Západočeská univerzita v Plzni

Fakulta elektrotechnická
Katedra elektroniky a informačních technologií

Bakalářská Práce

Monitoring polohy sedačky simulátoru padákového kluzáku

Autor práce: **Lukáš Sládek**
Vedoucí práce: **Ing. Petr Kropík, Ph.D.**

2023

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Lukáš SLÁDEK**
Osobní číslo:	**E19B0113P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Téma práce:	**Monitoring polohy sedačky simulátoru padákového kluzáku**
Zadávající katedra:	**Katedra elektroniky a informačních technologií**

Zásady pro vypracování

1. Prostudujte základní principy ovládání simulátoru padákového kluzáku, jeho řídícího systému a servomotorů.
2. Navrhněte vhodnou skladbu a topologii senzorických prvků pro sledování polohy sedačky.
3. Navrhněte a vytvořte vhodné algoritmy a postupy pro výpočet polohy sedačky.
4. Navrhněte a vyberte vhodnou platformu pro agregaci senzorických dat a provádění výpočtu polohy sedačky.
5. Implementujte a experimentálně ověřte navržené řešení na simulátoru padákového kluzáku.

Rozsah bakalářské práce: **30 – 40**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **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: **7. října 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

L.S.

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

Doc. Ing. Jiří Hammerbauer, Ph.D.
vedoucí katedry

V Plzni dne 7. října 2022

Abstrakt

Bakalářská práce je zaměřena na představení řešení snímaní polohy prototypu simulátoru padákového kluzáku a řízení jejího pohybu na základě dat ze simulace v reálném čase. První část je věnována seznámení s reálným padákovým kluzákem. Druhá část je zaměřena na konstrukci simulátoru společně se stručným popisem principu funkce použitých servomotorů a jejich způsobu řízení. Třetí část je věnována VR headsetům použitým při vývoji simulátoru. Ve čtvrté části jsou představeny senzorické prvky určené pro snímání polohy sedačky. Pátá část je zaměřena na koncept řešení polohování sedačky, který je následně aplikován na výpočet výsledné polohy a polohování na základě dat ze simulace. Poslední část je věnována testovacímu uživatelskému rozhraní, na kterém byly otestovány vlastnosti navrženého řešení. Všechny výpočty jsou psané v programovacím jazyce C#. Cílem práce je navrhnout algoritmus pro monitoring a polohování sedačky simulátoru padákového kluzáku.

Klíčová slova

Simulátor, padákový kluzák, lanový robot, servomotor, enkodér, sledování polohy, řízení polohy, virtuální realita, C#

Abstract

The bachelor thesis is focused on the presentation of a solution for sensing the position of a prototype paraglider simulator and controlling its movement based on real-time simulation data. The first part is devoted to the introduction of a real paraglider. The second part is focused on the design of the simulator together with a brief description of the principle of the function of the servomotors used and their control method. The third part is devoted to the VR headsets used in the development of the simulator. The fourth part presents the sensor elements designed to sense the seat position. The fifth section focuses on the concept of the seat positioning solution, which is then applied to calculate the final position and positioning based on the simulation data. The last section is devoted to a test user interface on which the features of the proposed solution were tested. All calculations are written in C# programming language. The aim of this work is to design an algorithm for monitoring and positioning of the paraglider simulator seat.

Keywords

Simulator, paraglider, cable robot, servo motor, encoder, position tracking, position control, virtual reality, C#

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce Ing. Petru Kropíkovi, Ph.D za jeho přístup a cenné rady, které vedly k vypracování této práce. Zároveň bych rád poděkoval kolegovi Richardu Siverovi za jeho pomoc při seznamování se projektem.

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 25.5.2023 Lukáš Sládek

Obsah

Seznam použitých symbolů a zkratek

Značka	**Popis**	**Jednotka**
m	Hmotnost tkaniny	g/m^2
P	Výkon	W
f	Frekvence	Hz
$1f, 3f$	Počet fází	-
U	Napětí	V
$tilt$	Normalizovaný úhel natočení	-
$Distance$	Relativní vzdálenost sedačky a pohonu	cm
$StartDistance$	Výchozí délka lana	cm
$Delta$	Rozdíl relativní a výchozí délky	cm
$MaxPosition$	Rozlišení enkodéru na jednu otáčku	inc
$Position$	Finální délka lana	inc

Seznam obrázků

Úvod

Předmětem této práce je představení řešení snímání polohy sedačky simulátoru padákového kluzáku a řízení jejího pohybu podle simulace v reálném čase. Paragliding je extrémní sport, který s sebou přináší mnoho rizik. Každá instance letu je jiná a pro správné zvládnutí všech požadavků je nutné ovládat základní znalosti meteorologie, aerodynamiky a mechaniky letu padákového kluzáku. Důležitým aspektem letu je také mít přehled o trase letu a znalosti, jak správně navigovat. Mnoho přírodních překážek, na které při letu v padákovém kluzáku pilot narazí, slouží jako kontrolní body, podle kterých se pilot orientuje. Znalost těchto bodů je nedílnou součástí přípravy na samotný let. [1]

Vzhledem k celkové náročnosti a strmé křivce učení všech aspektů letu, byl ve spolupráci Fakulty elektrotechnické Západočeské univerzity v Plzni, Beskydské školy létání a Skeleton Software s.r.o. koncipován pokročilý letecký simulátor padákového kluzáku s podporou virtuální reality Fly On Vision. Jeho úkolem je zvýšit bezpečnost a snížit riziko úrazu pilotů padákových kluzáků všech kategorií. V simulaci se pilot proletí skutečnými lokacemi, které mu umožní se seznámit s trasou letu a termikou daného prostředí. Zároveň mu to umožní se připravit na absolvování identické trasy s reálným padákovým kluzákem.

V úvodní části práce se věnuji přiblížení funkce reálného padákového kluzáku. Dále jsem představil koncept návrhu konstrukce simulátoru, jeho řídicího systému a ovládání. V druhé části se věnuji topologii senzorických prvků, principů jejich fungování a využití pro polohování sedačky s pilotem. Ve finální části se věnuji vývoji samotného programu pro ovládání polohování a experimentálnímu ověření jeho funkčnosti na simulátoru. Výsledky získaných vlastností pohybu jsem následně aplikoval na výpočty polohy s účelem zpřesnění pohybu. Zároveň jsem uvedl možná vylepšení pro budoucí verze prototypu simulátoru padákového kluzáku.

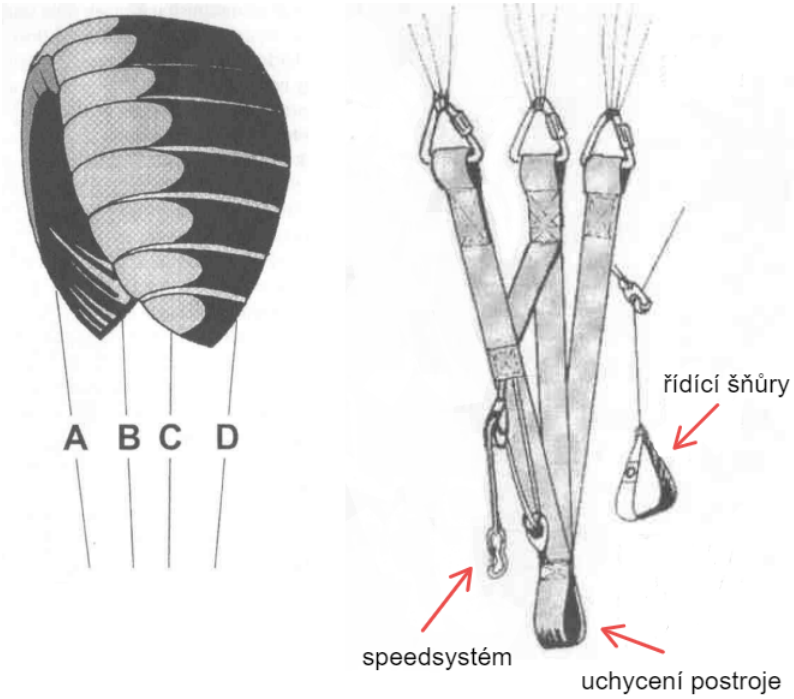
1 Reálný padákový kluzák

Úvodní kapitolu věnuji stručnému seznámení s reálným padákovým kluzákem jako předmětem simulace. Padákový kluzák je zařízení s vyšší hmotností než vzduch, bez pevné vnitřní konstrukce, určené pro sportovní létání. Skládá se z několika hlavních komponentů.

- Vrchlík
- Šňůry a vázání
- Popruhy a posed s pilotem

Obrázek 1: Hlavní části padákového kluzáku.

Pro výrobu vrchlíku se dnes nejčastěji používají vysokopevnostní vlákna na bázi polyamidu, které jsou nám známé pod jejich reklamním označením nylon. Na materiál se nanáší ještě finální chemický zátěr, který zajistí splnění hlavního kritéria a to je vysoká odolnost látky vůči průchodu vzduchu. Mezi další vlastnosti patří vysoká pevnost a nízká hmotnost kolem $45\,\mathrm{g/m^2}$. Vrchlík se skládá ze dvou vrstev potahu propojené strukturálními prvky, které v sobě mají ještě další otvory určené pro vyrovnávání tlaku. Při proudění vzduchu nabývá vrchlík profilu křídla. Křídlo následně při pohybu vzduchem vytváří vztlakovou sílu, díky které je padákový kluzák schopen letu.[1]



Obrázek 2: Šňůry a popruhy padákového kluzáku.[1]

Soustava vázání, která připojuje vrchlík s popruhy, se dá rozdělit dále na tři části: hlavní šňůry, větvení a řídící šňůry. Při pohledu ze strany jsou jednotlivé šňůry rozděleny pomocí písmen A, B, C a případně D (obr.2). Větvení, také označované jako „galerie“, slouží k rovnoměrnému rozprostření sil působící na vrchlík. Můžeme jej vidět pod vrchlíkem na obrázku 1.[1]

Pro propojení sedačky s pilotem a vrchlíku slouží popruhy, tím ale jejich funkčnost nekončí. Zároveň je na nich upevněno řízení, které si stručně rozebereme v další kapitole.[1]

1.1 Řízení

Mezi základní řídící prvky patří řízení pomocí řídicích šňůr a pomocí změny těžiště pilota. Při reálných letových situacích, vlivem dynamických povětrnostních podmínek a dalších aspektů letu, se od pilota očekává řízení pomocí kombinace uvedených prvků.

Řídicí šňůry jsou na každé straně upevněny k odtokové hraně vrchlíku. Slouží jak k zatáčení, tak k brzdění. Když za ně pilot zatáhne, dojde k částečné deformaci zadní části vrchlíku, tím se zvýší její aerodynamický odpor a zpomalí danou stranu vrchlíku natolik, že jí druhá strana předběhne a umožní pilotovi zatočit požadovaným směrem. Z této definice vyplývá, že brzdění se provádí zatažením za obě řídicí šňůry naráz. Řídicí šňůry zároveň působí pilotovi odpor. Tento odpor není lineární, postupně se stažením se zvyšuje a musel se při vývoji řídícího systému simulátoru vzít v potaz.[1]

Dalším ze základních způsobů, jak ovládat padákový kluzák za letu je přesouvání těžiště pomocí naklánění pilota. Při letu, když je těžiště vycentrováno se svislou osou padákového kluzáku, působí na obě strany vrchlíku stejné vztlakové síly za předpokladu, že zrovna pilot nezatáčí. Když se však pilot nahne do strany, napne tím šňůry na dané straně a lehce povolí šňůry na straně druhé. Tím budou na stranách vrchlíku působit rozdílně vztlakové síly a padákový kluzák bude zatáčet. Tento efekt je dobře vidět na obrázku 3.[1][2]

Obrázek 3: Efekt řízení pomocí změny těžiště.

2 Základní popis simulátoru

V této kapitole se zabývám přiblížením funkce simulátoru padákového kluzáku. Proberu jeho ovládání, návrh konstrukce, zvolené servomotory, důvody jejich výběru a jejich funkci společně s řídicím systémem.

2.1 Konstrukce a její návrh

Kompletní jednotka simulátor padákového kluzáku se skládá z pěti hlavních částí.

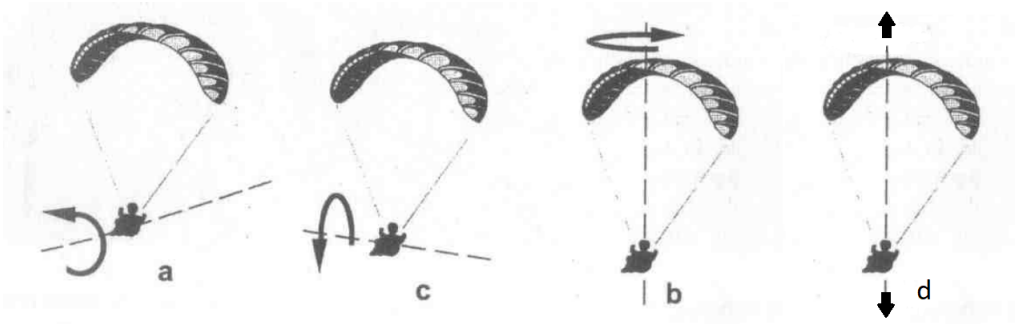
- Rám konstrukce se servo motory
- Zavěšení sedačky s pilotem
- Rozvaděč s řídícím systémem
- Výpočetní jednotka pro provádění simulace
- HeadSet s virtuální realitou

Obrázek 4: Konstrukce simulátoru padákového kluzáku.

