        portInputField2.text = newData[i++];
        baudRateInputField2.text = newData[i++];
        positionA21Field.text = newData[i++];
        positionA22Field.text = newData[i++];
        positionA23Field.text = newData[i++];
        maxMAmpersA21Field.text = newData[i++];
        maxMAmpersA22Field.text = newData[i++];
        maxMAmpersA23Field.text = newData[i++];
        speedA21Field.text = newData[i++];
        speedA22Field.text = newData[i++];
        speedA23Field.text = newData[i++];
        accelerationA21Field.text = newData[i++];
        accelerationA22Field.text = newData[i++];
```

```
        accelerationA23Field.text = newData[i++];
        torqueA21Field.text = newData[i++];
        torqueA22Field.text = newData[i++];
        torqueA23Field.text = newData[i++];
        maxSpeedA21Field.text = newData[i++];
        maxSpeedA22Field.text = newData[i++];
        maxSpeedA23Field.text = newData[i];
    }
}

void InitializeDefaultData()
{
    if ((Application.platform == RuntimePlatform.WindowsEditor) ||
        (Application.platform == RuntimePlatform.WindowsPlayer))
    {
        portInputField2.text = "COM5";
    }
    else if ((Application.platform == RuntimePlatform.LinuxEditor) ||
             (Application.platform == RuntimePlatform.LinuxPlayer))
    {
        portInputField2.text = "/dev/ttyACM0";
    }

    baudRateInputField2.text = "115200";
    serialController2.enabled = false;

    //version2.0
    positionA21Field.text = positionA22Field.text =
      positionA23Field.text = maxMAmpersA21Field.text =
      maxMAmpersA22Field.text = maxMAmpersA23Field.text =
      speedA21Field.text = speedA22Field.text =
      speedA23Field.text = accelerationA21Field.text =
      accelerationA22Field.text = accelerationA23Field.text =
      torqueA21Field.text = torqueA22Field.text =
      torqueA23Field.text = maxSpeedA21Field.text =
      maxSpeedA22Field.text = maxSpeedA23Field.text = "0";
}

void LoadDefaultClicked()
{
    InitializeDefaultData();
}

private void OnApplicationQuit()
{
    string[] toSave = new string[20];
    int i = 0;
    toSave[i++] = portInputField2.text;
    toSave[i++] = baudRateInputField2.text;
    toSave[i++] = positionA21Field.text;
    toSave[i++] = positionA22Field.text;
    toSave[i++] = positionA23Field.text;
    toSave[i++] = maxMAmpersA21Field.text;
    toSave[i++] = maxMAmpersA22Field.text;
    toSave[i++] = maxMAmpersA23Field.text;
    toSave[i++] = speedA21Field.text;
    toSave[i++] = speedA22Field.text;
    toSave[i++] = speedA23Field.text;
    toSave[i++] = accelerationA21Field.text;
```

```
    toSave[i++] = accelerationA22Field.text;
    toSave[i++] = accelerationA23Field.text;
    toSave[i++] = torqueA21Field.text;
    toSave[i++] = torqueA22Field.text;
    toSave[i++] = torqueA23Field.text;
    toSave[i++] = maxSpeedA21Field.text;
    toSave[i++] = maxSpeedA22Field.text;
    toSave[i] = maxSpeedA23Field.text;
    SaveData2(toSave);
}

void EnableA2Clicked()
{
    StructUpdate(1, 1, 1);
}

void DisableA2Clicked()
{
    StructUpdate(2, 2, 2);
}

void SendClicked()      //version2.0+
{
    SendA2Clicked();
}

void SendA2Clicked()    //version2.0
{
    byte command1, command2, command3;
    command1 = DropdownTranslate(dropDownA21.value);
    command2 = DropdownTranslate(dropDownA22.value);
    command3 = DropdownTranslate(dropDownA23.value);
    StructUpdate(command1, command2, command3);
}

void ResetA2Clicked()   //version2.0
{
    StructUpdate(10, 10, 10);
}

void StopAllClicked()   //version2.0
{
    StructUpdate(7, 7, 7);
}

byte DropdownTranslate(int value)
{
    switch (value)
    {
        case 0://enable
        {
            return 1;
        }
        case 1://disable
        {
            return 2;
        }
        case 2://stayStill
        {
```

```
            return 3;
        }
        case 3://torqueRight
        {
            return 4;
        }
        case 4://torqueLeft
        {
            return 5;
        }
        case 5://position
        {
            return 6;
        }
        case 6://startHoming
        {
            return 8;
        }
        case 7://setZeroPosition
        {
            return 9;
        }
        case 8://resetDriverErrors
        {
            return 11;
        }
    }
    return 3;

    /*switch (value)
    {
        case 0://stayStill
        {
            return 3;
        }
        case 1://torqueRight
        {
            return 4;
        }
        case 2://torqueLeft
        {
            return 5;
        }
        case 3://position
        {
            return 6;
        }
        case 4://startHoming
        {
            return 8;
        }
        case 5://setZeroPosition
        {
            return 9;
        }
        case 6://resetDriverErrors
        {
            return 11;
        }
```

```
    }
    return 3;*/
}

void Connect2Clicked()
{
    InputField port2 =
      GameObject.Find("PortInputField2").GetComponent<InputField>();
    InputField baudRate2 = GameObject.Find("BaudRateInputField2")
      .GetComponent<InputField>();
    serialController2.portName = port2.text;
    serialController2.baudRate = int.Parse(baudRate2.text);

    if (serialController2.enabled)
    {
        serialController2.enabled = false;
        connectButton2.GetComponentInChildren<Text>().text =
          "Connect";
    }
    else
    {
        serialController2.enabled = true;
        connectButton2.GetComponentInChildren<Text>().text =
          "Disconnect";
    }
}

void ButtonEnableClicked()
{
    LogTextUpdate("Enable");
    StructUpdate("1");
}

void ButtonDisableClicked()
{
    LogTextUpdate("Disable");
    StructUpdate("2");
}

void StayStillClicked()
{
    LogTextUpdate("Stay Still");
    StructUpdate("3");
}

void RotateRightClicked()
{
    LogTextUpdate("Rotate Right");
    StructUpdate("4");
}

void RotateLeftClicked()
{
    LogTextUpdate("Rotate Left");
    StructUpdate("5");
}

void TorqueRightClicked()
{
```

```
    LogTextUpdate("Torque mode - Rightwise");
    StructUpdate("6");
}

void TorqueLeftClicked()
{
    LogTextUpdate("Torque mode - Leftwise");
    StructUpdate("7");
}

void PositionClicked()
{
    LogTextUpdate("Position mode");
    StructUpdate("8");
}

void EmergencyStopClicked()
{
    LogTextUpdate("EMERGENCY STOP");
    StructUpdate("9");
}

void HomingClicked()
{
    LogTextUpdate("Homing");
    StructUpdate("10");
}

void StructUpdate(byte command1, byte command2, byte command3)
{
    //version2.0
    pcDataForA2.position1 = Int32.Parse(positionA21Field.text);
    pcDataForA2.amps1 = UInt16.Parse(maxMAmpersA21Field.text);
    pcDataForA2.rpm_profile1 = UInt32.Parse(speedA21Field.text);
    pcDataForA2.rps1 = UInt32.Parse(accelerationA21Field.text);
    pcDataForA2.torque1 = Int16.Parse(torqueA21Field.text);
    pcDataForA2.max_rpm1 = UInt32.Parse(maxSpeedA21Field.text);
    pcDataForA2.mode_command1 = command1;

    pcDataForA2.position2 = Int32.Parse(positionA22Field.text);
    pcDataForA2.amps2 = UInt16.Parse(maxMAmpersA22Field.text);
    pcDataForA2.rpm_profile2 = UInt32.Parse(speedA22Field.text);
    pcDataForA2.rps2 = UInt32.Parse(accelerationA22Field.text);
    pcDataForA2.torque2 = Int16.Parse(torqueA22Field.text);
    pcDataForA2.max_rpm2 = UInt32.Parse(maxSpeedA22Field.text);
    pcDataForA2.mode_command2 = command2;

    pcDataForA2.position3 = Int32.Parse(positionA23Field.text);
    pcDataForA2.amps3 = UInt16.Parse(maxMAmpersA23Field.text);
    pcDataForA2.rpm_profile3 = UInt32.Parse(speedA23Field.text);
    pcDataForA2.rps3 = UInt32.Parse(accelerationA23Field.text);
    pcDataForA2.torque3 = Int16.Parse(torqueA23Field.text);
    pcDataForA2.max_rpm3 = UInt32.Parse(maxSpeedA23Field.text);
    pcDataForA2.mode_command3 = command3;
    pcDataForA2.send_back = 1;
    if (nonlinearRidicky.isOn)
    {
        pcDataForA2.nonlinear_ridicky = 1;
    }
```