Úkolem simulátoru je snaha přiblížit se co nejvíce realitě, což není vždy nejjednodušší úkol. Speciálně, když se jedná o simulaci vysoce dynamického pohybu, jako je let padákového kluzáku. Přesně replikovat danou motoriku je velmi náročné, ale do značné míry je možné se jí přiblížit a velkou roli v tom hraje návrh konstrukce, správné umístění prvků pohonu a způsob zavěšení sedačky.

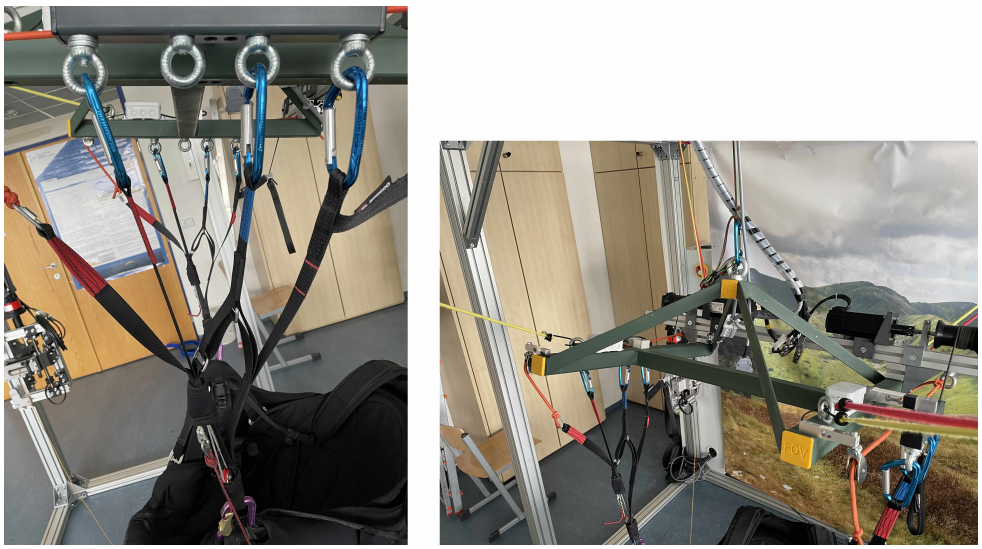
Kluzák vykonává za letu sadu pohybů, které by simulátor měl být schopný fyzicky napodobit. Mohu je rozdělit na:

- stoupání a klesání obr.5 d)
- náklon kolem os
  - podélná osa obr.5 a)
  - příčná osa obr.5 c)
  - svislá osa obr.5 b)



Obrázek 5: Náklony v jednotlivých osách padákového kluzáku.[1]

Jako nejvhodnější řešení, jak z finančního,tak praktického hlediska, byl zvolen koncept lanových paralelních robotů (cable-driven parallel robot), u kterých dochází ke změně polohy a orientace v důsledku namotávání a odmotávání lan. Pro vyvolání nejreálnějšího pocitu z letu bylo rozhodnuto, že pilot bude v simulátoru usazen v normálním paraglidingovém posedu s popruhy. Tyto popruhy jsou navedeny na jednotnou konstrukci z hliníkových profilů, která zajišťuje korektní zavěšení posedu. Zároveň umožní nahrazení typických nosných A, B, C, D šňůr, jedním lanem připevněným k hornímu pohonu, který je umístěn ve středu stropní stěny konstrukce simulátoru. Úkolem tohoto pohonu je pomocí pohybu v ose z replikovat pocit, který vyvolává stoupání a klesání.[3]

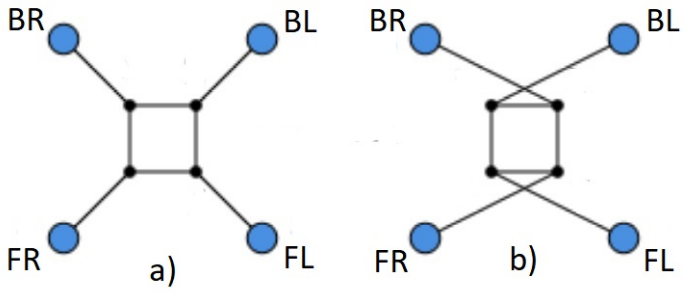


Obrázek 6: Konstrukce zavěšení posedu.

Je nezbytné také vyřešit, jakým způsobem se bude posed pohybovat v podélné, příčné a svislé ose. Důležitý fakt, který si musíme uvědomit je, že při letu dochází pouze k relativně malým náklonům, které kromě extrémních situací nepřesáhnou 30 stupňů. K tomu, aby byl simulátor schopen provádět všechny tyto pohyby, jsou zapotřebí alespoň čtyři boční pohony. Pohony byly po uvážení umístěny na každou ze čtyř rohových svislých tyčí kvádrové konstrukce s lany uchycenými za úchyty, rozmístěnými v rozích spodní část posedu, jak je znázorněno na následujícím obrázku. [3]

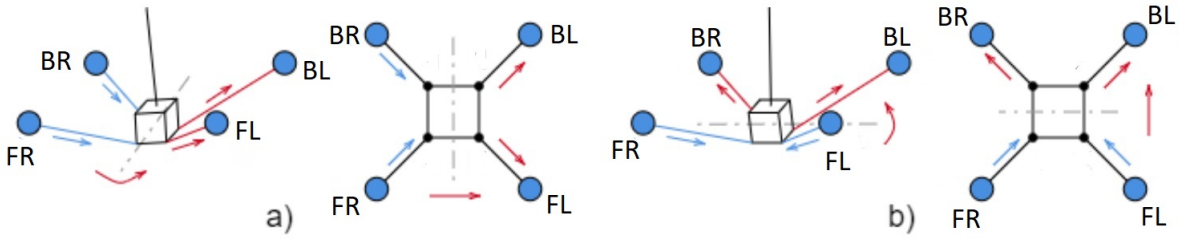
Obrázek 7: Upevnění posedu k lanům jednotlivých pohonů.

Pro uchycení lan byly koncipovány dvě konfigurace. Přímé uchycení, tedy napojení lan přímo od bočních pohonů k odpovídajícím úchytům v rozích posedu a křížové uchycení, obě znázorněné na obrázku 8. Každé z těchto řešení má své výhody a nevýhody, které představím. Pro přehlednost budu označovat jednotlivé pohony pomocí jejich anglických zkratek. Přední levý jako **FL**(front-left), přední pravý **FR**(front-right), zadní levý **BL**(back-left) a zadní pravý **BR**(back-right).



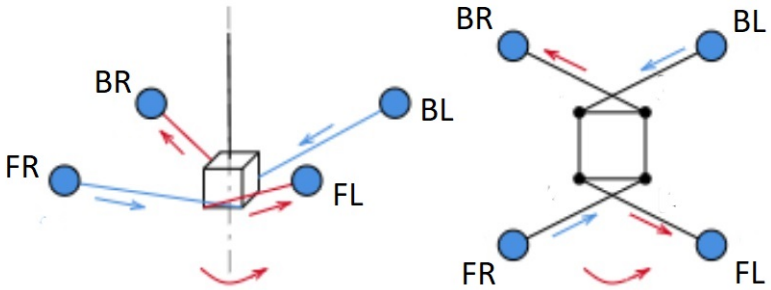
Obrázek 8: a) Přímé uchycení b) Křížové uchycení[3]

Při přímé konfiguraci je možné se sedačkou naklánět v příčné ose x do strany, navinutím lan připojených k bočním pohonům **FL**, **BL** a postupným odvinutím na pohonech **FR** a **BR**. Inverzí směru otáčení obou dvojic pohonů se sedačka bude pohybovat do strany druhé. V podélné ose y se naklánění dopředu provádí pomocí navinutí předních pohonů **FL**, **FR** a odvinutí zadních pohonů **BL** a **BR**. Pohyb opačným směrem je realizován stejným prohozením činností pohonů, jako při naklánění v ose x.[3]



Obrázek 9: Pohyb sedačky v podélné a příčné ose.[3]

Přímá konfigurace sice zvládne naklánět sedačkou v příčné a podélné ose, ale neřeší to problém rotování sedačky kolem svislé osy. Tento pohyb je při letu reálného padákového kluzáku podstatný, tím pádem nemohl být při návrhu konstrukce zanedbán. Nabízí se několik řešení, například přidání dalších pohonů, které přes kladky budou sedačku požadovaně naklánět nebo pohon, který bude rotovat s celou konstrukcí zavěšení. Mnoho řešení přišlo v úvahu, ale díky již zmíněnému faktu o relativně malém náklonu pilota při letu, bylo rozhodnuto, že nejlepší východisko je lana od jednotlivých pohonů připevnit k posedu křížově. Tato konfigurace nám dovolí náklon v osách x a y stejně jako v přímá konfiguraci. Zároveň nám umožní rotovat se sedačkou okolo svislé osy z, bez přidání dodatečných pohonů, což je z ekonomického a konstrukčního hlediska velká výhoda.[3]



Obrázek 10: Pohyb sedačky ve svislé ose.[3]

2.2 Pohony a řídící systém

Elektrické servomotory jsou automatické regulační pohony, které ke korekci svého pohybu používají mechanizmus zpětné vazby. S enkodéry a dalšími prvky tvoří společně mechatronický systém, který se celkově skládá z:

- Elektrického servopohonu s enkodéry
- Výkonového měniče
- Řídícího a regulačního obvodu

Servomotory operují vždy v uzavřené regulační smyčce záporné zpětné vazby. Regulační smyčka dále obsahuje řadu vnořených zpětnovazebních smyček, jako například rychlostní, polohovou a proudovou.[4][5] Servomotory můžeme dále dělit na rychlostní servopohon a polohový servopohon. Rychlostní servopohon má v regulační smyčce pouze otáčkovou zpětnou vazbu a jeho úkolem je přesné a rychle sledování a udržování zadávané rychlosti. Polohový servopohon sleduje polohu, v jaké se servopohon nachází pomocí úhlu natočení nebo absolutní polohy. Pro účely simulátoru se využívají polohové servopohony se sledovacím typem polohové regulace. Sledovací polohová regulace se užívá v případech, kdy při polohování záleží nejen na finální pozici, ale i na trajektorii a rychlosti, jakou se tam dostane. Mezi další aplikace tohoto typu regulace patří víceosé systémy, jako jsou roboti či posuvy obráběcích strojů.[4]

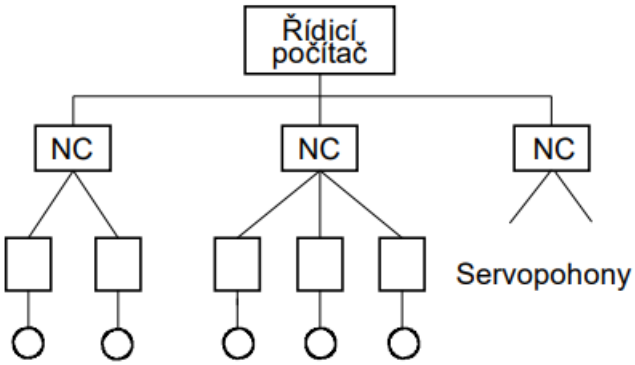
Největší nároky kladené na servomotory jsou v oblasti jejich regulačních parametrů, tudíž rychlost a přesnost regulace. Zároveň se musí dbát i na jejich spolehlivost.

Prostředí, v kterém se servopohony nacházejí, může mít na chod pohonu a regulačního procesu negativní vliv. Způsobují to rušivé vnější vazby, jako například teplota okolí či vlhkost. Ty ale nejsou jediné, zároveň na pohony působí i rušivé vlivy ze sítě. Ty můžou zahrnovat pokles napětí, přepěťové špičky a řadu dalších vlivů. Na naši soustavu bude mít však největší rušivý vliv změna zatěžovacího momentu a momentů setrvačnosti, díky charakteristice pohybu sedačky v simulátoru.[4]

2.3 Řídící systémy servopohonů

Nejjednodušší individuální servopohony mohou být ovládány jednotlivými pracovníky pomocí ovládacího panelu ručně, ale častěji, jak je to i v našem případě, jsou součástí rozsáhlejšího řídicího systému. Tyto systémy řízení mají rozdílné způsoby uspořádání, ale nejčastěji se využívá uspořádání hierarchické, které se dále může dělit na úrovně od nejnižší po nejvyšší.[4]

- **Nejnižší úroveň** je nejblíže technologickému procesu, to znamená přímo polohové nebo rychlostní pohony. Zajišťuje samotný dynamický pohyb a diagnostiku chyb pro řídicí systém.
- **Střední úroveň** se nachází mezi pohony a řídicím počítačem a zahrnuje numerický řídicí systém, který ovládá operace technologického procesu.
- **Nejvyšší úroveň** je nadřazený řídicí počítač, ke kterému jsou skrze lokální počítačovou síť připojené numerické řídicí jednotky jednotlivých pohonů.