```
    else
    {
        pcDataForA2.nonlinear_ridicky = 0;
    }
    MessageOutCombineAndSend();
}

void StructUpdate(string command)
{
    //version1.0
    /*if (rightHandToggle.isOn)
    {
        pcDataForA2.position1 = Int32.Parse(positionField.text);
        pcDataForA2.amps1 = UInt16.Parse(maxAmpersField.text);
        pcDataForA2.rpm1 = Int32.Parse(targetSpeedField.text);
        pcDataForA2.rpm_profile1=UInt32.Parse(profileSpeedField.text);
        pcDataForA2.rps1 = UInt32.Parse(profileAccelerationField.text);
        pcDataForA2.torque1 = Int16.Parse(torqueField.text);
        pcDataForA2.max_rpm1 = UInt32.Parse(maxSpeedField.text);
        pcDataForA2.n_limit1 = Int32.Parse(nLimitField.text);
        pcDataForA2.p_limit1 = Int32.Parse(pLimitField.text);
        pcDataForA2.offset1 = Int32.Parse(offsetField.text);
        pcDataForA2.mode_command1 = byte.Parse(command);
        MessageOutCombineAndSend();
    } else if (leftHandToggle.isOn)
    {
        pcDataForA2.position2 = Int32.Parse(positionField.text);
        pcDataForA2.amps2 = UInt16.Parse(maxAmpersField.text);
        pcDataForA2.rpm2 = Int32.Parse(targetSpeedField.text);
        pcDataForA2.rpm_profile2=UInt32.Parse(profileSpeedField.text);
        pcDataForA2.rps2 = UInt32.Parse(profileAccelerationField.text);
        pcDataForA2.torque2 = Int16.Parse(torqueField.text);
        pcDataForA2.max_rpm2 = UInt32.Parse(maxSpeedField.text);
        pcDataForA2.n_limit2 = Int32.Parse(nLimitField.text);
        pcDataForA2.p_limit2 = Int32.Parse(pLimitField.text);
        pcDataForA2.offset2 = Int32.Parse(offsetField.text);
        pcDataForA2.mode_command2 = byte.Parse(command);
        MessageOutCombineAndSend();
    }else if (upperServoToggle.isOn)
    {
        pcDataForA2.position3 = Int32.Parse(positionField.text);
        pcDataForA2.amps3 = UInt16.Parse(maxAmpersField.text);
        pcDataForA2.rpm3 = Int32.Parse(targetSpeedField.text);
        pcDataForA2.rpm_profile3=UInt32.Parse(profileSpeedField.text);
        pcDataForA2.rps3 = UInt32.Parse(profileAccelerationField.text);
        pcDataForA2.torque3 = Int16.Parse(torqueField.text);
        pcDataForA2.max_rpm3 = UInt32.Parse(maxSpeedField.text);
        pcDataForA2.n_limit3 = Int32.Parse(nLimitField.text);
        pcDataForA2.p_limit3 = Int32.Parse(pLimitField.text);
        pcDataForA2.offset3 = Int32.Parse(offsetField.text);
        pcDataForA2.mode_command3 = byte.Parse(command);
        MessageOutCombineAndSend();
    }*/
}

void MessageOutCombineAndSend()
{
    //version1.0
    /*pcDataForA2.send_back = 1;
```

```
if (serialController2.enabled)
{
    byte[] message =
      AppendArrays(GetByteArray(pcDataForA2.position1, 4),
      GetByteArray(pcDataForA2.amps1, 2));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rpm1, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rpm_profile1, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rps1, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.torque1, 2));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.max_rpm1, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.n_limit1, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.p_limit1, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.offset1, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.mode_command1));
    //message = AppendArrays(message, GetByteArray(
      pcDataForA2.send_back1));

    message = AppendArrays(message, GetByteArray(
      pcDataForA2.position2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.amps2, 2));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rpm2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rpm_profile2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rps2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.torque2, 2));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.max_rpm2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.n_limit2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.p_limit2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.offset2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.mode_command2));
    //message = AppendArrays(message, GetByteArray(
      pcDataForA2.send_back1));

    message = AppendArrays(message, GetByteArray(
      pcDataForA2.position3, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.amps3, 2));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rpm3, 4));
    message = AppendArrays(message, GetByteArray(
```

```
      pcDataForA2.rpm_profile3, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rps3, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.torque3, 2));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.max_rpm3, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.n_limit3, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.p_limit3, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.offset3, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.mode_command3));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.send_back));

    serialController2.SendSerialMessage(message);
}
else
{
    logText.text = "A2 not connected!\n" + logText.text;
}*/
//version2.0
if (serialController2.enabled)
{
    byte[] message =
      AppendArrays(GetByteArray(pcDataForA2.position1, 4),
      GetByteArray(pcDataForA2.amps1, 2));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rpm_profile1, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rps1, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.torque1, 2));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.max_rpm1, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.mode_command1));

    message = AppendArrays(message, GetByteArray(
      pcDataForA2.position2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.amps2, 2));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rpm_profile2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.rps2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.torque2, 2));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.max_rpm2, 4));
    message = AppendArrays(message, GetByteArray(
      pcDataForA2.mode_command2));

    message = AppendArrays(message, GetByteArray(
      pcDataForA2.position3, 4));
    message = AppendArrays(message, GetByteArray(
```

```
            pcDataForA2.amps3, 2));
        message = AppendArrays(message, GetByteArray(
            pcDataForA2.rpm_profile3, 4));
        message = AppendArrays(message, GetByteArray(
            pcDataForA2.rps3, 4));
        message = AppendArrays(message, GetByteArray(
            pcDataForA2.torque3, 2));
        message = AppendArrays(message, GetByteArray(
            pcDataForA2.max_rpm3, 4));
        message = AppendArrays(message, GetByteArray(
            pcDataForA2.mode_command3));
        message = AppendArrays(message, GetByteArray(
            pcDataForA2.send_back));
        message = AppendArrays(message, GetByteArray(
            pcDataForA2.nonlinear_ridicky));

        serialController2.SendSerialMessage(message);
    }
    else
    {
        logText.text = "A2 not connected!\n" + logText.text;
    }
}

byte[] AppendArrays(byte[] A, byte[] B)
{
    byte[] result = new byte[A.Length + B.Length];
    Buffer.BlockCopy(A, 0, result, 0, A.Length);
    Buffer.BlockCopy(B, 0, result, A.Length, B.Length);
    return result;
}

byte[] GetByteArray(int myValue, int myCount)
{
    byte[] bArray = new byte[myCount];
    for (int i = 0; i < myCount; i++)
    {
        bArray[i] = 0;
    }
    bArray = BitConverter.GetBytes(myValue);
    if (myCount == 2)
    {
        byte[] bArray2 = new byte[2];
        bArray2[0] = bArray[0];
        bArray2[1] = bArray[1];
        return bArray2;
    }
    return bArray;
}

byte[] GetByteArray(uint myValue, int myCount)
{
    byte[] bArray = new byte[myCount];
    for (int i = 0; i < myCount; i++)
    {
        bArray[i] = 0;
    }
    bArray = BitConverter.GetBytes(myValue);
    if (myCount == 2)
```

```
    {
        byte[] bArray2 = new byte[2];
        bArray2[0] = bArray[0];
        bArray2[1] = bArray[1];
        return bArray2;
    }
    return bArray;
}

byte[] GetByteArray(byte myValue)
{
    byte[] bArray = new byte[1];
    bArray[0] = 0;
    bArray = BitConverter.GetBytes(myValue);
    byte[] bArray2 = new byte[1];
    bArray2[0] = bArray[0];
    return bArray2;
}

void OnMessageArrived(string msg)
{
    string[] submsg = msg.Split(';');
    if (submsg[0] == "162")//162 = 0xA2
    {
        logText.text = "Incoming serial message A2: " + msg +
          "\n" + logText.text;
        if (submsg.Length > 0)
        {
            int i = 1;
            fromA2.current_rpm1 = Int32.Parse(submsg[i]);
            i++;
            fromA2.current_position1 = Int32.Parse(submsg[i]);
            i++;
            fromA2.current_amps1 = Int16.Parse(submsg[i]);
            i++;
            fromA2.device_temperature1 = Int16.Parse(submsg[i]);
            i++;
            fromA2.current_torque1 = Int16.Parse(submsg[i]);
            i++;