Obrázek 11: Příklad hierarchického uspořádání.[4]

U systémů řízených pomocí hierarchického uspořádání se předpokládá alespoň omezená funkce systému při selhání nadřazené úrovně řízení. Při řízení simulátoru padákového kluzáku to však není možné, jelikož je pohyb jednotlivých pohonů přímo závislý na datech ze simulace, která se provádí na řídicím počítači. Pro tok informací mezi numerickým řízením a řídicím počítačem se používá buď architektura se společnou sběrnicí, která má paralelní přenos dat nebo kruhová architektura s přenosem sériovým. Architektura se společnou sběrnicí se využije hlavně, když jsou servopohony součástí jednotného mechanického systému, nejčastěji u robotů. Zatímco kruhová architektura bude aplikována v systémech, kde se informace přenáší na větší vzdálenost, nejčastěji pomocí optického média.[4]

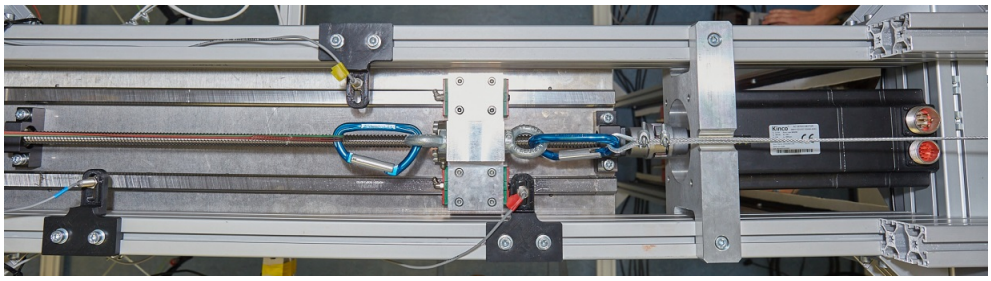
2.4 Zvolené servomotory pro použití na simulátoru

Pro účely simulátoru byly vybrány servomotory čínského výrobce Kinco. Mezi hlavní faktory výběru patřila atraktivní cena, požadovaný výkon a celkově byly jejich parametry adekvátní pro aplikaci v simulátoru. Velkou roli ve výběru hrála taky možnost pořídit servomotory společně s výkonovými drivery.

Ze série Kinco Servomotors byly vybrány celkem 3 specifické typy motorů pro rozdílné účely v simulátoru.

Pro pohon osy z byl zvolen servomotor s brzdou Kinco SMH110D-0157-30ABK-4HKC o výkonu 1570 W, který je schopen pomocí kuličkového šroubu dosáhnout lineárního typu pohybu s minimálními třecími ztrátami a vysokou přesností. Na návrh pohonné jednotky osy z byly kladené vysoké bezpečnostní nároky, jelikož drží celou váhu nosné konstrukce se sedačkou a pilotem a zároveň s ní musí ještě pohybovat.[3]

Do hloubky se návrhem pohonné jednotky zabývá práce kolegy Petra Staška pod názvem Návrh hlavních pohonů simulátoru padákového kluzáku.[3]



Obrázek 12: Pohon osy z.[3]

K ovládání simulátoru padákového kluzáku se používají dva servomotory Kinco SMH60S-0040-30AAK3LKH. Servomotor bude navíjet řídicí šňůry, které bude pilot stahovat a umožní mu tak řídit padákový kluzák v simulaci. Jelikož na motor nebudou kladeny vysoké mechanické nároky, bude jeho výkon 540 W pro naše účely dostačující.

Návrhem a odladěním řídícího programu se zabývá práce kolegy Richarda Sivery s názvem Pokročilé řízení servomotorů pro simulátor padákového kluzáku.[2]

Obrázek 13: Pohon řídícího lana.

Boční pohon zařizují 4 servomotory Kinco SMH80S-0100-30AXK-3LKX. Na tyto pohony jsou kladené rozdílné dynamické nároky než na pohon osy z a to rychlejší pohyb s menší zátěží. Zároveň je však zapotřebí pohybovat s větší délkou lana. Ze zmíněných důvodů již není možné použít lineární pohyb jako u pohonu osy z, a proto bylo zvoleno řešení pohonu pomocí navíjecího bubnu. Pro lepší využití výkonu servomotoru je zapotřebí instalace převodovky s přírubou pasující na hřídel motoru, jejíž cena se přibližuje ceně samotného motoru.[3]

Obrázek 14: Boční pohon.

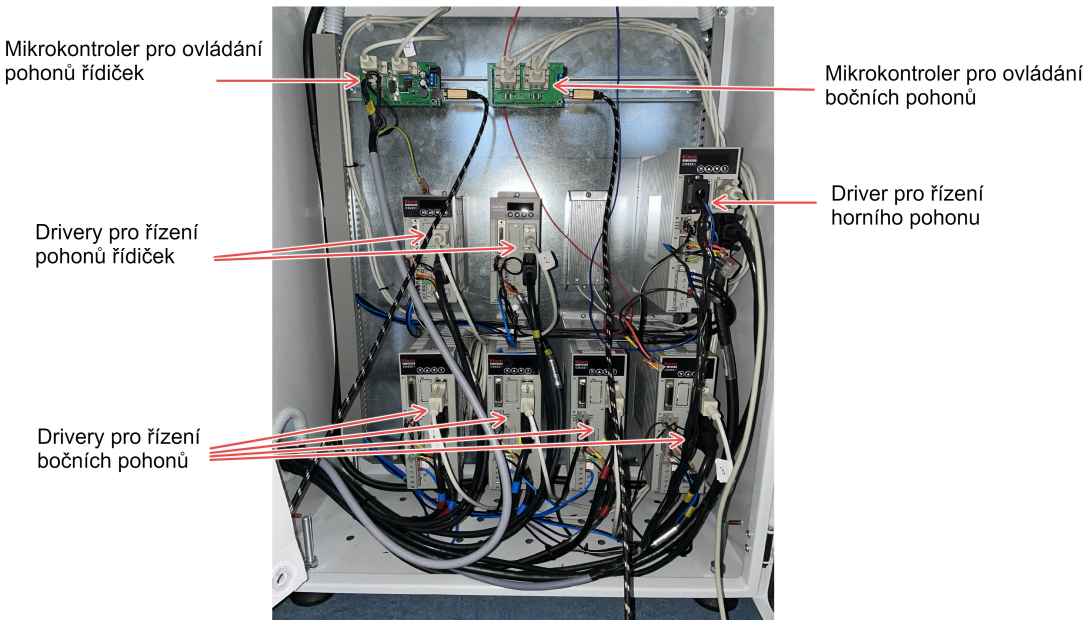
2.5 Řízení Kinco servomotorů

Řízení servomotorů Kinco zajišťuje řídící systém, který se skládá s mikrokontroleru a výkonového driveru. Každý účelový servomotor má svůj driver, všechny jsou ze série Kinco SD drivers.

Dva servomotory Kinco SMH60S-0040-30AAK3LKH zajišťující stahování řídicích šňůr, jsou řízeny pomocí dvou driverů Kinco CD422S-AA-000. Ty jsou připojené do 1f 230V sítě a jejich výstupem je řízené 3f napětí s maximálním výkonem 750W a frekvencí od 0 do 400 Hz.[5]

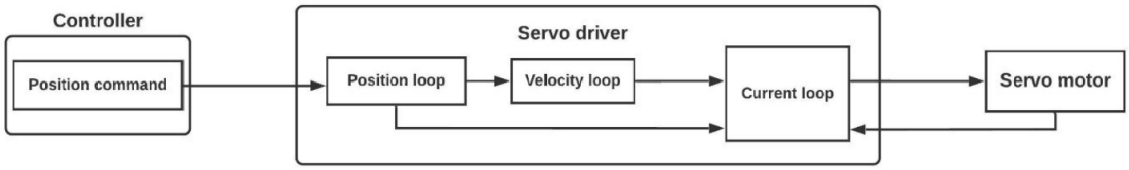
Samostatný servomotor Kinco SMH110D-0157-30ABK-4HKC, který zajišťuje pohyb osy z je řízený driverem Kinco CD622S-AA-000, který je připojen do 3f 400V sítě a je schopen dosáhnout výkonu 3000W s frekvencí od 0 do 400 Hz.[5]

Čtyři boční servomotory Kinco SMH80S-0100-30AXK-3LKX, které zajišťují prostorový pohyb jsou řízeny čtyřmi drivery CD432S-AA-000. Tyto drivery jsou připojené do 1f 230V sítě s výstupem o výkonu 2100W a frekvencí od 0 do 400 hz.[5]



Obrázek 15: Rozvaděč s výkonovými drivery a mikrokontrolery.

Popsáním regulace servopohonů se také zabývá práce kolegy Richarda Sivery [2], proto jsem pro popsání funkce čerpal výhradně z ní.
Na blokovém schématu je znázorněno propojení mikrokontroleru, driveru se vnitřními zpětnovazebními smyčkami a samotného servomotoru s jeho vazbami na driver. Řídicí smyčka servo driveru Kinco obsahuje tedy celkem tři smyčky:



Obrázek 16: Blokové schéma řídícího systému.[5]

- **Current loop** (proudová smyčka) – Hlavní účel této smyčky je řízení točivého momentu. Slouží k tomu PI regulátor, jehož proporcionální a integrační parametry zesílení lze nastavit pomocí členů *Kcp Kci*.
- **Velocity loop** (rychlostní smyčka) – V rychlostní smyčce figurují dva členy a to proporcionální *Kvp* a integrační *Kvi*. Nastavení členu *Kvp* nám určuje, jak velká bude regulační odchylka. Nastavením členu *Kvp* následně určíme rychlost, jakou se regulační odchylka odstraní.
- **Position loop** (polohová smyčka) – U této smyčky dovoluje výrobce nastavení pouze proporcionálního členu regulátoru *Kvp*. Zvýšením zesílení *Kvp* sice dojde ke zvýšení rychlosti dojetí na cílovou polohu, ale nastavením na moc vysokou úroveň může způsobit větší hlučnost motoru nebo dokonce oscilace.

Proudová smyčka úzce souvisí s parametry motoru a její parametry jsou již defaultně nastaveny, parametry zbylých smyček by se měly nastavit v závislosti na charakteru zátěže.[2]

Servomotory Kinco mají několik ovládacích režimů, které se dají v driveru měnit. Po náš způsob řízení mikrokontrolerem jich z celkových šesti použijeme pouze tři.[2]

- **Režim dojetí na cílovou polohu**

  V tomto režimu je nutné nastavení požadované rychlosti, akcelerace, zpomalení a cílové polohy. Aktivací tohoto příkazu se nejprve motor roztočí zadanou akcelerací na požadovanou rychlost, při které si udržuje maximální točivý moment, který je závislý na maximálním proudu. Při dojezdu k zadané pozici motor bude zpomalovat zadaným zpomalením.[2] Režim dojetí na cílovou polohu je pro nás důležitý, jelikož to je jeden ze dvou režimů užitých pro polohování.

- **Režim točivého momentu**

  Tento ovládací režim nabízí uživateli možnost nastavit konstantní točivý moment, který si následně bude motor udržovat. Po zapnutí tohoto režimu se motor roztočí vlastní akcelerací na vlastní nastavenou rychlost otáčení. Tuto rychlost si udržuje, dokud proti motoru nepůsobí zatížení o stejné velikosti. V ten moment se bude motor točit rychlostí o trochu menší než je maximální nastavená hodnota. Ve chvíli, kdy proti motoru působí točivý moment, který je větší než nastavený, je motor schopen se roztočit proti svému nastavenému směru otáčení. Když tedy nastavíme servomotoru nízký točivý moment, je možné ho přemoci i pouhou silou uchopení hřídele rukou. Uplatnění tohoto režimu se najde jak u řízení pomocí řídicích šňůr, jelikož bude muset pilot přemoct točivý moment motoru u jejich stahování, tak i u polohování, kde budou muset být boční lana přivázaná k sedačce neustále napnutá, i když se budou odmotávat pohybem sedačky na opačnou stranu.[2]

- **Režim dojetí na výchozí polohu**

  Tento režim se uplatní k dojetí do výchozí polohy. Ideálně by se měl využít hned po zapnutí driveru a může fungovat například na principu dojetí ovládaného předmětu na polohu určenou senzorem nebo pootočení hřídele o určitý offset. V našem případě se výchozí poloha určuje tak, že lana, která motor přitahuje jsou napnutá a působí tedy na motor silou, kterou již svým relativně malým krouticím momentem nedokáže přemoci.[2]

2.6 Numerické řízení

Jako jednotka numerického řízení se používá mikrokontroler Arduino Mega 2560, který je postaven na procesu ATmega2560. Obsahuje celkem 54 digitálních vstupů/výstupů, 16 analogových vstupů a 4 vstupy určené pro sériovou komunikaci, které se využívají pro komunikaci s driverem. Dále obsahuje oscilátor o frekvenci 16 MHz. Pro práci s mikrokontrolerem musí uživatel využívat vývojové prostředí Arduino Software(IDE) a vývoj se provádí pomocí variace programovacího jazyka C++. Pro komunikaci mezi zvolenými drivery a mikrokontrolerem se zvolila sériová komunikace RS-232, která bude připojena do 4 sériových portů na desce mikrokontroleru. Arduino sériová komunikace využívá logických napěťových úrovní technologie TTL, tedy 0V pro logickou „0“ a 5V pro logickou „1“, zatímco RS-232 využívá napěťové úrovně -5V až -15V pro logickou „1“ a 5V až 15V pro logickou „0“. Z důvodů rozdílných napěťových úrovní mezi komunikacemi se musí využít převodník, jinak by mohlo dojít k poškození desky. Konkrétně se jedná o převodník MAX3232, jehož maximální přenosová rychlost s garantovanou přesností je 120 Kbps a vyžaduje své vlastní napájení s napětím 3,3 nebo 5 V.[2]

3 Headset s virtuální realitou

Headset s virtuální realitou je nedílnou součástí celku simulátoru padákového kluzáku, bez něj by se pilot nikdy nemohl vnořit do simulace a celý projekt by ztratil pointu. V průběhu vývoje se používaly dva rozdílné modely headsetu. Prvotní myšlenka byla používat headset Vive od společnosti HTC a postupně se přešlo na zařízení Quest 2 od společnosti Meta. V této kapitole bych rád probral výhody obou zařízení a důvody, proč došlo k jejich výměně, zároveň stručně přiblížím historii zařízení s virtuální realitou.

3.1 Stručná historie vývoje Virtuální reality

Historie virtuální reality sahá dále do minulosti, než může být na první zamyšlení zřejmé. Virtuální realita, dále pouze VR, byla vynalezena v 50. letech 20. století a její vývoj zažíval vrcholy a pády. První systém VR s displejem na hlavě (HMD – Head Mounted Device) pod názvem The Sword of Damocles, vynalezli v roce 1968 počítačový vědec Ivan Sutherland a jeho student Bob Sproull. Zatímco termín "virtuální realita"zpopularizoval Jaron Lanier v 80. letech 20. století. V 90. letech se VR začala používat pro výcvik a simulace v americké armádě a Národním úřadu pro letectví a vesmír (NASA). Masová výroba systémů VR začala koncem 20. století v čele s firmou Virtuality.[6]

Současná VR zařízení se objevila s uvedením prototypu Oculus Rift připojeného k počítači v roce 2010. V letech 2014 až 2017 se trh posunul od souprav připojených k PC (např. **HTC Vive**) k soupravám připojeným ke konzolím (např. PSVR od Sony) a soupravám připojeným k mobilním zařízením (např. Samsung GearVR a Google Cardboard). V roce 2018 se objevily soupravy bez vázání (např. Oculus Go, Lenovo Mirage Solo a HTC Vive Focus), čímž se VR stala nezávislou platformou.[6]

VR vstupuje do své druhé generace a ve srovnání s generací předchozí se očekává větší přitažlivost pro spotřebitele i společnosti. Technologie VR se za posledních pět let výrazně vyvinula, přičemž došlo ke zlepšení jak na straně hardwaru, tak softwaru. Překážkou širokého přijetí však byly problémy, jako je latence, nevolnost, vysoké ceny a nedostatečně rozvinuté ekosystémy. Společnosti zabývající se VR stále více využívají umělou inteligenci a cloudové technologie k rozvoji silnějších ekosystémů, zatímco příchod 5G slibuje vyřešit problémy s latencí a nevolností.[6]

3.2 Headset HTC Vive

Zařízení HTC Vive je jedním z prvních komerčně dostupných zařízení s virtuální realitou, určené pro pozorování a pohybování se ve virtuálním prostředí až o velikosti místnosti. Komplet zařízení obsahuje samotný headset, dva ovladače, dvě senzorické základny a linkbox.[7]

Obrázek 17: Komplet zařízení HTC Vive.

Výhodou zařízení HTC Vive je možnost pohybovat se v předem definovaném prostředí, které bude korespondovat s prostředím ve virtuálním světě. Oblast, ve které se může uživatel pohybovat, je dána rozmístěním dvou senzorických základen. Tyto základny se umisťují diagonálně proti sobě, do rohů čtvercové nebo obdélníkové oblastí a vzdálenost mezi nimi by neměla přesáhnout 5 metrů. Zároveň by měly být umístěné do určité výšky a pod úhlem, aby byla celá oblast v jejich zorném poli. Sledování pozice funguje na principu lighthouse laserů, které se po vyzáření do určené oblasti odrazí od senzorů na headsetu a ovladačích. Senzorické základny jsou schopné z časové prodlevy vyzáření a odražení vyvodit přesnou polohu v prostoru. Nevýhodu zařízení HTC Vive je nutnost propojení přes kabel, což se při vývoji ukázalo jako nepraktické.[7]

3.3 Headset Meta Quest2

Zařízení Meta Quest 2 bylo na trh uvedené v roce 2020 a je to samostatné zařízení, což znamená, že na rozdíl od HTC Vive nepotřebuje ke své funkci počítač. Vše, co je potřebné pro simulaci, se už nachází v samotném headsetu včetně vlastního procesoru, paměti, grafického čipu a baterie. Komplet zařízení tedy obsahuje pouze headset a sadu dvou ovladačů.[8]

Obrázek 18: Komplet zařízení Meta Quest2.

Meta Quest2 disponuje vestavěnými senzory pro snímání polohy a pohybu, které umožňují uživateli pohyb v prostoru a možnost pomocí gest a pohybů interagovat s objekty v simulaci. Meta Quest2 také obsahuje vystavěnou knihovnu aplikací, které může uživatel hned využít. Zároveň je možné se k počítači připojit pomocí Wifi nebo pomocí kabelu a softwaru Oculus link, což umožní uživateli hrát nejen předurčené aplikace pro Meta Quest2, ale také i další tituly zaměřené na VR headsety nebo i aplikace vlastního vývoje.[8]

3.4 Důvod změny headsetu během vývoje

Jak jsem se již zmínil, během vývoje došlo ke změně primárního headsetu pro simulátor padákového kluzáku. Nejdříve vývoj probíhal s Headsetem HTC Vive, který měl pro náš účel jednu hlavní výhodu a to možnost sledování polohy v předem definovaném prostotu. Myšlenka byla taková, že se jeden z ovladačů permanentně umístí na konstrukci zavěšení a díky němu se bude moct odvodit přesná poloha pilota v konstrukci simulátoru. Tato poloha by se měla dále využít pro polohování. Při testování se ale ukázalo, že tato poloha není dostatečně přesná, aby byla k účelům polohování dostačující. Zároveň se jako problémové u zařízení HTC Vive ukázalo připojení headsetu pomocí kabelu, který v konstrukci překážel. Kvůli zmíněným důvodům a s ohledem na nižší cenu se ukázalo výhodnější a praktičtější pořízení a pokračování vývoje se zařízením Meta Quest 2. To mělo za výsledek kompletní ukončení testování konceptu polohování pomocí senzorů na headsetu a ovladačích.

4 Senzorické prvky

V následující kapitole představím senzorické prvky využité v konstrukci simulátoru a vysvětlím principu jejich funkce. Konkrétně to jsou tenzometry, využívané k zjišťování náklonu pilota v sedačce a enkodéry servomotorů, pomocí kterých sledujeme polohu pilota v konstrukci simulátoru.

4.1 Funkce tenzometrů pro sledování náklonu sedačky

Tenzometr je jedním z nejdůležitějších nástrojů elektrické měřicí techniky, jehož odpor se mění v závislosti na působící síle. Převádí sílu, tlak, tah, hmotnost atd. na změnu elektrického odporu, kterou lze následně měřit. Když na nehybný objekt působí vnější síly, vzniká mechanické napětí a deformace.[9]

V našem případě jsou tenzometry celkem dva, umístěné na každé straně konstrukce se sedačkou a je na nich zavěšená celá váha pilota a sedačky. Velikostí působící síly na každý z tenzomentrů lze určit váhu pilota, která by se dala dále použít k přizpůsobení dynamiky pohybu. V prototypu simulátoru, to ale nebylo implementováno. Rozdílem jednotlivých sil působících na tenzomentry, je možné získat náklon pilota, což je jeden z hlavních způsobů řízení padákového kluzáku probraný v kapitole 1.1. Tento způsob řízení implementovaný je. Řízení padákového kluzáku v simulaci je navrženo tak, že je možné zadat hodnotu o změně těžiště pilota a tím ovlivnit vlastnosti řízení. Pomocí hodnot získávaných z tenzometrů a hodnot stažení jednotlivých řídicích šňůr, je možné zajistit paralelní řízení simulátoru, což odpovídá způsobu řízení reálného padákového kluzáku.

4.2 Enkodéry servomotorů - snímání polohy a řízení

K určení přesné polohy hřídele motoru slouží snímače polohy neboli enkodéry. Enkodéry můžeme dále podle způsobu zjišťování polohy hřídele rozdělit na absolutní, inkrementální a cyklicky absolutní. [4]

Absolutní snímače získávají absolutní polohu hřídele, to nám říká, že každé poloze je udělena jednoznačná hodnota signálu. Výhodou absolutního snímače je existence informace o poloze hřídele okamžitě po zapnutí motoru. Nejčastěji se takové snímače realizují pomocí optických snímačů s kódovacími kroužky.
Inkrementální snímače disponují vysokou přesností a rozlišením, ale informaci o absolutní poloze jsou schopné získat pouze najetím na referenční polohu, hned po zapnutí servomotoru. Reálná poloha snímače je následně dána obsahem čítače, u kterého inkrementálně narůstá hodnota snímaných pulzů.
Cyklické absolutní snímače odměřují polohu pouze v omezené oblasti, na rozdíl od absolutních

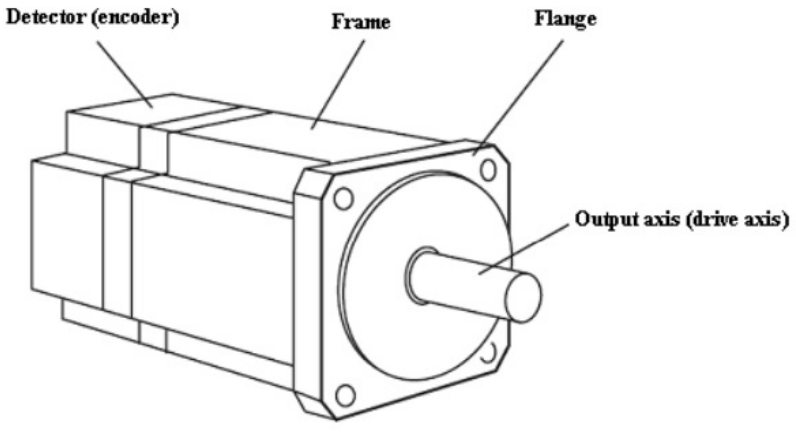
snímačů. Tato omezená oblast může být například pouze v rozsahu jedné otáčky. [4][2]

Snímače polohy můžeme následně rozdělit na snímače přímé a nepřímé. U nepřímého snímání je rotační polohový snímač umístěn na hřídeli motoru. Nevýhodou tohoto způsobu odměřování je skutečnost, že nelinearity mechanických převodů se nacházejí uvnitř motoru a způsobují chybu řízení, která se již nedá regulací odstranit. Přímé snímání využívá snímačů lineárních nebo rotačních, podle druhu výsledného pohybu. Na rozdíl od nepřímého snímaní, se senzor polohy nachází přímo na části vykonávají pohyb. Takovýto způsob snímání by v našem příkladě mohl být zrealizován umístěním senzoru přímo na navíjené lano.[4][2]

Rozlišení snímačů určuje počet pulzů generovaných na jednu otáčku. U rotačních snímačů se rozlišení vyjadřuje v pulzech na otáčku (PPR) a u lineárních snímačů se označuje jako pulzy na palec (PPI) nebo pulzy na milimetr (PPM). Počítáním generovaných impulzů může snímač určit velikost dráhy, nikoli však její směr. K určení směru potřebuje snímač více informací. Tento problém se řeší pomocí druhého kanálu, který je fázově posunutý oproti prvnímu. Přečtením, který kanál vede a který následuje, může snímač určit směr otáčení nebo lineárního pohybu.[10]

4.2.1 Zvolené enkodéry

Servomotory od výrobce Kinko využívají inkrementální enkodér. Jak jsem již zmínil, jedná se o druh snímače, který zjišťuje relativní polohu hřídele. V servomotorech série SHM se používají enkodéry s rozlišením 2500 pulzů na otáčku. Zároveň se jedná o enkodéry kvadraturní, což nám říká, že jsou schopné snímat rychlost i směr otáčení a disponují enkódováním typu X4, díky kterému se rozlišení zvyšuje na 10000 pulzů na otáčku. To znamená, že enkodér je schopný zjistit deset tisíc odlišných poloh na jednu otáčku hřídele a každé je schopný dosáhnout.[2][10]



Obrázek 19: Popsané schéma servomotoru s enkodérem.[5]

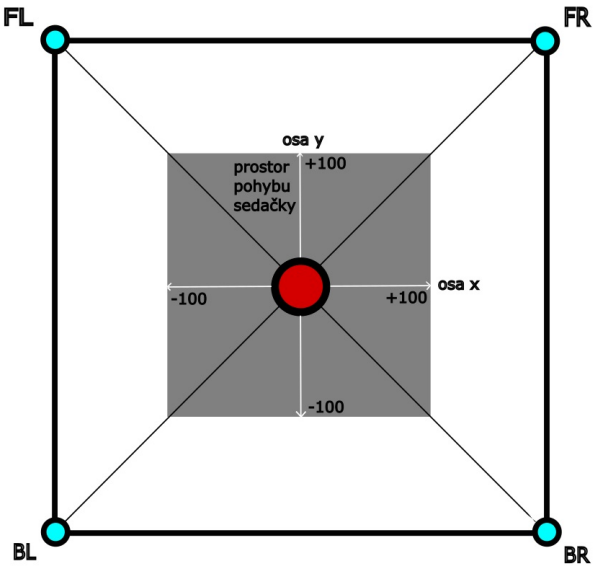
5 Návrh platformy pro polohování sedačky simulátoru

V této kapitole se zaměřím na můj zvolený koncept řešení polohování simulátoru padákového kluzáku. Nejprve přiblížím způsob normalizace dat ze simulace, následně popíšu výpočet výsledné polohy, určím nedostatky řešení a jejich možné kompenzování. Všechny výpočty a funkce jsou naprogramovány v jazyce C# a znalosti potřebné ke správnému naprogramování polohování jsem čerpal z oficiální dokumentace [11].

5.1 Získaná data ze simulace

Data o poloze padákového kluzáku v simulaci nám chodí ve formátu úhlu natočení v jednotlivých osách. V ose x se toto natočení nazývá roll, v ose y pitch a v ose z yaw.

Tyto data nám ale o poloze, kam má sedačka dojet nic neříkají, proto je musíme nejdříve normalizovat. Také si musíme nadefinovat prostor, ve kterém se bude sedačka v konstrukci pohybovat, protože z fyzického a bezpečnostního hlediska se nemůže pohybovat v celém prostoru konstrukce. Zároveň pro další návrh normalizace zanedbáme úhel natočení v ose z, který není možný se zvoleným přímím uchycením realizovat, popsáno v kapitole 2.1. Tím pádem se budeme pohybovat pouze ve dvojdimenzionálním prostoru. Z těchto důvodů jsem zvolil dvouosý souřadnicový systém, který má rozmezí hodnot obou os od -100 do $+100$.



Obrázek 20: Diagram znázorňující přibližné rozmezí pohybu sedačky v konstrukci.

Normalizace dat ze hry se provádí funkcí *NormalizeAngle*, která probíhá konstantně pro obě chtěné osy s frekvencí toku dat do řídícího mikrokontroleru. Za proměnou *float angle* se dosadí natočení v jedné ose a za *float maxRealAngle* se dosadí maximální reálný úhel, kterého může sedačka v simulátoru padákového kluzáku dosáhnout.

```
1  private const float HalfCircle = 180; // degrees
2  private const float Circle = 360; // degrees
3
4  private float NormalizeAngle(float angle, float maxRealAngle)
5          {
6              if (angle ≤ HalfCircle)
7              {
8                  angle = -angle;
9                  if (angle < -maxRealAngle) angle = -maxRealAngle;
10             }
11             else if(angle > HalfCircle)
12             {
13                 angle = Circle - angle;
14                 if (angle > maxRealAngle) angle = maxRealAngle;
15             }
16
17             return (angle / maxRealAngle) * 100;   // <-100 , 100>
18         }
```

Výsledkem této funkce je reálná poloha sedačky v dané ose, v konstrukci simulátoru. Tato poloha nabývá hodnot -100 až 100 a odpovídá úhlu natočení v dané ose v simulaci. Funkce se provádí pro obě osy zároveň, aby byla získána poloha ve zvoleném souřadnicovém systému.

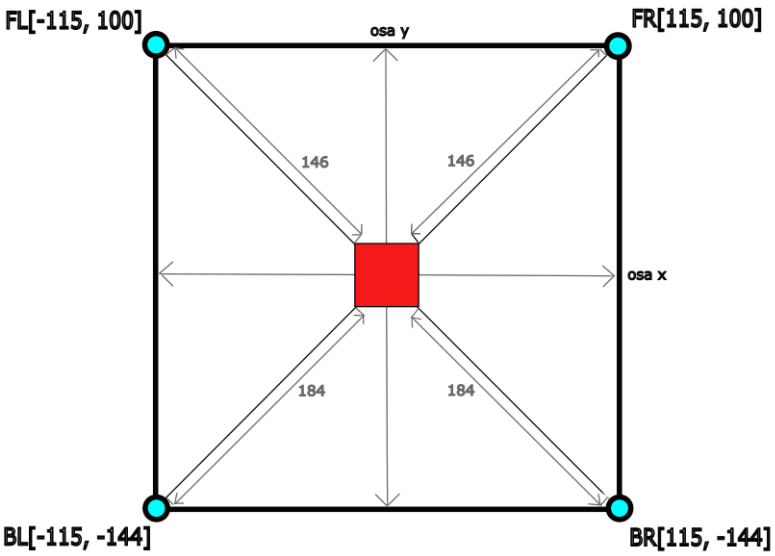
5.2 Koncept řešení polohování

V bakalářské práci s názvem Návrh hlavních pohonů simulátoru padákového kluzáku [3] se kolega Petr Stašek věnuje řešení polohování sedačky. Je třeba ale zdůraznit, že dané řešení bere v potaz, že je k dispozici i horní pohon osy z a klade důraz na křížové uchycení sedačky simulátoru padákového kluzáku. Řešení řízení horního pohonu je částečně připraveno na funkci, ale ve výsledném polohování nebylo implementováno. Při vývoji jsem také experimentoval pouze s přímým uchycením sedačky, tím pádem mé zvolené řešení umožňuje pouze pohyb v 2D prostoru bez možnosti natočení v ose z, což celkový výpočet polohy značně ulehčí. Zároveň jsem ale došel k závěru, že při výsledném pohybu sedačky a díky nasazenému headsetu, ve kterém probíhá simulace, pilot nepozná rozdíl přesných dodatečných náklonů, které by původně koncipované řešení přineslo. Z těchto důvodů si myslím, že je toto jednodušší řešení problému polohování dostačující pro účely prototypu simulátoru.

Polohování funguje na principu přitahování a povolování lan jednotlivých motorů. K tomu, abychom věděli délky jednotlivých lan a výchozí pozici sedačky, musí nejdříve sedačka simulátoru po zapnutí do takové pozice najet. Slouží k tomu režim dojetí na výchozí pozici, který jsem popsal v kapitole 2.5. Před zapnutím tohoto režimu musí v posedu již sedět pilot, jinak by síla působící proti pohybu motoru byla menší, než dostačující a sedačka by mohla dojet do výchozí pozice s menší odchylkou a ve výpočtech by vznikla chyba.

Když máme sedačku ve výchozí pozici, můžeme určit konstanty délek jednotlivých lan a relativní pozice pohonů vůči sedačce s pilotem. Konstanty jsem změřil pomocí metru na reálných délkách v konstrukci simulátoru a jelikož není konstrukce čtvercového tvaru, budou délky lan předních a zadních pohonů rozdílné.

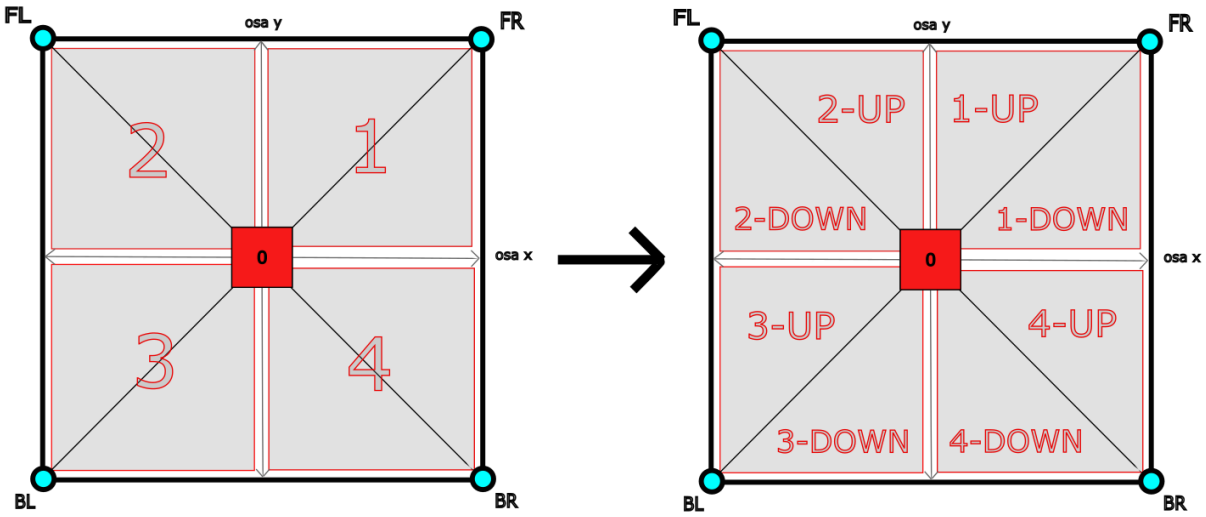
```
1   private const float StartDistanceF = 146; //cm
2   private const float StartDistanceB = 184; //cm
3
4           private static readonly Vector2 StartFL = new Vector2
5           {
6               x = -115,
7               y = 100
8           };
9
10          private static readonly Vector2 StartFR = new Vector2
11          {
12              x = 115,
13              y = 100
14          };
15
16          private static readonly Vector2 StartBL = new Vector2
17          {
18              x = -115,
19              y = -144
20          };
21
22          private static readonly Vector2 StartBR = new Vector2
23          {
24              x = 115,
25              y = -144
26          };
```



Obrázek 21: Znázornění reálných délek lan a relativní pozice pohonů vůči sedačce.

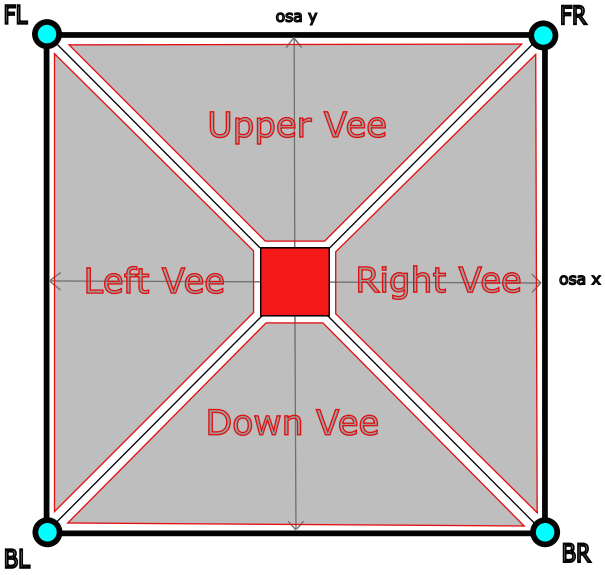
Dále musím určit způsob, jakým se budou přepínat pracovní režimy jednotlivých pohonů. Pro pohony, co budou zrovna přitahovat, jsem použil režim dojetí na cílovou polohu. Pro pohony, co povolují, jsem použil režim točivého momentu, jehož úkolem bude držet lana napnutá. Díky nízkému nastavenému točivému momentu se nechá pohon jednoduše přemoci přitahováním sedačky opačným směrem a tím je realizováno odmotávání. Jelikož ale budou lana neustále napnutá, nebude pro pohon problém, při změně pohybu a pracovního režimu, instantně zabrat za sedačku s pilotem.

Pro určení pracovních režimů jednotlivých pohonů jsem rozdělil prostor v konstrukci na čtyři kvadranty. Kvadranty jsem následně rozdělil na subkvadranty. Každý kvadrant má horní (UP) a dolní (DOWN) subkvadrant. Podle toho, v jakém kvadrantu nám vyjde požadovaná poloha ze simulace, je určen pracovní režim jednotlivých pohonů.



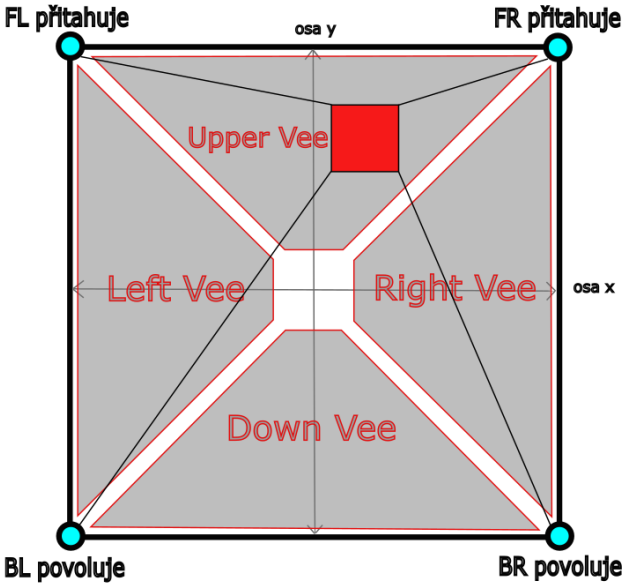
Obrázek 22: Rozdělení kvadrantů a subkvadrantů.

Vzhledem k charakteru návrhu polohování je možné si všimnout, že pracovní režimy pohonů v subkvadrantech 2-UP a 1-UP se chovají stejně. To samé platí pro subkvadranty 1-Down/4-UP, 3-Down/4Down a 2-Down/3-Up. Finálně jsem tedy prostor pohybu sedačky rozdělil na sektory ve tvaru písma „V“.



Obrázek 23: Rozdělení sektorů „V“.

Například, když by nám výsledná pozice vyšla v obou osách kladná a hodnota v ose y by byla větší, než hodnota v ose x, výsledná pozice by se nacházela v subkvadrantu 1-UP, neboli v sektoru horního „V“ (Upper Vee). To nám říká, že oba přední pohony FL a FR musí sedačku přitáhnout, tudíž budou operovat v režimu dojetí na cílovou polohu a oba zadní pohony BL a BR musí sedačku povolit, takže budou pracovat v režimu točivého momentu.



Obrázek 24: Příklad pozice v sektorech.

V kódu by to vypadalo následovně, přičemž proměnné *tilt.x* a *tilt.y*, reprezentují pozici získanou ze simulace. Porovnáním velikosti hodnot se pomocí proměnné *(konkrétní pohon)Command* nastaví potřebný pracovní režim pro korektní provedení polohování. Proměnná se bere ze struktury *commandDataA1*, ve které jsou uložené všechny řídicí proměnné pro motory. Tyto proměnné se odesílají prvnímu ze dvou mikrokontrolerů (obr. 2.5), který zajišťuje řízení pohybu všech čtyřech bočních pohonů. Společně s pracovním režimem se nastaví i sektor, který se později použije k určení, jakým pohonům poslat údaje o poloze.

```
1  if (_tilt.x ≥ 0 && _tilt.y ≥ 0)   // 1. Kvadrant
2              {
3                  if (_tilt.x > _tilt.y)    // sub-Kvadrant 1-DOWN
4                  {
5                      commandDataA1.FrontRightCommand = (byte)Commands.Position;
6                      commandDataA1.BackRightCommand = (byte)Commands.Position;
7                      _sector = ControlSector.RightVee;
8                  }
9                  else if (_tilt.x < _tilt.y)   // sub-Kvadrant 1-UP
10                 {
11                     commandDataA1.FrontLeftCommand = (byte)Commands.Position;
12                     commandDataA1.FrontRightCommand = (byte)Commands.Position;
13                     _sector = ControlSector.UpperVee;
14                 }
15                 else   // Diagonal
16                 {
17                     commandDataA1.FrontRightCommand = (byte)Commands.Position;
18                     _sector = ControlSector.Diagonal;
19                 }
```

Obdobné porovnání se provádí i pro ostatní kvadranty. Jediným rozdílem je, že následné rozdělení sub-kvadrantů se musí provádět v absolutních hodnotách, když je údaj natočení v jedné z os záporný.

5.3 Výpočet výsledné polohy

Z normalizace hodnot natočení v jednotlivých osách jsem zjistil polohu, na kterou by sedačka měla dojet. Z konstant naměřených vzdáleností následně určím, o jakou délku se mají pohony, v režimu dojetí na cílovou polohu, přitáhnout. Ve všech předchozích diagramech byla sedačka s pilotem znázorněna ve formě červeného čtverce s úchyty v rozích a s přímým upevněním lan. Z důvodu současně probíhajícího vývoje zavěšení sedačky s vývojem polohování, jsem provedl zjednodušení pro výpočet v podobě zavedení polohy sedačky pouze pomocí jednoho bodu. Alespoň pro malé vykompenzování vzniklých odchylek je konstanta délky lana měřena od středu sedačky. Pro reálnou sedačku je možné provést zpřesnění v kódu pomocí připočtení offsetu mezi jednotlivými úchyty na sedačce a změření délky lan od daných úchytů k pohonu. Tento offset by se následně mohl umístit do volitelné konfigurace s možností využití odlišných posedů, které mají rozdílné body uchycení. To bude navíc relevantní i proto, že existuje zmenšená, zjednodušená verze prototypu simulátoru, kde je uchycení mírně odlišné.

Nejprve si zjistím relativní polohu sedačky vůči jednotlivým pohonům. Relativní poloha je uložená v datovém typu *vector2* s tím že první prvek je souřadnice v ose x a druhý prvek je souřadnice v ose y. Z výpočtů vyplývá, že souřadnicový systém pro relativní polohu sedačky k pohonu, se u všech pohonů pohybuje pouze v kladných hodnotách.

Hodnota *tilt.y* se v záporné části osy násobí koeficientem $0,8$ a to z důvodu reálných dimenzí konstrukce, jejíchž půdorys je obdélníkového tvaru, zatímco zvolený souřadnicový systém normalizovaných úhlů nabývá hodnot odpovídajícím čtvercovému půdorysu. Konstanta o hodnotě $0,8$ se odvodila při následném testování navrženého řešení.

$$CoordsFL.x = -(StartFL.x - tilt.x) \tag{1}$$

$$CoordsFL.y = (StartFL.y - tilt.y); \tag{2}$$

$$CoordsFR.x = StartFR.x - tilt.x \tag{3}$$

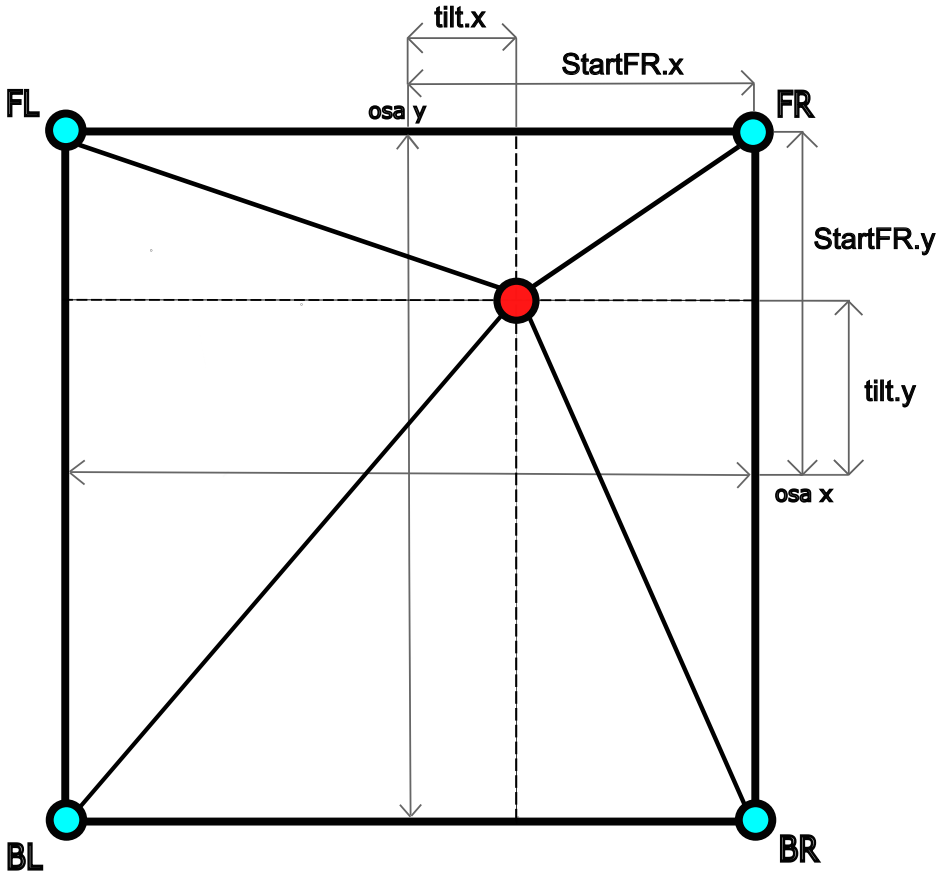
$$CoordsFR.y = StartFR.y - tilt.y \tag{4}$$

$$CoordsBL.x = -(StartBL.x - tilt.x) \tag{5}$$

$$CoordsBL.y = -(StartBL.y - (tilt.y \cdot 0.8)) \tag{6}$$

$$CoordsBR.x = (StartBR.x - tilt.x) \tag{7}$$

$$CoordsBR.y = -(StartBR.y - (tilt.y \cdot 0.8)) \tag{8}$$



Obrázek 25: Příklad relativní pozice sedačky vůči přednímu pravému pohonu.

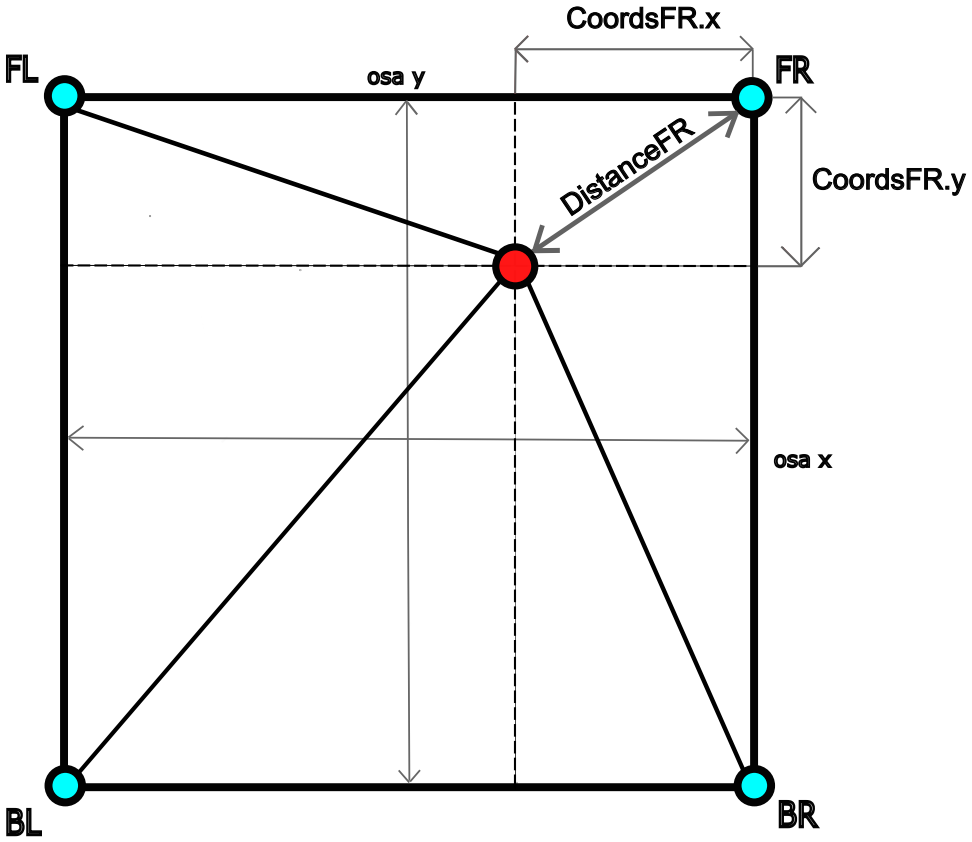
Ze získaných relativních poloh následně pomocí Pythagorovy věty vypočítám délku lana od výsledné polohy do relativní nulové pozice motoru.

$$DistanceFR = \sqrt{(CoordsFR.x \cdot CoordsFR.x) + (CoordsFR.y \cdot CoordsFR.y)} \tag{9}$$

$$DistanceFL = \sqrt{(CoordsFL.x \cdot CoordsFL.x) + (CoordsFL.y \cdot CoordsFL.y)} \tag{10}$$

$$DistanceBL = \sqrt{(CoordsBL.x \cdot CoordsBL.x) + (CoordsBL.y \cdot CoordsBL.y)} \tag{11}$$

$$DistanceBR = \sqrt{(CoordsBR.x \cdot CoordsBR.x) + (CoordsBR.y \cdot CoordsBR.y)} \tag{12}$$



Obrázek 26: Příklad výpočtu délky lana vůči přednímu pravému pohonu.

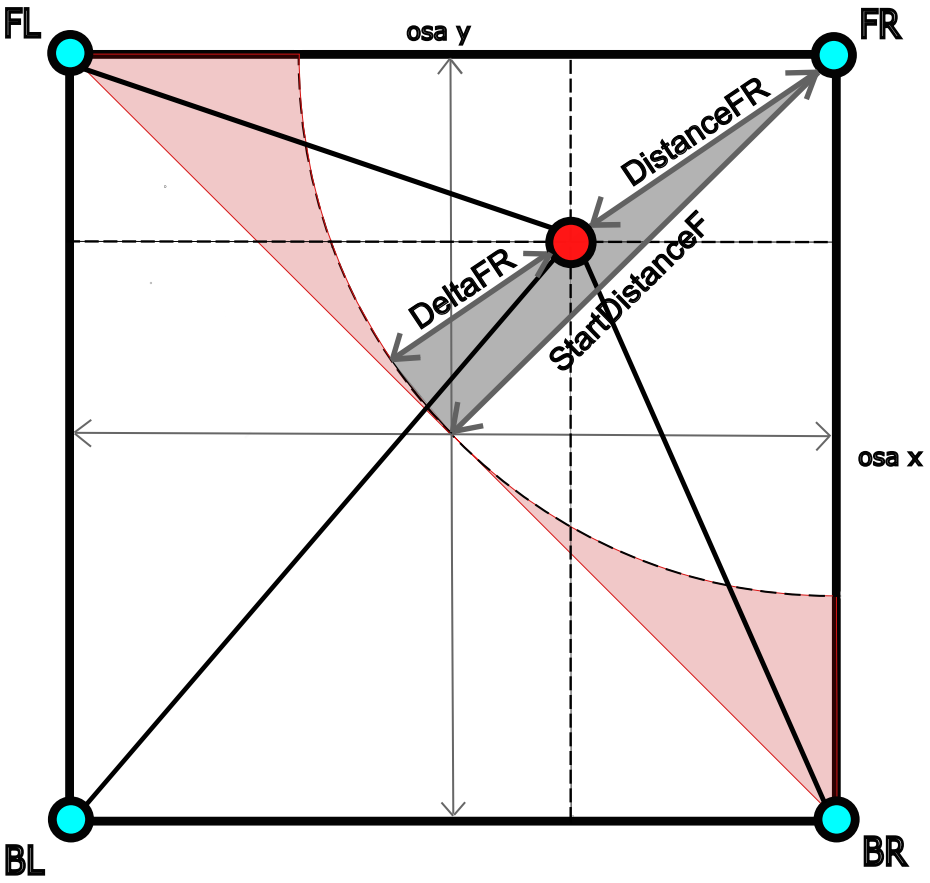
V dalším kroku si vypočítám Delta, rozdíl naměřené délky lana ve výchozí pozici a vypočítanou požadovanou délkou lana. Výsledek se bude převážně pohybovat v kladných hodnotách, což znamená, že se bude sedačka s pilotem přitahovat k danému pohonu. Může se ale stát, že se delta dostane do záporných hodnot. Znázorněno na obrázku 27 pomocí načervenalé oblasti. I přestože v těchto hodnotách pohon lano povoluje, zůstává v režimu dojetí na cílovou polohu. Kdyby tomu tak nebylo, režim točivého momentu by neměl dostatečnou sílu na udržení sedačky v cílové pozici. Odmotávání je tedy realizováno dvěma způsoby, v závislosti na tom, jestli se cílová poloha nachází v jednom ze dvou operačních sektorů „V“ příslušného motoru.

$$DeltaFR = StartDistanceF - distanceFR \tag{13}$$

$$DeltaFL = StartDistanceF - distanceFL \tag{14}$$

$$DeltaBL = StartDistanceB - distanceBL \tag{15}$$

$$DeltaBR = StartDistanceB - distanceBR \tag{16}$$



Obrázek 27: Zobrazení rozdílu delta pro přední pravý pohon

Posledním krokem k dosažení polohování je převést získané hodnoty výsledných délek lan na inkrementy pulzů otočení jednotlivých pohonů. Proto si zavedu novou konstantu a to maximální pozici.

```
1 private const float MaxPosition = 100000.0f;
```

Jak jsem již uvedl, zvolené servomotory disponují inkrementálním enkodérem, který díky enkódováním typu X4, dosahuje rozlišení 10000 pulzů na otáčku. Motor má ale na své hřídeli ještě přidanou převodovku s převodním poměrem 10:1. Tím pádem, na jednu otáčku bubnu s namotaným lanem se hřídel otočí desetkrát. Finální rozlišení se v důsledku přidání převodovky zvedá na 100 000 poloh na jednu otáčku bubnu. Konstanta maximální pozice tedy reflektuje jednu otáčku budnu.

Ve finálním výpočtu si nejdříve převedu rozdíl delta na procenta, kterými následovně vynásobím hodnotu maximální pozice. Tím si určím pohyb motoru v rozmezí jedné otáčky bubnu. Násobkem hodnoty maximální pozice lze určit výsledné rozmezí pohybu jednotlivých pohonů a tím i celkovou dynamiku pohybu sedačky s pilotem v prostoru konstrukce simulátoru.

$$FrontRightPosition = \left(\frac{deltaFR}{100}\right) \cdot 2 \cdot MaxPosition \tag{17}$$

$$FrontLeftPosition = \left(\frac{deltaFL}{100}\right) \cdot 2 \cdot MaxPosition \tag{18}$$

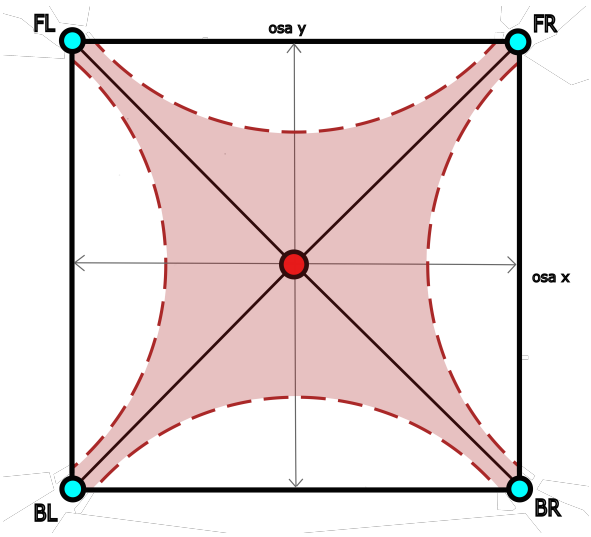
$$BackLeftPosition = \left(\frac{deltaBL}{100}\right) \cdot 2 \cdot MaxPosition \tag{19}$$

$$BackRightPosition = \left(\frac{deltaBR}{100}\right) \cdot 2 \cdot MaxPosition \tag{20}$$

Má to ale své limity. Při testování jsem došel k závěru, že pro nejideálnější dynamické vlastnosti pohybu, které do uspokojivé míry odpovídají dynamice pohybu reálného padákového kluzáku, se maximální poloha vynásobí konstantou dva. Tudíž se lano každého pohonu bude pohybovat v rozmezí $+/-$ dvou otáček bubnu, přičemž se většinová část pohybu, v režimu dojetí na cílovou polohu, bude provádět přitahováním.

5.4 Relativní přizpůsobení a polohování

Navržené řešení je v popsaném stavu schopné polohování, ale při testování jsem zaznamenal negativní vliv, který charakter řešení způsobuje. V extrémech jednotlivých os, kdy pohony tahají přes největší páku, mají problém dojet úplně na kraj zvoleného prostoru pohybu. Zároveň, když pohyb vyžaduje, aby sedačka přejela z blízkosti jednoho pohonu k druhému a kopírovala při pohybu jednu ze stěn konstrukce, má tendenci najet nejdříve do středu a až pak do chtěné pozice. Prostor pohybu tedy nabývá tvaru červené oblasti na obrázku 28.



Obrázek 28: Zobrazení prostoru pohybu bez kompenzace negativního vlivu.

Tento negativní jev kompenzuji pomocí výpočtu, který nejdříve provede rozdíl absolutních hodnot úhlu natočení jednotlivých os. Když je rozdíl největší, tak je poloha v extrému jedné z os. Následně si výsledek převedu, aby nabýval hodnot 0 až 0,5 a přičtu k němu jedna. Tím dostanu proměnnou *relativeAdj*, kterou vynásobím výslednou délku lana. V důsledku toho budou pohony přitahovat větší silou v extrémech os a umožní to polohování lépe využívat prostor pohybu.

$$xyDifference = |tilt.x| - |tilt.y| \tag{21}$$

$$xyRelative = \frac{xyDifference}{100} \tag{22}$$

$$xyRelative = \left| \frac{xyRelative}{2} \right| \tag{23}$$

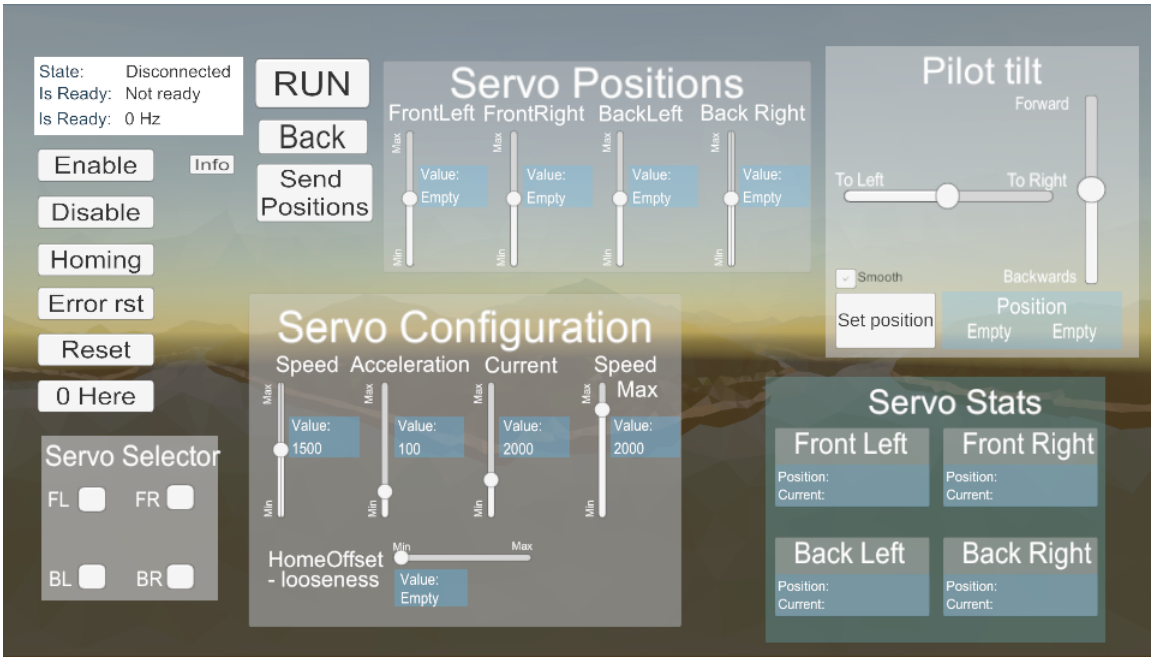
$$relativeAdj = xyRelative + 1 \tag{24}$$

Získaná data v inkrementech následně pošleme těm pohonům, které jsou ve svých operačních sektorech a vynásobíme je proměnnou relativního přizpůsobení. V kódu by to například pro sektor horního „V“ vypadalo následovně a obdobně i pro zbylé sektory, jen s rozdílem proměnných, které reprezentují data pro pozici jednotlivých pohonů. Kompletní řešení funkce je přidáno v příloze nebo je možné se k němu dostat po přihlášení skrze GitLab repozitář[12].

```
1 switch (sector)    // rozdeleni podle sektoru
2     {
3             case ControlSector.UpperVee:
4                     commandDataA1.FrontLeftPosition = ...
                          (Int32)(commandDataA1.FrontLeftPosition * relativeAdj);
5                     commandDataA1.FrontRightPosition = ...
                          (Int32)(commandDataA1.FrontRightPosition * relativeAdj);
6             break;
```

6 Testování navrženého řešení

Testování navrženého řešení probíhalo přes grafické uživatelské rozhraní vytvořené kolegou Richardem Siverou. Spouštění tohoto uživatelského rozhraní probíhá přes vizuální editor Unity a je naprogramovaný pomocí jazyka C#. Původně byl navržen pro testování jednotlivých pohonů, proto jsem ho musel lehce modifikovat. Přidal jsem modul s názvem Pilot tilt, v kterém pomocí dvou posuvníků můžu nastavit výslednou polohu, která odpovídá formátu normalizovaných úhlů ze simulace. Tento modul může pracovat ve dvou režimech funkce. Prvním je statický režim, kterým nastavím statickou výslednou polohu a pomocí tlačítka Set position, tuto polohu zadám do výpočtů a sedačka s pilotem na ní dojede. Druhým z režimů je smooth positioning, pomocí kterého se výsledná pozice mění s každým pohnutím bodu na posuvníku. Můžu tak otestovat dynamické vlastností pohybu, při neustále změně výsledné polohy, jako tomu bude při neustálém toku dat ze simulace. V režimu smooth positioning jsem odhalil negativní vliv polohování, popsaný v kapitole 5.4 a mohl jsem ho kompenzovat. Experimentálním testováním jsem také určil rozmezí pohybu jednotlivých bubnů pohonů, které pro korektní dynamiku pohonu odpovídá +/− dvěma otáčkám, jak je popsáno v kapitole 5.3.[12]



Obrázek 29: Ukázka uživatelského rozhraní pro testování polohování.

Obrázek 30: Ukázka testování náklonu pravým směrem.

Obrázek 31: Ukázka testování náklonu levým směrem.

7 Závěr

Cílem práce bylo navržení platformy pro monitoring polohy sedačky simulátoru padákového kluzáku. Z důvodu postupně probíhajícího vývoje rozdílných částí simulátoru, je náplň výsledné práce současně s monitoringem i představení konceptu řešení polohování sedačky prototypu simulátoru pomocí lanových paralelních robotů.

Nejprve jsem si prostudoval základní principy ovládání reálného padákového kluzáku jako předmětem simulace a stručně jsem je popsal. Dále jsem představil návrh konstrukce a zavěšení, které replikuje to reálné. Následně jsem uvedl principy polohování pomocí konceptu paralelních lanových robotů realizovaný skrze čtyři boční pohony. Pro tyto pohony byly použity servomotory od výrobce Kinco, jejichž funkci a řídicí systém jsem přiblížil na základě znalostí získaných z odborné literatury včetně bakalářských prací na příslušná témata od kolegů, co se podíleli na vývoji konstrukce a řízení pohonů prototypu simulátoru.

Během mého rok dlouhého působení na projektu jsem se nejprve věnoval získání polohy sedačky pomocí senzorických prvků integrovaných v kompletu VR headsetu HTC vive, popsáno v kapitole 3.3. To se ale nakonec ukázalo jako neuspokojivé řešení s nedostačující přesností pro aplikování na simulátoru. Následně z důvodu plánu přenést zjednodušený proces simulace přímo do headsetu bez použití počítače, se postupně přešlo na samostatné zařízení Meta Quest2. Toto zařízení také dokáže snímat polohu svých ovladačů, ale pouze relativní k headsetu. Kvůli neustálému pohybu pilota s nasazeným headsetem v sedačce, tato relativní poloha ztrácí smysl, pro sledování polohy sedačky v simulátoru. Kvůli zmíněným problémům se zrušil mezikrok snímání polohy, která měla být následně využita pro porovnávání s polohou sedačky v simulaci.

Jako senzorické prvky pro určení polohy sedačky v konstrukci simulátoru jsou použité enkodéry jednotlivých servomotorů. Pracovní režim najetí na výchozí pozici určuje středovou výchozí pozici sedačky v konstrukci, od které se odvíjí polohování. Následně jsem vytvořil funkci, která normalizuje data získaná ze simulace a převede je na výslednou polohu, která se pohybuje ve zvoleném souřadnicovém systému. Rozdílem výchozí a výsledné pozice se provádí polohování v každé instanci letu v simulaci. Koncept řešení polohování jsem zobecnil pro vysvětlení principu a uvedl jsem způsoby případného kompenzování vzniklých offsetů pro možnosti použití rozdílných konfigurací posedů a jejich uchycení k pohonům.

Zadáním bylo také experimentálně ověřit navržené řešení, což jsem provedl pomocí upraveného uživatelského rozhraní. Získané informace o vlastnostech polohování jsem následně použil pro navržení výpočtů, které kompenzují vzniklé negativní vlivy. Navržené výpočty jsem následně zpětně aplikoval do řešení polohování sedačky, což mělo za efekt zvýšení přesnosti polohování.

Celkově se domnívám, že je řešení polohování uspokojivé s možnostmi další úpravy pro zpřesnění polohování. Cílem projektu bylo vyvinout co nejvěrohodnější simulaci, díky čemu bylo finální řešení prototypu simulátoru padákového kluzáku rozměrné a se všemi komponenty i velice drahé. To mělo za následek, že se simulátor, jako komplet ukázal být nevhodný pro komerční využití v Beskydské škole létání. Praktické využití zaznamenala pouze zmenšená verze prototypu simulátoru, která neumožňuje polohování a obsahuje pouze řídicí prvky a VR headset. Získané znalosti o pohybu sedačky v konstrukci a koncept polohování budou využity pro vývoj v oblasti lanových robotů a případně pro budoucí zpřesnění řešení polohování. V počátku vývoje je také verze prototypu simulátoru, která k pohybu sedačky bude využívat koncept Stewartovy platformy, řešený pomocí hydraulických zvedáků nebo elektrických lineárních pohonů. Získané znalosti z oblasti řízení elektrických pohonů budou využity a zohledněny při vývoji další verze simulátoru.

Seznam použité literatury

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

[2] R. Sivera, *Pokročilé řízení servomotorů pro simulátor padákového kluzáku, Advanced Control of Servomotors for Paraglider Simulator*. Plzeň, 2021.

[3] P. Stašek, *Návrh hlavních pohonů simulátoru padákového kluzáku, Design of the main drives of the paraglider simulator*. Plzeň, 2021.

[4] J. Skalický, *Elektrické servopohony*, Vyd. 2. Brno: Vysoké učení technické, 2001, ISBN: 80-214-1978-4.

[5] *Kinco automation*. URL: https://www.kincoautomation.com/marketing/servo/.

[6] *History of virtual reality*, 2023. URL: https://www.verdict.co.uk/history-virtual-reality-timeline/.

[7] *HTC Vive PRE User Guide*, -, 2020. URL: https://www.htc.com/managed-assets/shared/desktop/vive/Vive_PRE_User_Guide.pdf.

[8] *Getting started with Meta Quest 2*, -, 2023. URL: https://www.meta.com/help/quest/articles/getting-started/getting-started-with-quest-2/.

[9] *Strain Gauges*, -, 2023. URL: https://www.omega.com/en-us/resources/strain-gages.

[10] *What is quadrature encoding?* -, 2017. URL: https://www.linearmotiontips.com/what-is-quadrature-encoding/.

[11] *C# documentation*, -, 2023. URL: https://learn.microsoft.com/en-us/dotnet/csharp/.

[12] L. Sládek, *GitLab Repository*, Plzeň, 2023. URL: https://gitlab.fel.zcu.cz/parasim/unity.

Příloha A

```
        // zakladni pozice
        private static readonly Vector2 StartFL = new Vector2
        {
            x = -115,
            y = 100
        };

        private static readonly Vector2 StartFR = new Vector2
        {
            x = 115,
            y = 100
        };

        private static readonly Vector2 StartBL = new Vector2
        {
            x = -115,
            y = -144
        };

        private static readonly Vector2 StartBR = new Vector2
        {
            x = 115,
            y = -144
        };

        private const float StartDistanceF = 146;
        private const float StartDistanceB = 184;

        private Vector2 CoordsFL = new Vector2
        {
            x = 0,
            y = 0
        };

        private Vector2 CoordsFR = new Vector2
        {
            x = 0,
            y = 0
        };

        private Vector2 CoordsBL = new Vector2
        {
            x = 0,
            y = 0
        };

        private Vector2 CoordsBR = new Vector2
        {
            x = 0,
```

```
50              y = 0
51          };
52
53          Vector2 _tilt = new Vector2
54          {
55              x = 0,
56              y = 0
57          };
58
59          private ControlSector _sector;
60
61          private float NormalizeAngle(float angle, float maxRealAngle)
62          {
63              if (angle ≤ 180f)
64              {
65                  angle = -angle;
66                  if (angle < -maxRealAngle) angle = -maxRealAngle;
67              }
68              else if(angle > 180f)
69              {
70                  angle = 360 - angle;
71                  if (angle > maxRealAngle) angle = maxRealAngle;
72              }
73
74              return (angle / maxRealAngle) * 100;   // <-100 , 100>
75          }
76
77          public void CalculatePosition2D(float roll, float pitch)
78          {
79              SetDriverDefaults();
80
81              commandDataA1.FrontLeftCommand = (byte)Commands.Torque;
82              commandDataA1.BackLeftCommand = (byte)Commands.Torque;
83              commandDataA1.BackRightCommand = (byte)Commands.Torque;
84              commandDataA1.FrontRightCommand = (byte)Commands.Torque;
85
86              _tilt.x = - NormalizeAngle(roll, 45);     // <-100 , 100>
87              _tilt.y = NormalizeAngle(pitch, 45);    // <-100 , 100>
88
89              if (_tilt.x ≥ 0 && _tilt.y ≥ 0)   // 1. quadrant
90              {
91                  if (_tilt.x > _tilt.y)    // sub-quadrant DOWN
92                  {
93                      commandDataA1.FrontRightCommand = (byte)Commands.Position;
94                      commandDataA1.BackRightCommand = (byte)Commands.Position;
95                      _sector = ControlSector.RightVee;
96                  }
97                  else if (_tilt.x < _tilt.y)   // sub-quadrant UP
98                  {
99                      commandDataA1.FrontLeftCommand = (byte)Commands.Position;
100                     commandDataA1.FrontRightCommand = (byte)Commands.Position;
101                     _sector = ControlSector.UpperVee;
102                 }
```

```
103                 else   // Diagonal
104                 {
105                     commandDataA1.FrontRightCommand = (byte)Commands.Position;
106                     _sector = ControlSector.Diagonal;
107                 }
108             } else if (_tilt.x ≤ 0 && _tilt.y ≥ 0)   // 2. quadrant
109             {
110                 if ((-_tilt.x) > _tilt.y)    // sub-quadrant DOWN
111                 {
112                     commandDataA1.FrontLeftCommand = (byte)Commands.Position;
113                     commandDataA1.BackLeftCommand = (byte)Commands.Position;
114                     _sector = ControlSector.LeftVee;
115                 }
116                 else if ((-_tilt.x) < _tilt.y)   // sub-quadrant UP
117                 {
118                     commandDataA1.FrontLeftCommand = (byte)Commands.Position;
119                     commandDataA1.FrontRightCommand = (byte)Commands.Position;
120                     _sector = ControlSector.UpperVee;
121                 }
122                 else   // Diagonal
123                 {
124                     commandDataA1.FrontLeftCommand = (byte)Commands.Position;
125                     _sector = ControlSector.Diagonal;
126                 }
127             } else if (_tilt.x ≤ 0 && _tilt.y ≤ 0)   // 3. quadrant
128             {
129                 if (_tilt.x > _tilt.y)    // sub-quadrant DOWN
130                 {
131                     commandDataA1.BackRightCommand = (byte)Commands.Position;
132                     commandDataA1.BackLeftCommand = (byte)Commands.Position;
133                     _sector = ControlSector.DownVee;
134                 }
135                 else if (_tilt.x < _tilt.y)   // sub-quadrant UP
136                 {
137                     commandDataA1.FrontLeftCommand = (byte)Commands.Position;
138                     commandDataA1.BackLeftCommand = (byte)Commands.Position;
139                     _sector = ControlSector.LeftVee;
140                 }
141                 else   // Diagonal
142                 {
143                     commandDataA1.BackLeftCommand = (byte)Commands.Position;
144                     _sector = ControlSector.Diagonal;
145                 }
146             }
147             else   // 4. quadrant
148             {
149                 if (_tilt.x < (-_tilt.y))    // sub-quadrant DOWN
150                 {
151                     commandDataA1.BackLeftCommand = (byte)Commands.Position;
152                     commandDataA1.BackRightCommand = (byte)Commands.Position;
153                     _sector = ControlSector.DownVee;
154                 }
155                 else if (_tilt.x > (-_tilt.y))   // sub-quadrant UP
```

```
156                 {
157                     commandDataA1.FrontRightCommand = (byte)Commands.Position;
158                     commandDataA1.BackRightCommand = (byte)Commands.Position;
159                     _sector = ControlSector.RightVee;
160                 }
161                 else   // Diagonal
162                 {
163                     commandDataA1.BackRightCommand = (byte)Commands.Position;
164                     _sector = ControlSector.Diagonal;
165                 }
166             }
167 
168             CoordsFR.x = StartFR.x - _tilt.x;
169             CoordsFR.y = StartFR.y - _tilt.y;
170             float distanceFR = (float) Math.Sqrt((CoordsFR.x * CoordsFR.x) + ...
                    (CoordsFR.y * CoordsFR.y));
171             float ΔFR = StartDistanceF - distanceFR;
172             commandDataA1.FrontRightPosition = (Int32)((ΔFR / 100f) * 2 * ...
                    MaxPosition);    // Distance in INC units
173 
174             CoordsFL.x = - (StartFL.x - _tilt.x);
175             CoordsFL.y = (StartFL.y - _tilt.y);
176             float distanceFL = (float) Math.Sqrt((CoordsFL.x * CoordsFL.x) + ...
                    (CoordsFL.y * CoordsFL.y));
177             float ΔFL = StartDistanceF - distanceFL;
178             commandDataA1.FrontLeftPosition = (Int32)((ΔFL / 100f) * 2 * ...
                    MaxPosition);
179 
180             CoordsBL.x = - (StartBL.x - _tilt.x);
181             CoordsBL.y = - (StartBL.y - (_tilt.y * 0.8f));
182             float distanceBL = (float) Math.Sqrt((CoordsBL.x * CoordsBL.x) + ...
                    (CoordsBL.y * CoordsBL.y));
183             float ΔBL = StartDistanceB - distanceBL;
184             commandDataA1.BackLeftPosition = (Int32)((ΔBL / 100f) * 2 * ...
                    MaxPosition);
185 
186             CoordsBR.x = (StartBR.x - _tilt.x);
187             CoordsBR.y = - (StartBR.y - (_tilt.y * 0.8f));
188             float distanceBR = (float) Math.Sqrt((CoordsBR.x * CoordsBR.x) + ...
                    (CoordsBR.y * CoordsBR.y));
189             float ΔBR = StartDistanceB - distanceBR;
190             commandDataA1.BackRightPosition = (Int32)((ΔBR / 100f) * 2 * ...
                    MaxPosition);
191 
192             var xyDifference = Math.Abs(_tilt.x) - Math.Abs(_tilt.y);
193             var xyRelative = xyDifference / 100f;
194             xyRelative = Math.Abs(xyRelative / 2);
195             var relativeAdj = xyRelative + 1f;
196 
197 
198             switch (_sector)
199             {
200                 case ControlSector.RightVee:
```

```
201                         commandDataA1.BackRightPosition = ...
                                (Int32)(commandDataA1.BackRightPosition * relativeAdj);
202                         commandDataA1.FrontRightPosition = ...
                                (Int32)(commandDataA1.FrontRightPosition * relativeAdj);
203                         break;
204                     case ControlSector.UpperVee:
205                         commandDataA1.FrontLeftPosition = ...
                                (Int32)(commandDataA1.FrontLeftPosition * relativeAdj);
206                         commandDataA1.FrontRightPosition = ...
                                (Int32)(commandDataA1.FrontRightPosition * relativeAdj);
207                         break;
208                     case ControlSector.LeftVee:
209                         commandDataA1.FrontLeftPosition = ...
                                (Int32)(commandDataA1.FrontLeftPosition * relativeAdj);
210                         commandDataA1.BackLeftPosition = ...
                                (Int32)(commandDataA1.BackLeftPosition * relativeAdj);
211                         break;
212                     case ControlSector.DownVee:
213                         commandDataA1.BackRightPosition = ...
                                (Int32)(commandDataA1.BackRightPosition * relativeAdj);
214                         commandDataA1.BackLeftPosition = ...
                                (Int32)(commandDataA1.BackLeftPosition * relativeAdj);
215                         break;
216                     case ControlSector.Diagonal:
217                         // For future use?
218                         break;
219                 }
220
221             }
```