            fromA2.current_rpm2 = Int32.Parse(submsg[i]);
            i++;
            fromA2.current_position2 = Int32.Parse(submsg[i]);
            i++;
            fromA2.current_amps2 = Int16.Parse(submsg[i]);
            i++;
            fromA2.device_temperature2 = Int16.Parse(submsg[i]);
            i++;
            fromA2.current_torque2 = Int16.Parse(submsg[i]);
            i++;

            fromA2.current_rpm3 = Int32.Parse(submsg[i]);
            i++;
            fromA2.current_position3 = Int32.Parse(submsg[i]);
            i++;
            fromA2.current_amps3 = Int16.Parse(submsg[i]);
            i++;
            fromA2.device_temperature3 = Int16.Parse(submsg[i]);
```

```
            //DriverDataToLog();
            DataPanelUpdate();
        }
    }
}

void DataPanelUpdate()      //version2.0+
{
    rpmA21Text.text = fromA2.current_rpm1.ToString();
    positionA21Text.text = fromA2.current_position1.ToString();
    ampsA21Text.text = fromA2.current_amps1.ToString();
    temperatureA21Text.text = fromA2.device_temperature1.ToString();
    torqueA21Text.text = fromA2.current_torque1.ToString();

    rpmA22Text.text = fromA2.current_rpm2.ToString();
    positionA22Text.text = fromA2.current_position2.ToString();
    ampsA22Text.text = fromA2.current_amps2.ToString();
    temperatureA22Text.text = fromA2.device_temperature2.ToString();
    torqueA22Text.text = fromA2.current_torque2.ToString();

    rpmA23Text.text = fromA2.current_rpm3.ToString();
    positionA23Text.text = fromA2.current_position3.ToString();
    ampsA23Text.text = fromA2.current_amps3.ToString();
    temperatureA23Text.text = fromA2.device_temperature3.ToString();
}

void LogTextUpdate(string message)
{
    if (rightHandToggle.isOn || leftHandToggle.isOn ||
        upperServoToggle.isOn)
    {
        logText.text = message + "\n" + logText.text;
    }
}

void DriverDataToLog()
{
    logText.text =
      "A2-1: "+ fromA2.current_rpm1 +"; + fromA2.current_amps1 +";"+
        fromA2.current_position1 +";"+ fromA2.device_temperature1
        +";"+ fromA2.current_torque1 +"\n"+
      "A2-2: "+ fromA2.current_rpm2 +";"+ fromA2.current_amps2 +";"+
        fromA2.current_position2 +";"+ fromA2.device_temperature2
        +";"+ fromA2.current_torque2 +"\n"+
      "A2-3: "+ fromA2.current_rpm3 +";"+ fromA2.current_amps3 +";"+
        fromA2.current_position3 +";"+
        fromA2.device_temperature3 +"\n"+ logText.text;
}

void OnConnectionEvent(bool success)
{
    Debug.Log("success = " + success);

    if (success)
    {
        connectionText.text = "Connection established";
        Debug.Log("Connection established");
    }
    else
```

```
        {
            connectionText.text = "Connection attempt failed
              or disconnection detected";
            Debug.Log("Connection attempt failed or
              disconnection detected");
        }
    }
}
```

A.2 PCDATA.CS

Struktura obsahující data posílaná z počítače do arduina

```
using System;

namespace PcData
{

    public struct PcData1
    {
        public Int32 position;
        public UInt16 amps;
        public UInt32 rpm_profile;
        public UInt32 rps;
        public Int16 torque;
        public UInt32 max_rpm;
        public byte mode_command;

        public Int32 position1;
        public UInt16 amps1;
        public UInt32 rpm_profile1;
        public UInt32 rps1;
        public Int16 torque1;
        public UInt32 max_rpm1;
        public byte mode_command1;

        public Int32 position2;
        public UInt16 amps2;
        public UInt32 rpm_profile2;
        public UInt32 rps2;
        public Int16 torque2;
        public UInt32 max_rpm2;
        public byte mode_command2;

        public Int32 position3;
        public UInt16 amps3;
        public UInt32 rpm_profile3;
        public UInt32 rps3;
        public Int16 torque3;
        public UInt32 max_rpm3;
        public byte mode_command3;
        public byte send_back;
    }

    public struct PcData2
    {
        public Int32 position1;
```

```
        public UInt16 amps1;
        public UInt32 rpm_profile1;
        public UInt32 rps1;
        public Int16 torque1;
        public UInt32 max_rpm1;
        public byte mode_command1;

        public Int32 position2;
        public UInt16 amps2;
        public UInt32 rpm_profile2;
        public UInt32 rps2;
        public Int16 torque2;
        public UInt32 max_rpm2;
        public byte mode_command2;

        public Int32 position3;
        public UInt16 amps3;
        public UInt32 rpm_profile3;
        public UInt32 rps3;
        public Int16 torque3;
        public UInt32 max_rpm3;
        public byte mode_command3;

        public byte send_back;
        public byte nonlinear_ridicky;
    }
}
```

A.3 DRIVERDATA.CS

Struktura obsahující data posílaná z arduina do počítače

```
using System;

namespace DriverData
{
    public struct DriverData1
    {
        public Int32 current_rpm;
        public Int32 current_position;
        public Int16 current_amps;
        public Int16 device_temperature;

        public Int32 current_rpm1;
        public Int32 current_position1;
        public Int16 current_amps1;
        public Int16 device_temperature1;

        public Int32 current_rpm2;
        public Int32 current_position2;
        public Int16 current_amps2;
        public Int16 device_temperature2;

        public Int32 current_rpm3;
        public Int32 current_position3;
        public Int16 current_amps3;
        public Int16 device_temperature3;
```

```
    }

    public struct DriverData2
    {
        public Int32 current_rpm1;
        public Int32 current_position1;
        public Int16 current_amps1;
        public Int16 device_temperature1;
        public Int16 current_torque1;    //NEW

        public Int32 current_rpm2;
        public Int32 current_position2;
        public Int16 current_amps2;
        public Int16 device_temperature2;
        public Int16 current_torque2;    //NEW

        public Int32 current_rpm3;
        public Int32 current_position3;
        public Int16 current_amps3;
        public Int16 device_temperature3;
    }
}
```

A.4 SLIDERSBEHAVIOUR.CS

Třída spravující chování sliderů - Kód je umístěn v repozitáři projektu. [7]

A.5 SAVELOAD.CS

Třída spravující ukládání a načítání dat vyplněných v UI [7]

A.6 DROPDOWNVALUECHANGED.CS

Třída spravující chování UI v závislosti na hodnotě vybrané v Dropdownu [7]

LITERATURA

[1] PLOS Richard. *Paragliding: moderní učebnice létání s padákovými kluzáky.* 4. vyd. Cheb: Svět křídel, 2008. ISBN 978-80-86808-47-5.

[2] TSAROV Nikolay a Nikolay YOTOV. Aerodynamics theory for beginners paragliding pilots. *Skynomad* [online]. 2007 [cit. 2021-5-26]. Dostupné z: http://skynomad.com/articles/beginners-aerodynamics.html.

[3] KINCO AUTOMATION. *Servo System Catalog* [online]. [cit. 2021-5-26]. Dostupné z: https://en.kinco.cn/download/d_encatalog/servo/kincocatalog_cd3fd3_k1c15_190719.pdf.

[4] SHOWA MEASURING INSTRUMENTS INC. *Datasheet CZL601* [online]. 2010 [cit. 2021-5-26]. Dostupné z: http://www.cwmelettronica.com/wp-content/uploads/2016/09/czl601-brochure.pdf.

[5] AVIA SEMICONDUCTOR. *HX711 Datasheet* [online]. [cit. 2021-5-26]. Dostupné z: https://cdn.sparkfun.com/assets/b/f/5/a/e/hx711f_en.pdf.

[6] WILCHES Daniel. *Ardity* [online]. [cit. 2021-5-26]. Dostupné z: https://ardity.dwilches.com/.

[7] KARBAN Pavel a Petr KROPÍK. *Repozitář projektu FlyOnVision GUI* [online]. [cit. 2021-5-26]. Dostupné z: https://gitlab.fel.zcu.cz/parasim/control-gui/-/tree/master.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK