Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Prototypovanie riadenia dronov v zdieľanej webovej virtuálnej realite

Diplomová práca

2021 **Bc. Dávid Gula**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Prototypovanie riadenia dronov v zdieľanej webovej virtuálnej realite

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Štefan Korečko PhD.
Konzultant:	

Košice 2021 **Bc. Dávid Gula**

Abstrakt v SJ

Táto práca je zameraná na vytvorenie virtuálneho prostredia, v ktorom je možné testovať a overovať funkčnosť prototypov riadiacich systémov pre drony. Jedná sa o zdieľané prostredie podporujúce viacerých pripojených používateľov v rovnakom čase. Navrhli a implementovali sme modulárny komponent realizujúci simuláciu a autonómne ovládanie kvadrokoptéry, s ohľadom na bezpečnosť používateľov. Ten pozostáva z niekoľkých navzájom prepojených modulov, ktoré je možné v prípade potreby jednoducho meniť. Komponent sme vytvorili ako rozšírenie pre existujúcu platformu *LIRKIS G-CVE*. Použiteľnosť výsledného systému sme overili pomocou používateľského testovania a dotazníka. Testovanie sme realizovali v scéne, ktorá predstavovala miesto prebiehajúcej stavby, v ktorom dron roznášal stavebný materiál.

Kľúčové slová v SJ

A-Frame, Networked-Aframe, virtuálna realita, zdieľané prostredie, prototypovanie, ovládanie dronov

Abstrakt v AJ

This work is focused on creating a virtual environment in which it is possible to test and verify the functionality of prototype control systems for drones. It is a shared environment that supports multiple connected users at the same time. We designed and implemented a modular component implementing the simulation and autonomous control of a quadcopter with regard to user safety. It consists of several interconnected modules, which can be easily changed if necessary. We created the component as an extension for the existing *LIRKIS G-CVE* platform. We verified the usability of the resulting system using user testing and a questionnaire. We carried out the testing in a scene that represented the site of ongoing construction in which the drone was distributing the building material.

Kľúčové slová v AJ

A-Frame, Networked-Aframe, virtual reality, shared environment, prototyping, drone control

Bibliografická citácia

GULA, Dávid. *Prototypovanie riadenia dronov v zdieľanej webovej virtuálnej realite*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2021. 65s. Vedúci práce: Ing. Štefan Korečko PhD.

61378

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**

Študijný program: **Informatika**

Názov práce:

Prototypovanie riadenia dronov v zdieľanej webovej virtuálnej realite

Drone Control Prototyping in Shared Web Virtual Reality

Študent: **Bc. Dávid Gula**

Školiteľ: **Ing. Štefan Korečko, PhD.**

Školiace pracovisko: **Katedra počítačov a informatiky**

Konzultant práce:

Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

1. Analyzovať platformu LIRKIS G-CVE z hľadiska jej použitia pre prototypovanie riadenia dronov.
2. Analyzovať súčasné riešenia zdieľanej webovej virtuálnej reality a ich využitie.
3. Analyzovať súčasné prístupy a dostupné riešenia pre realizáciu simulácie dronov a ich interakcie s používateľmi na platforme LIRKIS G-CVE.
4. Experimentálne overiť prepojenie formálne vyvinutého riadiaceho softvéru autonómneho systému so zdieľaným virtuálnym prostredím na platforme LIRKIS G-CVE.
5. Na základe predchádzajúcich bodov navrhnúť a implementovať zdieľané virtuálne prostredie pre evaluáciu prototypov riadenia dronov.
6. Implementované prostredie vyhodnotiť z hľadiska spoľahlivosti a použiteľnosti.
7. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 23.04.2021

Dátum zadania diplomovej práce: 30.10.2020

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY -12-

12

.....

prof. Ing. Liberios Vokorokos, PhD.

dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 23.4.2021 .....

Vlastnoručný podpis

Poďakovanie

Na tomto mieste by som sa chcel poďakovať svojmu vedúcemu práce za jeho čas, trpezlivosť, odborné vedenie a rady počas riešenia mojej záverečnej práce.

Ďalej by som sa chcel poďakovať svojim rodičom za ich neustálu podporu a povzbudzovanie počas celého môjho štúdia a riešenia tejto práce.

Nakoniec by som chcel vyjadriť svoju úprimnú vďaku aj svojim priateľom, ktorí mi boli silnou oporou počas štúdia a tvorby tejto diplomovej práce.

Obsah

Úvod **1**

1 Analýza platformy LIRKIS G-CVE **3**
1.1 Tvorba virtuálnych prostredí v A-Frame 5

2 Analýza súčasných riešení v oblasti zdieľaných virtuálnych prostredí **7**
2.1 Riešenia založené na A-Frame 7
2.1.1 Webovo-orientovaný nástroj na tvorbu obsahu vo VR 7
2.1.2 Circles: VR platforma pre e-learning 8
2.2 Riešenia založené na Unity 9
2.2.1 SimCEC 9
2.2.2 Virtual Family Room 11
2.2.3 CollaVRLap 12
2.2.4 Viac-používateľské univerzitné prostredie 13
2.2.5 Megacity 14
2.2.6 Kolaboratívne VR prostredie pre návrh produktov 15
2.3 Záver analýzy 16

3 Analýza súčasných prístupov a dostupných riešení pre realizáciu simulácie dronov **18**
3.1 Manuálne ovládanie dronov a simulátory 18
3.2 Autonómne riadené drony a softvér autopilota 19
3.3 Analýza dostupných riešení pre realizovanie simulácie 21
3.3.1 Simulátor DroneVR 21
3.3.2 AirSim 24
3.3.3 Flightmare 25
3.3.4 Ďalšie menšie open-source simulátory 26
3.4 Záver analýzy 26

4 Integrácia riadiaceho softvéru so zdieľaným virtuálnym prostredím 29
4.1 Komponent pre komunikáciu medzi entitami 29
4.2 Networked-Aframe knižnica 29
4.3 Model ovládača pre cBot-a a generovanie kódu 30
4.4 Nasadenie ovládača do virtuálnej scény 31

5 Návrh a implementácia virtuálneho prostredia pre evaluáciu prototypov riadenia dronov 35
5.1 Typ modelovaného dronu a spôsob jeho ovládania 35
5.2 Štruktúra ovládacích komponentov kvadrokoptéry 38
5.3 Implementácia fyziky a fyzikálneho modelu 40
5.3.1 Výber a použitie vhodného fyzikálneho rámca 42
5.3.2 Realizácia fyzikálneho modelu 43
5.4 Stabilizácia kvadrokoptéry 45
5.4.1 Jednoduchý stabilizátor 46
5.4.2 Stabilizátor využívajúci cieľové pozície 49
5.5 Modul pre spracovanie ovládania pomocou klávesnice 49
5.6 Modul pre automatické riadenie využitím riadiaceho systému pre kvadrokoptéru 50
5.7 Hlavný komponent pre entitu kvadrokoptéry 53
5.8 Komponent bodov záujmu pre drona 55

6 Zhodnotenie implementovaného systému 57

7 Záver 60

Literatúra 62

Zoznam príloh 66

Zoznam obrázkov

1.1 Príklad zdieľaného virtuálneho prostredia vytvoreného v systéme *LIRKIS G-CVE* [2] 4

2.1 Architektúra systému [5] 8
2.2 Príklad používania systému pomocou grafického rozhrania [5] . . 8
2.3 Jedna z virtuálnych miestností systému *Circles* [6] 9
2.4 Moduly systému *SimCEC* [7] 10
2.5 Rôzne typy úloh vykonávaných počas simulácie [7] 10
2.6 Virtuálna tabuľa umožňujúca používateľom kresliť vo virtuálnom priestore [8] 11
2.7 Rôzne aktivity implementované v systéme [8] 12
2.8 Simulácia rezania tkaniva orgánu v systéme CollaVRLap [9] 13
2.9 Simulovaná evakuácia vo virtuálnom prostredí univerzity [10] . . . 14
2.10 Simulácia požiaru budovy v systéme Megacity [11] 14
2.11 Menu zobrazené ľavým ovládačom [12] 15
2.12 Práca s objektmi vo virtuálnom prostredí [12] 16

3.1 Prostredie simulátora *DroneVR* [14] 22
3.2 Architektúra systému *AirSim* [22] 24
3.3 Architektúra systému Flightmare [23] 25

4.1 Špecifikácia ovládača v jazyku B [32] 30
4.2 Jednotlivé sektory senzorov pre snímanie okolia robota [32] 33
4.3 Rozhranie ovládača a jeho vstupné a výstupné hodnoty [32] 33
4.4 Reprezentácia uhlov v *A-Frame* (a) a reprezentácia uhlov ovládača (b) 34

5.1 Konfigurácia a smer rotácie motorov modelovanej kvadrokoptéry (a), fyzikálne sily generované motormi kvadrokoptéry (b) 37

5.2 Ovládanie pohybu kvadrokoptéry. Zelená farba predstavuje zvýšenie, červená farba zníženie výkonu daných motorov. Pohyb smerom nahor (a), nadol (b), dopredu (c) a dozadu (d) 37
5.3 Ovládanie pohybu kvadrokoptéry. Zelená farba predstavuje zvýšenie, červená farba zníženie výkonu daných motorov. Pohyb smerom doľava (a), doprava (b), rotácia vľavo (c) a vpravo (d) 38
5.4 Hierarchia jednotlivých ovládacích komponentov 39
5.5 Konceptuálny diagram riadiaceho systému kvadrokoptéry 40
5.6 Diagram tried implementovaného systému 41
5.7 Smer aplikovania síl krútiaceho momentu (a) a vztlakovej sily (b) na model kvadrokoptéry vzhľadom na osi vo virtuálnom prostredí 44
5.8 Štruktúra PID regulátora so spätnou väzbou 45
5.9 Diagram tried zobrazujúci štruktúru implementovaných stabilizátorov 48
5.10 Diagram aktivít zobrazujúci proces riadenia drona pomocou ovládača pre cBot-a 51
5.11 Diagram aktivít zobrazujúci proces riadenia drona pomocou ovládača navrhnutého pre kvadrokoptéru 56

6.1 Vytvorené virtuálne prostredie použité pri vyhodnotení riešenia . . 57
6.2 Výsledné hodnoty SUS skóre jednotlivých používateľov 59

Zoznam tabuliek

3.1 Tabuľka so zoznamom analyzovaných systémov pre implementácie fyzikálneho modelu 27

Úvod

Virtuálna realita je oblasť, ktorá existuje už dlhšiu dobu, no až v poslednom desaťročí sa výrazne rozširuje a ukazuje svoj potenciál v mnohých oblastiach. Najčastejšie sa jedná o oblasti medicíny (rehabilitácie, liečenie fóbií, svalových a nervových porúch, atď.), vzdelávania a trénovania na diaľku (vzdelávanie žiakov a študentov, trénovanie zamestnancov, pracovníkov a lekárov, simulácia nešťastí atď.), ale aj prototypovanie a návrh, kde dokáže ušetriť veľké množstvo financií a času, ktoré by boli inak potrebné pre tvorbu fyzických prototypov a testovanie produktov. Vďaka pohlcujúcemu zážitku a miere interaktivity, ktorú táto technológia ponúka, je veľmi pravdepodobné, že sa bude aj naďalej rozvíjať závratným tempom a ovplyvňovať aj predtým nedotknuté oblasti.

Jednou z týchto oblastí je aj testovanie prototypov riadenia autonómnych zariadení. Technológie majú za cieľ zjednodušovať náš život a asistovať nám pri dennodenných činnostiach. Príkladom sú robotické vysávače a kosačky, ktoré dokážu vykonávať určité činnosti úplne samostatne. Avšak predtým, ako môžu byť použité v praxi, je nutné ich otestovať, či fungujú správne a sú bezpečné pre používateľov.

Fyzické testovanie je náročné, vyžaduje si vytvoriť reálny prototyp produktu a overovať ho fyzicky, za pomoci prístrojov a prítomnosti odborníkov. Virtuálna realita v tomto smere ponúka možnosť vykonať toto testovanie vo virtuálnom prostredí, ktoré je možné jednoducho nakonfigurovať podľa potreby, vytvoriť rôzne scenáre a overovať tak funkčnosť prototypu. To všetko je možné realizovať bez nutnosti tvorby reálneho stroja a hľadania vhodného prostredia. Takéto prostredia ale nedokážu simulovať všetky aspekty reálneho sveta, preto nie je možné reálne testovanie úplne nahradiť. V závislosti od reálnosti simulácie je nutné vykonať menšie, či väčšie úpravy pred nasadením prototypu do prevádzky.

Jednou zo zaujímavých oblastí, ktorá v dnešnej dobe zažíva rozsiahlu expanziu, sú komerčné drony. Tieto stroje sú dostupné pre široký okruh ľudí, v rôznych veľkostiach, rozmeroch a pre rôzne použitia. Vďaka ich dostupnosti a jednoduchosti je cena nízka, čo znamená, že sa ich počet rapídne zvyšuje. Zvláštnu po-

zornosť si zaslúžia autonómne drony, ktoré sa už dnes používajú na prieskum, fotografovanie a natáčanie scenérií, či donášku balíčkov. Rovnako, ako pri iných autonómnych strojoch, aj tieto je nutné pred ich nasadením testovať. Avšak v reálnej prevádzke hrozí, že nesprávna činnosť zariadenia spôsobí materiálne škody, v ojedinelých prípadoch dokonca zranenia.

Pomocou virtuálnej reality dokážeme testovať funkčnosť a bezpečnosť takýchto zariadení bez akýchkoľvek rizík a s ľubovoľne konfigurovateľným prostredím. Navyše nám rôzne softvérové rámce a knižnice umožňujú vytvoriť zdieľané virtuálne prostredia, do ktorých sa môžu pripojiť viacerí používatelia naraz a v reálnom čase navzájom spolupracovať. Ide o príklad kolaboratívneho virtuálneho prostredia (CVE), v ktorom ľudia spolupracujú a navzájom medzi sebou interagujú [1].

Formulácia úlohy a štruktúra práce

Ešte predtým, ako začneme s návrhom a implementáciou systému, je nutné získať prehľad o riešenej problematike. Na začiatok analyzujeme použiteľnosť platformy *LIRKIS G-CVE*, ktoré vzniklo u nás na katedre [2], pre tvorbu zdieľaného virtuálneho prostredia, v ktorom by bolo možné overovať prototypy riadenia dronov.

Ďalším cieľom analýzy je preskúmať existujúce riešenia v oblasti zdieľaných virtuálnych prostredí, najmä z hľadiska použitých technológií, využitia systémov a ich poskytnutej funkcionality. Taktiež vykonáme prieskum systémov, ktoré simulujú virtuálneho lietajúceho drona, aby sme zistili, akým spôsobom je možné vytvoriť verný a hlavne realistický model takéhoto stroja.

Po vykonaní analýzy experimentálne overíme prepojenie a integráciu prototypu riadiaceho softvéru s virtuálnym zdieľaným prostredím na báze platformy *LIRKIS G-CVE*. Konkrétne sa jedná o model autonómneho riadenia robotického vysávača, ktorý reaguje na používateľov vo svojom okolí.

Následne navrhneme a implementujeme modulárny systém simulácie a riadenia drona vo virtuálnom prostredí s ohľadom na aktuálne pripojených používateľov vo svojom okolí. Tento systém nasadíme do prostredia vytvoreného pomocou platformy *LIRKIS G-CVE* a pripravíme relevantný scenár pre drona, ktorý by odzrkadľoval jedno z jeho reálnych využití - prenášanie objektov medzi pozíciami.

Na konci overíme použiteľnosť nášho systému vykonaním experimentu s používateľmi, ktorí sa budú pohybovať v prostredí a interagovať s dronom.

1 Analýza platformy LIRKIS G-CVE

Platforma *LIRKIS G-CVE*, opísaná v článku [2], predstavuje imersívne zdieľané virtuálne prostredie, podporujúce pripojenie viacerých používateľov v reálnom čase. Je vytvorené na základe voľne dostupného rámca *A-Frame* [3] a jeho rozšírenia - *Networked-Aframe* [4]. *A-Frame* je webový nástroj na tvorbu prostredí a systémov pre virtuálnu realitu. Funguje na báze technológie *WebVR*, ktorú ale postupne nahradzuje *WebXR*. Tá navyše podporuje aj zariadenia pre rozšírenú realitu.

A-Frame je spúšťaný priamo vo webovom prehliadači, z čoho vyplýva rozsiahla podpora rôznych zariadení, od počítačov a notebookov cez tablety a smartfóny, až po rôzne prilby pre virtuálnu a rozšírenú realitu. Architektúra *A-Frame* je postavená na entitno-komponentovom softvérovom architektonickom vzore, ktorý poskytuje vysokú flexibilitu pri vytváraní aplikácii a ich rozšírení. Každá entita môže obsahovať viacero znovu použiteľných komponentov, ktoré je možné kombinovať podľa potreby za účelom vytvorenia finálneho objektu.

Komunikácia medzi systémom a používateľmi je založená na klient-server architektúre. Hlavnú časť systému *LIRKIS G-CVE* predstavuje webový server, ktorý je zodpovedný za manažment komunikácie a šírenie informácii o entitách medzi používateľmi. Implementovaný je využitím troch softvérových rámcov:

- *Node.js* - poskytuje funkcionalitu pre prepojenie klientov v reálnom čase a manažuje asynchrónnu komunikáciu medzi klientom a serverom
- *Express.js* - obstaráva HTTP požiadavky od klientov
- *Networked-Aframe* - synchronizuje a distribuuje atribúty komponentov a entít vo virtuálnom prostredí

Networked-Aframe slúži na sprostredkovanie a distribúciu informácií o zdieľaných entitách a ich atribútoch vo virtuálnom prostredí. Obstaráva komunikáciu medzi pripojenými používateľmi a ostatnými entitami. Prostredníctvom servera a prenosového protokolu sú najčastejšie distribuované atribúty pozícia, rotácia a

zmeny mierky. Takto je možné jednoducho vytvoriť virtuálne prostredie, do ktorého sa môže pripojiť viacero používateľov a navzájom ovplyvňovať seba alebo spoločné prostredie.

Samotná komunikácia medzi klientom a serverom je založená na *WebSocket* protokole. Jednotlivé virtuálne prostredia sú dostupné na vlastnej špecifickej URL adrese. Server spracováva všetky dáta klientov a propaguje ich prostredníctvom všetkých aktívnych spojení. Objem prenesených dát závisí od zložitosti jednotlivých entít a ich počte v prostredí.

Obr. 1.1: Príklad zdieľaného virtuálneho prostredia vytvoreného v systéme *LIRKIS G-CVE* [2]

Interakciu používateľa s prostredím zabezpečuje rozhranie integrované v *A-Frame*, ktoré pozostáva z viacerých komponentov, z ktorých každý spracováva údaje z rôznych typov ovládačov a vstupných zariadení. Avšak toto rozhranie výkonnostne nepostačovalo. Preto autori platformy *LIRKIS G-CVE* vytvorili ako náhradu rozhranie *Smart-client* [2], ktoré dramaticky zvyšuje plynulosť a výkonnosť klientského rozhrania a znižuje čas odozvy medzi vstupom a vizuálnym výstupom. Taktiež umožňuje využiť smartfóny a tablety na interakciu s virtuálnym prostredím. Týmto spôsobom sa môžu do virtuálnej scény zapojiť aj používatelia využitím svojho chytrého zariadenia, ktoré majú stále pri sebe.

Platforma *LIRKIS G-CVE* poskytuje možnosť jednoducho a rýchlo vytvoriť zdieľané virtuálne prostredie podporujúce veľké množstvo zariadení, pomocou ktorých sa môžu klienti pripojiť odkiaľkoľvek a interagovať medzi sebou. Tieto výhody vyplývajú predovšetkým zo spomínaných technológií, ktoré túto funkcionalitu obstarávajú. Implementáciou *Smart-client* rozhrania sa navyše podarilo

výrazne zlepšiť používateľský zážitok a kvalitu interakcie s ostatnými používateľmi a virtuálnymi objektmi.

Jednou z ďalších výhod rámca *A-Frame*, a teda aj platformy *LIRKIS G-CVE*, je možnosť jednoducho pridávať ďalšiu funkcionalitu pomocou komponentov vytvorených inými používateľmi. Napríklad pre potreby simulovania fyzikálneho pohybu a kolízií existuje rozšírenie `aframe-physics-system`, ktoré používa samostatný fyzikálny rámec a integruje ho do prostredia. Podporuje dva fyzikálne rámce - *Cannon.js* a *Ammo.js*.

Rámec *Ammo.js* je pôvodne fyzikálny rámec *Bullet*, ktorý obsahuje veľmi prepracované a realistické simulovanie fyziky. *Bullet* je vytvorený v jazyku *C++*, vďaka čomu je výkonný a veľmi dobre optimalizovaný. Existuje taktiež aj verzia *PyBullet*, použiteľná v jazyku *Python*. Obe verzie sú open-source. *Ammo.js* vznikol prekladom zdrojového kódu *Bullet* rámca, v dôsledku čoho nejakú dobu trvá, kým *Ammo.js* prevezme najnovšie zmeny. Taktiež nie všetka funkcionalita je z pôvodného rámca dostupná. V porovnaní s *Cannon.js* je tento rámec oveľa zložitejší a náročnejší na výpočtový výkon, ale poskytuje možnosť simulovať zložitejšie typy objektov. Veľkou nevýhodou *Ammo.js* je absencia návodov a dokumentácie. Práca s ním je náročná, keďže je nutné hľadať si informácie o metódach, triedach a rozhraniach prostredníctvom dokumentácie pre fyzikálny rámec *Bullet*.

Rámec *Cannon.js* je oproti tomu vytvorený od základov v jazyku *Javascript*, čo jeho autorovi dovolilo využiť prvky dostupné v tomto jazyku a optimalizovať ho pre použitie vo webovom prostredí. Výsledkom toho je malý a výkonný fyzikálny rámec, ktorý obsahuje funkcionalitu postačujúcu pre väčšinu prípadov použitia. Jeho veľkou výhodou v porovnaní s *Ammo.js* je podrobná dokumentácia a niekoľko príkladov, kde sú vysvetlené možnosti a funkcie rámca. Preto sme si na simuláciu zvolili *Cannon.js*.

Vzhľadom na spomínané výhody a možnosti rozšírenia je táto platforma vhodná na vytvorenie virtuálneho prostredia, v ktorom možno testovať interakciu reálnych používateľov s prototypmi zariadení.

1.1 Tvorba virtuálnych prostredí v A-Frame

Keďže sme si zvolili systém implementovať na spomínanej platforme, musíme si byť vedomí niekoľkých obmedzení pri práci s rámcom *A-Frame*. Proces tvorby zdieľaných virtuálnych prostredí v ňom a implementácia riešení je relatívne rýchla a jednoduchá v porovnaní s komplikovanejšími a rozsiahlejšími rámcami, ako sú napríklad *Unity*, alebo *Unreal Engine*. Na druhú stranu tieto rozsiahle herné rámce

poskytujú robustné nástroje pre prácu s virtuálnou scénou, ladenie, prácu s virtuálnymi objektmi atď. Počas implementácie robotického vysávača sme objavili niekoľko nedostatkov a problémov, ktoré sťažujú proces tvorby takýchto prostredí využitím rámca *A-Frame* v porovnaní s vyššie spomínanými možnosťami.

Prvým obmedzením *A-Frame* je, že sa jedná o webový rámec a teda neexistuje žiadna vyhradená aplikácia (editor), v ktorej by bol realizovaný vývoj a implementácia. Existuje však inšpektor scény, ktorý nám umožňuje sledovať virtuálne objekty v scéne, ich atribúty a priamo ich v ňom upravovať. Výsledok zmien vidíme okamžite, avšak pre zachovanie vykonaných zmien je potrebné zmenené atribúty skopírovať a prepísať v kóde. Taktiež pridávanie nových objektov do scény je zložitejšie. V inšpektore vieme vytvoriť iba základnú entitu, ktorej musíme pridávať komponenty a nastavovať atribúty ručne. Pri zložitejších objektoch je rýchlejšie prejsť do kódu a vytvoriť entitu tam. Ak ale chceme vytvoriť viacero rovnakých objektov, môžeme ich kopírovať priamo v inšpektore. *Unity* a *Unreal Engine* sú v spomínaných ohľadoch prívetivejšie, napríklad dizajnéri a grafici môžu navrhovať scénu a pracovať s objektmi v nej bez zasahovania do kódu.

Ďalším obmedzením implementácie v *A-Frame* je náročnosť ladenia kódu. Rámec v podstate nijako neumožňuje ladiť kód pomocou vlastných prostriedkov, sme teda odkázaní na ladenie programu v danom webovom prehliadači, čo nie je ideálne. Samozrejme môžeme použiť nástroje ponúkané rôznymi vývojovými prostrediami, napríklad *WebStorm* od spoločnosti *Jetbrains*, alebo po doinštalovaní rozšírení aj *Sublime Text*, či *Visual Studio Code*. V takom prípade sme však nútení mať implementáciu spustenú na lokálnom počítači. V prípade, ak chceme používať napríklad platformu *Glitch*, ktorá predstavuje najrýchlejší a najjednoduchší spôsob, ako začať pracovať s *A-Frame*, nemáme inú možnosť, než použiť vstavané nástroje vo webovom prehliadači.

2 Analýza súčasných riešení v oblasti zdieľaných virtuálnych prostredí

Našim cieľom je zistiť, aké technológie, knižnice a softvérové rámce sú v dnešnej dobe používané na tvorbu zdieľaných virtuálnych prostredí. Pre nás sú najzaujímavejšie tie, ktoré sú implementované pomocou rámca *A-Frame*. Okrem toho sa zaujímame o zámer a využitie týchto prostredí, ako aj poskytovanú funkcionalitu pre používateľov.

2.1 Riešenia založené na A-Frame

Obe analyzované riešenia pracujú s rovnakým rámcom a technológiami, aké využíva platforma *LIRKIS G-CVE*. Každý zo systémov sa špecializuje na inú oblasť a ukazuje tak dve rôzne možnosti využitia zdieľaných virtuálnych prostredí.

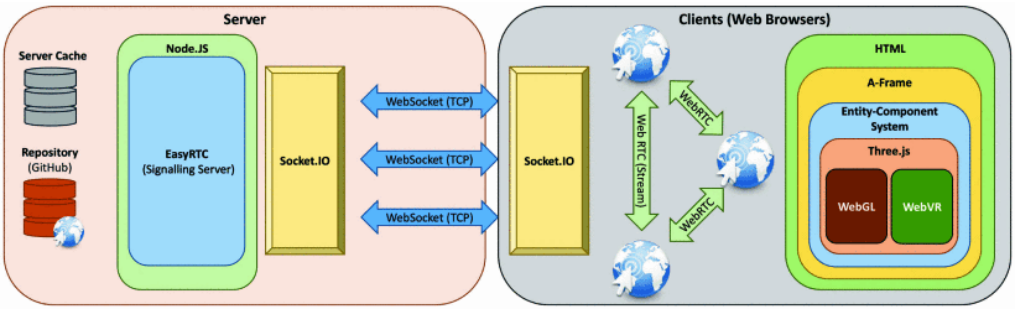
2.1.1 Webovo-orientovaný nástroj na tvorbu obsahu vo VR

Systém opísaný v článku [5] umožňuje navrhovať a tvoriť virtuálne svety prostredníctvom virtuálnej reality. Podnetom na jeho vznik bola neexistencia takýchto všeobecne akceptovaných nástrojov. Navyše systém funguje priamo v prostredí internetového prehliadača.

Používatelia sú navzájom prepojení pomocou hybridnej peer-to-peer (P2P) siete. Jedná sa o kombináciu tradičnej klient-server architektúry, zväčša využívanej na ukladanie a perzistenciu údajov, a P2P siete slúžiacej na synchronizáciu klientov.

Centrálny server udržiava zoznam aktívnych klientov a distribuuje počiatočnú konfiguráciu aktuálnej 3D scény. Implementovaný je pomocou rámcov *Node.js* a *EasyRTC*. Prepojenie servera a klienta je implementované pomocou knižnice *Socket.IO*, *WebRTC* protokol zabezpečuje synchronizáciu klientov v reálnom čase.

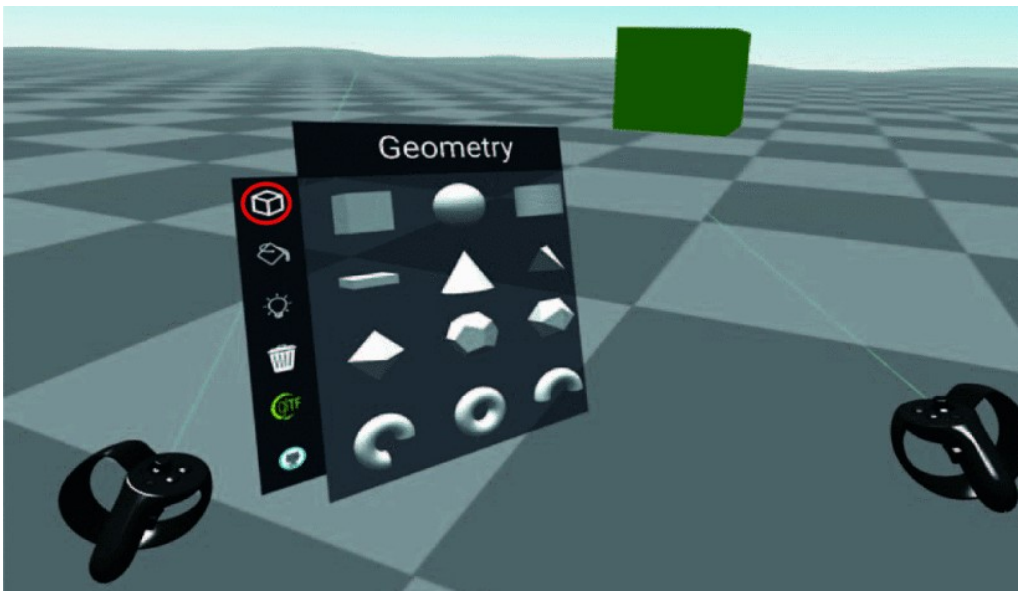
Klientska časť je implementovaná pomocou rámca *A-Frame*, vďaka čomu systém podporuje klasické head mounted displays (HMD), ako napríklad *Oculus*



Obr. 2.1: Architektúra systému [5]

Rift a *HTC Vive*, ale aj mobilné zariadenia. Knižnica *Networked-Aframe* zabezpečuje podporu pre viacerých používateľov v spoločnom prostredí a taktiež poskytuje manažment vlastníctva, pomocou ktorého je zamedzená súbežná modifikácia rovnakého objektu dvoma rôznymi používateľmi.

Používatelia môžu objektom meniť ich veľkosť, pozíciu, rotáciu a farbu. Pre voľbu činností v prostredí sa využíva grafické rozhranie. Systém podporuje jednoduché tvary, ale aj importované modely prostredníctvom repozitára Github.



Obr. 2.2: Príklad používania systému pomocou grafického rozhrania [5]

2.1.2 Circles: VR platforma pre e-learning

Systém *Circles* umožňuje vzdelávať na diaľku viacerých používateľov súčasne vo virtuálnom prostredí [6]. Použitie virtuálnej reality vo vzdelávaní nie je nový koncept, avšak väčšina riešení nepodporuje klasické zariadenia (počítače, mobily a pod.).

Vytvorený je podobne, ako predošlý systém, pomocou rámca *A-Frame* a technológie *WebVR*. Podporu kolaborácie a pripojenia viacerých používateľov zabezpečujú knižnice *Networked-Aframe* a *Mozilla HUB*. Všetky informácie o účtoch (avatar používateľa a jeho nastavenia) sa ukladajú do *MongoDB* databázy. Server je vytvorený pomocou rámca *Node.js*.

Zaujímavosťou tohto systému je, že každý používateľ má svoj vlastný účet, možnosť prispôsobiť si svojho avatara a vytvoriť si spoločnú miestnosť.

Obr. 2.3: Jedna z virtuálnych miestností systému *Circles* [6]

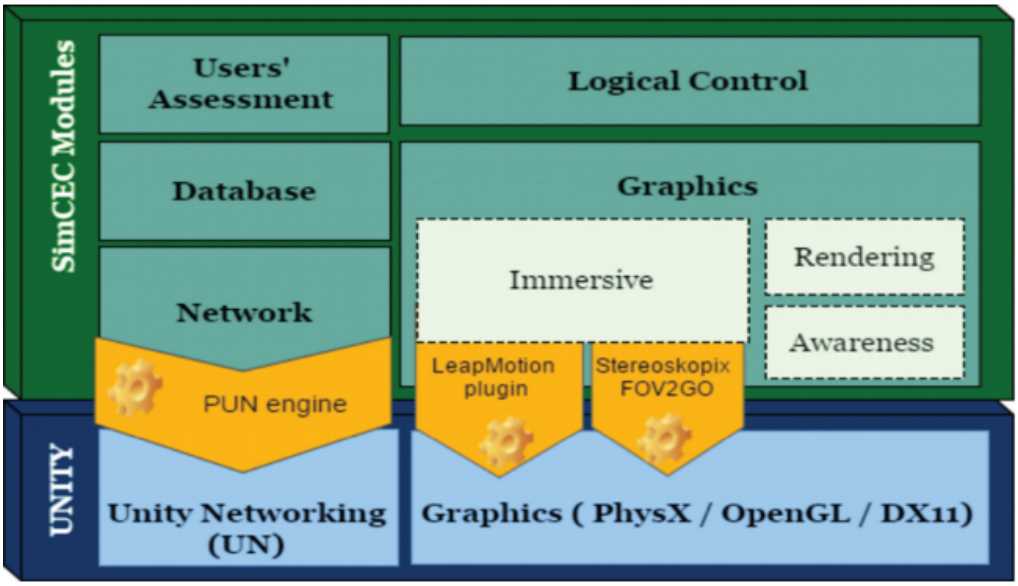
2.2 Riešenia založené na Unity

Všetky systémy spomínané v tejto sekcii vyžadujú inštaláciu herného rámca *Unity*, aby ich bolo možné spustiť. V porovnaní s predchádzajúcimi systémami tieto nie je možné jednoducho spustiť v kompatibilnom prehliadači. *Unity* podporuje možnosť exportovať projekt pre webovú platformu, avšak ani jeden z uvedených systémov nie je na to navrhnutý.

2.2.1 SimCEC

Úlohou tohto systému je trénovanie študentov medicíny, predovšetkým so zameraním na chirurgiu [7]. Umožňuje viacerým študentom pripojiť sa do rovnakého prostredia a hodnotiť ich výkon pri vykonávaní rôznych úkonov vo virtuálnom prostredí. Pomocou detailných metrík hodnotí ako výkon jednotlivcov, tak aj celého tímu.

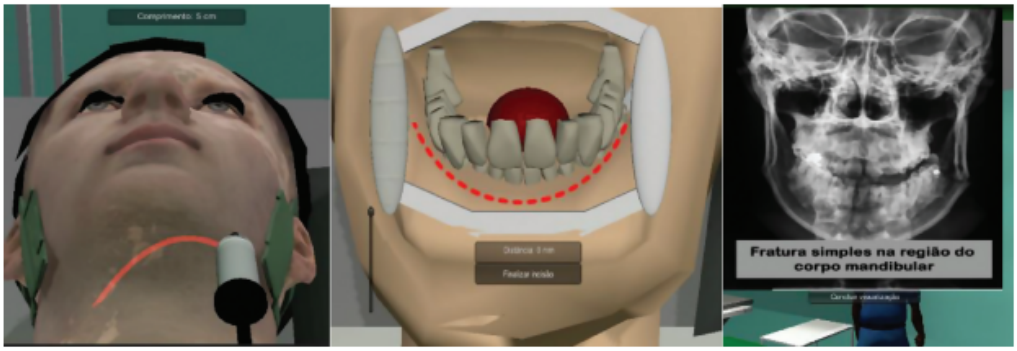
Autori sa rozhodli použiť herný rámec *Unity* najmä kvôli flexibilite vývoja distribuovaných virtuálnych prostredí a širokej ponuke dostupných rozšírení. Modely vytvorili buď pomocou softvéru *Blender*, alebo ich získali z internetu bezplatne.



Obr. 2.4: Moduly systému *SimCEC* [7]

Systém má dve verzie: lokálnu (*Local SimCEC*) a sieťovú (*SimCEC Cloud*). Sieťová verzia umožňuje ostatným študentom pripojiť sa a sledovať výkon svojich spolužiakov. Rovnako sa môžu pripojiť aj učitelia a viesť tak svojich študentov počas simulácie.

SimCEC sa skladá z niekoľkých modulov, z ktorých každý má na starosti rozdielne úlohy, a to konkrétne vykresľovanie, spracovanie vstupov a ovládanie avatara, komunikácia cez sieť, riadenie simulácie a hodnotenie používateľov.



Obr. 2.5: Rôzne typy úloh vykonávaných počas simulácie [7]

Používatelia v simulácii zastávajú jednu z rolí: chirurg, zdravotná sestra alebo anestéziológ. Kolaborácia v systéme pozostáva z tímovej spolupráce medzi sestrou a chirurgom. Úlohou sestry je výber vhodných nástrojov počas simulácie v

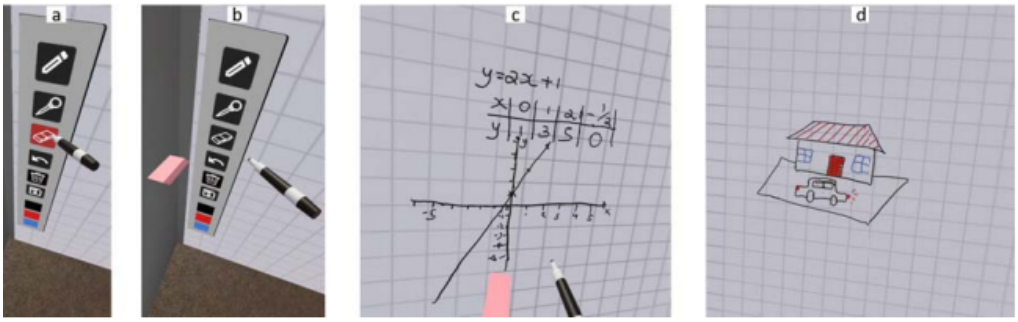
závislosti od vykonávaného úkonu. Úlohou chirurga je vykonať tieto úkony čo najpresnejšie a najsprávnejšie.

2.2.2 Virtual Family Room

Toto virtuálne prostredie vzniklo za účelom spojenia členov rodiny, ktorých oddeľuje veľká vzdialenosť [8]. Obsahuje niekoľko rôznych zábavných hier, ktoré sú určené nie len pre deti, ale aj ich rodičov. Prostredie ponúka aj nástroje pre spoločné riešenie domácich úloh. Vstavaný návod oboznámi používateľov so systémom a s jeho ovládaním.

Systém bol vyvinutý v *Unity*, pričom pripojenie a synchronizácia používateľov je implementovaná pomocou rámca *Photon Unity Network*. Každá rodina dostane unikátne číslo, ktoré im umožňuje pripojiť sa do rovnakej miestnosti naraz a zároveň to zabraňuje iným používateľom pripojiť sa do cudzej miestnosti. Stav jednotlivých miestností sa ukladá na server a počas pripojenia sa načíta posledný uložený stav.

Hlavnú časť systému tvorí rodinná miestnosť, kde sa odohrávajú všetky aktivity. Používateľ si môže prispôsobiť svojho avatara, konkrétne nastaviť meno a farbu, z ktorých je na výber osem. V scéne sa nachádza 3D zrkadlo, aby používateľ videl svoje zmeny. Pohybuje sa buď pohybom v reálnom svete, alebo využitím presúvania sa (teleportácie). Zaujímavosťou je používateľské rozhranie, ku ktorému pristupuje pomocou virtuálnych hodiniek, ktoré nosí jeho avatar.



Obr. 2.6: Virtuálna tabuľa umožňujúca používateľom kresliť vo virtuálnom priestore [8]

V systéme sa nachádza niekoľko aktivít zahŕňajúcich športové aktivity (minigolf, ihrisko a basketbalový kôš), strategické hry rozvíjajúce myslenie (stavanie lego kociek, 3D šachy a piškvorky), alebo aj interaktívna tabuľu, pomocou ktorej je možné kolektívne riešiť domáce úlohy. Taktiež je v scéne aj virtuálna video obrazovka, na ktorej môžu sledovať rovnaké video viacerí používatelia. Na dorozumievanie sa využíva priestorový zvuk. Ak niekto rozpráva, zobrazuje sa malá

ikonka reproduktora nad ich avatarom.

Obr. 2.7: Rôzne aktivity implementované v systéme [8]

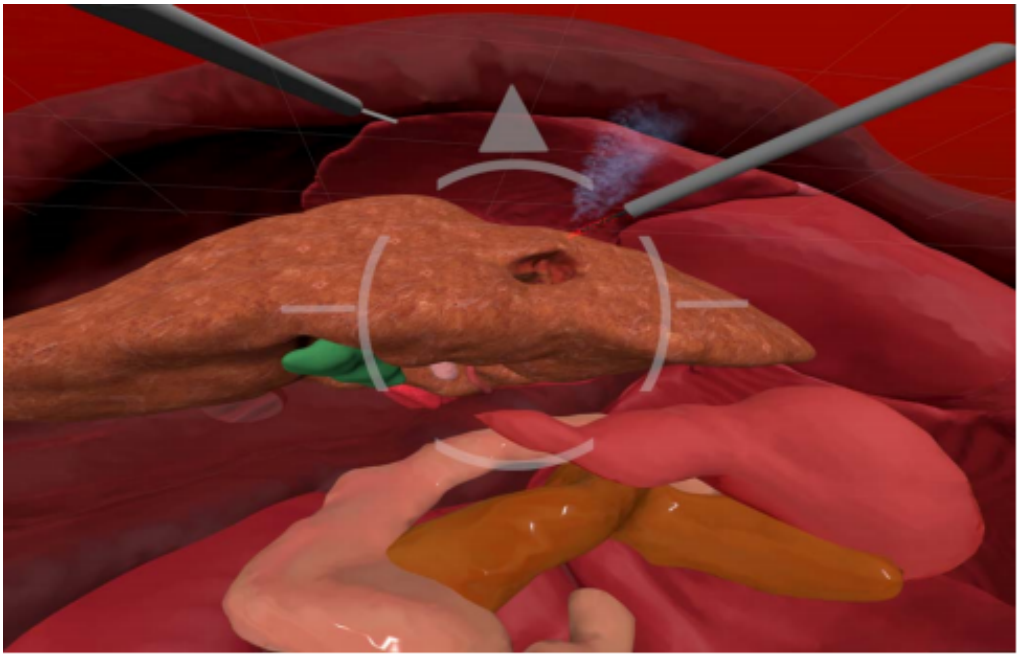
2.2.3 CollaVRLap

Príkladom ďalšieho kolaboratívneho VR systému je *CollaVRLap* [9]. Slúži na trénovanie chirurgov pri laparoskopickej operácii pečene. Podnetom na vznik bol nedostatok takýchto simulačných a výcvikových nástrojov, ale aj ich nerealistická simulácia.

Dôvodom použitia *Unity* v tomto systéme je natívna podpora VR prilby *HTC Vive* a taktiež niektorých nástrojov, najmä *Virtual Reality Toolkit* (VRTK) pre implementáciu základných interakcií vo virtuálnej realite. Kolaboráciu zabezpečuje rámec *Unity networking* (*Unet*).

Veľkou výhodou je, že používatelia pracujú s modelmi orgánov rekonštruovaných z reálnych dát skutočných pacientov. To umožňuje napríklad naplánovanie operácie ešte pred jej realizáciou, vo virtuálnom priestore. Všetko je možné vďaka tomu, že systém verne simuluje rezanie, krvácanie, strihanie a deformáciu tkaniva orgánov a tela.

Vo virtuálnom priestore je jedným z používateľov chirurg, zatiaľ čo druhý sa stará o ovládanie laparoskopickej kamery. Systém obsahuje dva módy, z ktorých jeden slúži na prezeranie si 3D modelu orgánu a v druhom sa realizuje simulácia operácie.



Obr. 2.8: Simulácia rezania tkaniva orgánu v systéme CollaVRLap [9]

2.2.4 Viac-používateľské univerzitné prostredie

Ďalší systém sa zameriava na trénovanie a simuláciu evakuácie univerzity a realizáciu bezpečnostných kurzov v jej priestoroch [10]. Okrem používateľov sa v prostredí nachádzajú aj počítačovo kontrolované entity, ktorých účelom je prekážať používateľom počas evakuácie.

Podobne, ako pri iných spomínaných riešeniach, aj tu je využívaná kombinácia rámcov *Unity* a *Unity Photon Network*. Používatelia sa pripájajú na cloud servery tak, že jeden z používateľov vytvorí miestnosť a prizve do nej ostatných. Grafické objekty prostredia, predovšetkým modely budov univerzity boli vytvorené v programe *3DS Max*.

Jedným z hlavných cieľov systému je simulácia správania sa ľudí počas evakuácie, konkrétne podliehanie panike, stresu, hnevu a blúdenie po oblasti. Takáto simulácia potom poskytuje dáta o časoch potrebných pre evakuáciu celej univerzity, alebo času potrebného pre záchranné zložky, aby dorazili na určité miesto.

Obr. 2.9: Simulovaná evakuácia vo virtuálnom prostredí univerzity [10]

2.2.5 Megacity

Narozdiel od predošlého systému sa tento zameriava na simuláciu evakuácie vo veľkomeste [11]. Cieľom je zníženie počtu ranených a mŕtvych počas kritických životných situácií, akými sú napríklad požiar a zrútenie budovy, teroristický útok a podobne.

Aj tento systém využíva rovnaké rámce a technológie, ako systémy *Virtual Family Room* a *VR univerzitné prostredie*, pričom sa používatelia pripájajú rovnakým spôsobom, ako v predošlom systéme. Narozdiel od nich *Megacity* podporuje ako prilby na VR (podporovaný je *Oculus Rift*), tak aj stolné počítače. Modely boli vytvorené pomocou 3D modelovacích softvérov *3DS Max* a *Google sketchUp*.

Obr. 2.10: Simulácia požiaru budovy v systéme Megacity [11]

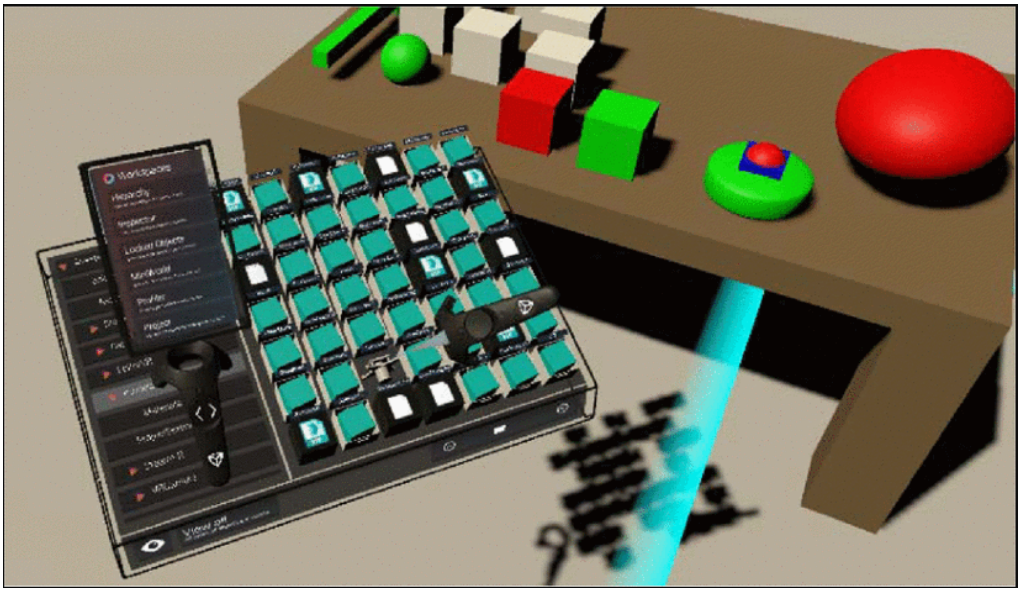
Systém simuluje správanie sa ľudí podobne, ako vo *VR univerzitnom prostredí*

a teda pomocou počítačových entít, ktoré reagujú na svoje okolie a prejavujú známky paniky, stresu, hnevu a dezorientácie. Rozdiel je len v rôznorodosti simulovaných entít.

2.2.6 Kolaboratívne VR prostredie pre návrh produktov

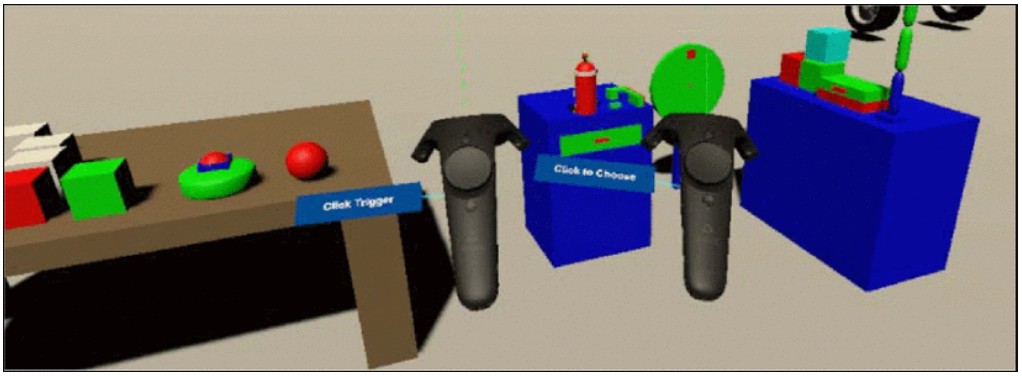
Posledný spomínaný systém umožňuje dizajnérom a inžinierom pracovať s prototypmi produktov a zostavovať ich v zdieľanom virtuálnom prostredí [12]. Návrh a tvorba prototypov takýmto spôsobom ušetrí firmám veľa času a financií. Ďalšou výhodou je možnosť spolupracovať bez ohľadu na geografickú vzdialenosť. Systém však nie je použiteľný pre všetky typy produktov, obzvlášť tých veľmi zložitých.

Tento systém využíva, podobne ako *CollaVRLap*, *Unity* s rozšírením *Virtual Reality Toolkit* a je vytvorený pre rovnakú VR prilbu - *HTC Vive*. Navyše podporuje aj platformu *SteamVR*. Interakcia s objektmi je zabezpečená ovládačmi, pričom ľavý ovládač slúži na ovládanie menu a pravý na interakciu s objektmi.



Obr. 2.11: Menu zobrazené ľavým ovládačom [12]

Tie môže používateľ uchopiť a manipulovať s nimi, meniť ich rozmery, dokonca si ich aj prehadzovať z jednej ruky do druhej. Systém je rozdelený do troch častí. Prvá je určená pre návrh a tvorbu 3D objektov v prostredí, zatiaľ čo zvyšné dve slúžia na prezentáciu modelu, pričom jedna je určená výhradne pre prezentéra a druhá pre zúčastnených používateľov, ktorým je model prezentovaný.



Obr. 2.12: Práca s objektmi vo virtuálnom prostredí [12]

2.3 Záver analýzy

Z analýzy vyplýva, že veľká časť spomínaných systémov je vytvorená pomocou herného rámca *Unity* a podporných knižníc a doplnkov. Dôvodmi sú napríklad široká podpora rôznych zariadení, virtuálnych headsetov a platforiem. Taktiež je možné v *Unity* vytvárať virtuálne svety pomocou virtuálnej reality, ako už bolo spomenuté, čo zjednodušuje a urýchľuje tvorbu virtuálnych prostredí, keďže dizajnéri majú možnosť vidieť svoje zmeny priamo v reálnom čase očami používateľa. Ďalšími výhodami sú rozsiahla komunita používateľov a množstvo kvalitných návodov pre prácu s týmto rámcom. V porovnaní s *A-Frame* beží systém vytvorený v *Unity* plynulejšie, čo je veľmi dôležité pre virtuálnu realitu, keďže pri nekonzistencii vykresľovania snímkov za sekundu a trhaní dochádza k nevoľnosti, závratom a iným zdravotným rizikám. Avšak práca s *Unity* je zložitejšia, keďže herný rámec obsahuje množstvo funkcií, ktoré je potrebné sa naučiť a osvojiť. Vývoj v takomto prostredí je teda časovo náročnejší pre začiatočníkov, v porovnaní s *A-Frame*.

Za najväčšiu výhodu *A-Frame* považujeme jednoduchosť práce s rámcom a rýchlosť, akou je možné vytvoriť prostredie pre virtuálnu realitu. Na začiatok stačí ovládať HTML jazyk a nie je nutné nič inštalovať. Tvorcovia pripravili štartovací projekt, ktorý tvorí akúsi šablónu pre tvorbu virtuálneho prostredia a obsahuje rôzne návody, ktoré ukážu vývojárom, ako pracovať s rámcom. Všetko je možné tvoriť priamo v prehliadači s využitím platformy *Glitch*, ktorá poskytuje hosting pre vlastné projekty zadarmo. Avšak v porovnaní s *Unity* je týchto návodov oveľa menej a vzhľadom na relatívne nový rámec je aj komunita o poznanie menšia, než v prípade *Unity*. Rovnako, ako aj *Unity*, *A-Frame* podporuje mnoho rôznych hardvérových riešení od mobilných zariadení, cez stolové počítače, až po virtuálne prilby. Keďže funguje na báze technológie *WebVR*/*WebXR*, je možné projekt

spustiť takmer vo všetkom, čo má kompatibilný prehliadač.

Ďalšia vec, čo nás zaujíma, je využitie takýchto systémov v praxi. Väčšina z nich našla využitie v oblasti vzdelávania, trénovania lekárov, doktorov a simulácie chirurgických zákrokov, simulácie a trénovania používateľov v rizikových situáciách, napríklad počas prírodnej katastrofy, teroristického útoku a pod.. Počas analýzy sme našli aj systémy, ktoré sa zameriavajú na tvorbu virtuálnych prostredí a prototypov. Pre naše použitie sú najzaujímavejšie práve systémy zaoberajúce sa tvorbou prototypov a ich testovaním vo virtuálnom prostredí. Avšak systémov s týmto zameraním je ešte veľmi málo, keďže virtuálna realita je pomerne mladá oblasť a ešte sa len hľadajú možnosti jej využitia. Potenciál tejto oblasti sa ešte len začína objavovať. Vzhľadom na to by mala táto práca okrem iného aj ukázať smer, akým sa môžu takéto systémy uberať a ich potenciálne využitie v oblasti prototypovania, napríklad zameranie na evaluáciu riadenia dronov.

3 Analýza súčasných prístupov a dostupných riešení pre realizáciu simulácie dronov

Popularita dronov a podobných lietajúcich strojov v posledných rokoch rapídne stúpa. Podľa posledného prieskumu trhu, opísaného v príspevku [13], je len v Európe registrovaných vyše 10 000 komerčných jednotiek. Autori toho istého prieskumu predvídajú zvýšenie tohto čísla v priebehu najbližších piatich rokov na hodnotu 200 000 dronov a do roku 2035 by ich dokonca malo byť 395 000. Až 40% z nich by malo nájsť využitie v poľnohospodárstve. Ostatné využitia vyplývajúce z prieskumu zahŕňajú energetický sektor, verejnú bezpečnosť a taktiež aj online nakupovanie a donáškové služby. Okrem toho sa bude v budúcnosti veľká časť dronov využívať pri ťažbe, staviteľstve a architektúre, telekomunikáciách a výskumnej činnosti.

V článku [14] sa spomína prieskum vykonaný v Spojených štátoch amerických, podľa ktorého tam bolo v roku 2018 zaregistrovaných cez 110 000 dronov, čo je dva a pol násobne viac, ako to bolo v roku 2016 a predpokladá sa, že toto číslo presiahne hranicu 600 000 jednotiek v roku 2022.

Existujú dva spôsoby, akými sú v súčastnosti ovládané tieto zariadenia, a to buď pilotované človekom na diaľku alebo autonómne pomocou riadiaceho softvéru.

3.1 Manuálne ovládanie dronov a simulátory

S rastúcim počtom dronov je dôležité, aby sa predišlo škodám a haváriám kvôli nedostatočným skúsenostiam pilota. V dôsledku toho vznikajú rôzne simulátory, kde je možné trénovať ovládanie virtuálneho lietajúceho drona. Takýto prístup umožňuje budúcim používateľom získať skúsenosti s pilotovaním bez nutnosti si zariadenie zakúpiť a riskovať jeho poškodenie alebo zničenie. V dnešnej dobe

existuje viacero realistických simulátorov, avšak drvivá väčšina z nich je spoplatnená. Z tohto dôvodu sme sa nimi nezaoberali. Je ale nutné poznamenať, že veľká časť z nich je na profesionálnej úrovni a predstavuje najvernejšiu simuláciu na trhu (príkladmi sú simulátory *DJI Flight Simulator*, *Liftoff: FPV Drone Racing* alebo *Phoenix R/C Pro Flight Simulator*).

Nás zaujal simulátor *DroneVR* opísaný v článku [14]. Jeho cieľom je minimalizovanie rizika nárazu drona do budovy alebo iného objektu. Princípom je modelovanie reálneho sveta a budov vo virtuálnom prostredí a následného pilotovania virtuálneho drona človekom, ktorý sa takto naučí ovládať zariadenie v konkrétnom simulovanom prostredí, ktoré zodpovedá reálnemu prostrediu. Dáta o budovách a ich parametroch systém čerpá zo systému *OpenStreetMap*, podľa ktorého vytvorí virtuálne prostredie. Je teda možné simulovať akékoľvek miesto, ktoré sa nachádza v mapách.

Drona je možné v simulátore ovládať buď manuálne alebo jeho ovládanie prenechať na autopilotovi. Ručné ovládanie je realizované pomocou dvoch virtuálnych ovládačov. V prípade autopilota je dron riadený autonómne tak, aby prešiel všetkými bodmi záujmu a zároveň nenarazil do žiadnej z virtuálnych budov. Ak používateľ chce, môže kedykoľvek počas letu prikázať dronu, aby sa sám vrátil späť. Táto funkcionalita sa už dnes nachádza aj v niektorých reálnych zariadeniach, avšak narozdiel od tohto simulátora väčšinou nezahŕňa kontrolu a vyhýbanie sa kolíziám s objektmi.

Systém *DroneVR* je vytvorený v jazyku *Javascript* s využitím vizualizačného rámca *ThreeJS*. To predstavuje veľkú výhodu, pretože je dostupný pre širší okruh záujemcov, keďže je možné ho spustiť vo webovom prehliadači bez nutnosti akejkoľvek dodatočnej inštalácie.

Príkladom ďalšieho simulátora určeného na trénovanie je systém opísaný v článku [15]. Tento simulátor poskytuje odlišný prístup k interakcii pilota s virtuálnym prostredím. Umožňuje totiž trénovať používateľov s využitím technológií virtuálnej reality, dôsledkom čoho je viac pohlcujúci zážitok v porovnaní s tradičnými simulátormi, ktoré spoliehajú na 2D zobrazovacie technológie.

3.2 Autonómne riadené drony a softvér autopilota

S rastúcim trhom rastie aj požiadavka na autonómne riadené drony. Výskumom v tejto oblasti je venované veľké úsilie. Je náročné vytvoriť autopilota, ktorý by bol schopný ovládať drona tak, aby nedošlo ku kolízii s ostatnými dronmi a objektmi v reálnom svete a zároveň bol schopný splniť svoju úlohu správne a efektívne.

Skúmajú sa možnosti sledovania okolia drona pomocou rôznych senzorov a kamier, aby mal autopilot spoľahlivé údaje potrebné na vykonanie úhybných manévrov. V článku [16] sú opísané rôzne spôsoby snímania okolia. Prvou kategóriou je využívanie špeciálnych senzorov, ako napríklad LiDAR senzory, sonar, infračervené lúče, alebo zložitejším spôsobom je možné využiť aj stereoskopické kamery na snímanie vzdialenosti od objektov. Druhou kategóriou je detekcia vzdialenosti od prekážok za pomoci algoritmov a umelej inteligencie v prípade, ak je dron, či už z finančných, hmotnostných alebo iných dôvodov, vybavený iba jedinou kamerou bez dodatočných senzorov. Výhodou tohto prístupu je, že je možné úspešne a s dostatočnou mierou presnosti snímať okolie drona v reálnom čase, iba s využitím jedinej kamery, čo prináša možnosti autonómneho riadenia aj do sektoru malých a lacných zariadení.

Ďalšou oblasťou záujmu je testovanie a overovanie funkčnosti autopilotov vo virtuálnom prostredí. V článku [17] je opísaný nástroj umožňujúci testovať funkcionalitu autonómneho ovládania bez nutnosti zásahu do ich softvéru. Existujú aj iné systémy, v ktorých je možné modelovať a simulovať drony a iné stroje vo virtuálnom prostredí, napríklad *Gazebo* a *Weebots*. Využíva sa aj *MATLAB* a jeho súčasť - *Simulink*, ktoré ale priamo nepodporujú virtuálnu realitu. Žiaden systém však nie je schopný modelovať všetky existujúce fyzické parametre reálneho sveta. Z tohto dôvodu je nutné otestovať autopilota aj v reálnych podmienkach, pod dohľadom experta. Autori využívajú platformu *Gazebo* [18] na zber dát z virtuálneho prostredia a systém *VICON* pre získavanie dát z reálneho sveta snímaním pohybu lietajúceho drona. Tieto dáta následne použili pre natrénovanie algoritmu typu rozhodovacieho stromu. Tento algoritmus potom využili na detekovanie jedného z piatich stavov dronu (pohotovostný stav, štart, vznášanie, vyhľadávanie a pristátie) podľa aktuálnych údajov o výške a rýchlosti v troch osiach. Porovnávali predikovaný stav s reálnym stavom a zaznamenávali úspešnosť algoritmu. Výsledkom je nástroj, ktorý dokáže automaticky testovať správanie autopilotov bez akéhokoľvek zásahu do riadiaceho softvéru.

Iný pohľad na testovanie a overovanie riadiaceho softvéru je opísaný v článku [19]. Cieľom je overiť funkcionalitu autopilota v každej možnej situácii a sledovať jeho reakciu a čas potrebný na vykonanie adekvátnej akcie. Pre tento účel vytvorili model, ktorý použili na simulovanie všetkých stavov pre riadiaci program a zaznamenávali výstupné údaje. Rozdielom oproti vyššie spomínanému nástroju je, že tento nástroj je špecifický pre konkrétneho autopilota, keďže testuje funkcionalitu pre jeho vnútorné stavy. Dôsledkom je, že hoci je veľmi špecifický, dokáže veľmi podrobne otestovať funkčnosť a správanie konkrétneho autopilota vo väč-

šine situácií.

3.3 Analýza dostupných riešení pre realizovanie simulácie

V nasledujúcich častiach práce sa budeme zaoberať dronmi vybavenými štyrmi pohonnými jednotkami, nazývanými aj kvadrokoptéry. Dôvodom je ich popularita, jednoduchosť a rozšírenie v komerčnej oblasti. Z tohto dôvodu aj všetky nižšie spomínané simulátory implementujú model takýchto typov dronov. Prvým krokom pri tvorbe prostredia s realistickou simuláciou lietajúceho drona je implementácia vhodného fyzikálneho modelu, ktorý by definoval správanie sa kvadrokoptéry podľa známych fyzikálnych zákonov. Tieto fyzikálne zákony určujú správanie sa kvadrokoptéry a je možné ich popísať matematickými rovnicami, ktoré tvoria jej matematický model. Existuje mnoho prác zaoberajúcich sa matematickým modelom správania sa dronov, napríklad práce [20] a [21]. Pomocou matematických modelov opísaných v spomínaných prácach je možné následne implementovať fyzikálnu simuláciu v rôznych jazykoch a prostrediach. Nás zaujíma už konkrétna implementácia takéhoto modelu, preto sme sa rozhodli vykonať prieskum systémov, ktoré už takúto simuláciu realizujú. Pomocou získaných poznatkov tak získame predstavu o zložitosti a štruktúre takéhoto systému a budeme môcť vytvoriť vlastnú simuláciu v prostredí *LIRKIS G-CVE*.

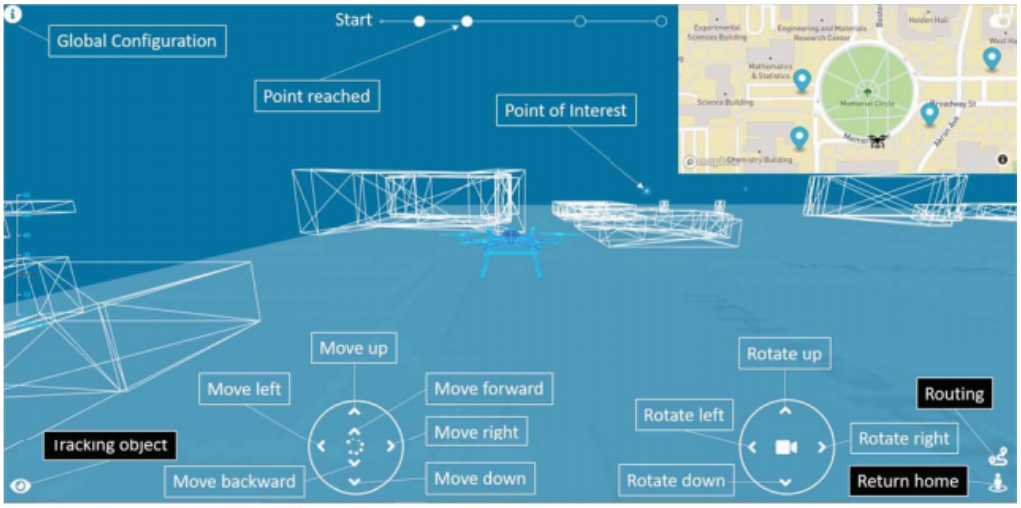
3.3.1 Simulátor DroneVR

V tejto časti sa pozrieme bližšie na už spomínaný simulátor *DroneVR* [14]. Naším cieľom je zistiť, akým spôsobom je v tomto simulátore implementovaný pohyb a ovládanie drona, najmä či jeho pohyb verne simuluje lietanie reálnych strojov v skutočnom svete. Ďalej nás zaujíma implementácia a využitie dát z *OpenStreetMap* na vytvorenie virtuálneho prostredia na základe údajov z reálneho sveta.

Simulácia pohybu drona

Po spustení simulátora sa dron nachádza už pripravený vo vzduchu v staticky nastavenej výške 7 metrov nad zemou. Simulátor teda nijakým spôsobom neponúka možnosť simulovať štart a ani pristátie. Ako už bolo spomínané v predchádzajúcej časti tejto kapitoly, používateľ má k dispozícii niekoľko ovládacích prvkov, ktorými je možné ovládať drona. Taktiež tento systém umožňuje ponechať ovládanie na autopilotovi za predpokladu, že sme v systéme nastavili body záujmu,

ktoré má dron dosiahnuť, alebo sme mu zadali príkaz návratu domov, keďže jeho štartovacia pozícia je v systéme predvolene nastavená.



Obr. 3.1: Prostredie simulátora *DroneVR* [14]

Ovládanie drona pomocou virtuálnych ovládačov

Ručné ovládanie je realizované prostredníctvom dvoch virtuálnych ovládačov. Na ovládači vľavo sa nachádza ovládanie pohybu. Stlačením a držaním daného tlačidla dron vykonáva žiadaný pohyb. Na ovládači vpravo sa nachádza ovládanie rotácie. Konkrétne sa jedná o rotáciu vo vertikálnej osi smerom vľavo a vpravo. Simulátor obsahuje aj ovládače pre ovládanie náklonu smerom dopredu a dozadu, avšak počas testovania sme zistili, že táto funkcionalita nebola implementovaná. Okrem toho v systéme úplne chýba možnosť ovládať náklon doľava a doprava.

Našim hlavným cieľom však bolo zistiť, či sa na simuláciu pohybu využíva nejaký fyzikálny model, ktorý by verne simuloval lietanie takéhoto zariadenia vo vzduchu. Avšak počas analýzy sme zistili, že pohyb drona je vyriešený veľmi jednoducho. Pohyb vo všetkých osiach je okamžitý a perfektný, bez akejkoľvek simulácie postupného zrýchľovania alebo spomaľovania. Taktiež nedochádza pri pohybe k náklonu lietajúceho stroja. Pohyb do všetkých smerov je obmedzený iba maximálnou rýchlosťou, ktorá je pre stúpanie, klesanie, ale aj pre zvyšné smery rovnaká. Nedostatkom je tiež absencia kontroly kolízií medzi dronom a virtuálnym svetom. Dôsledkom je, že dron nijako neinteraguje s okolitým prostredím a teda jeho pohyb nie je nijako obmedzený. Je možné s dronom prejsť cez podlahu alebo vojsť do budovy a navyše na to systém používateľa nijako neupozorní.

V prípade otáčania môže používateľ drona otáčať smerom doľava alebo doprava. Otáčanie taktiež prebieha jednoducho a okamžite, bez simulácie fyzikál-

nym modelom a tým pádom bez zotrvačnosti. Rýchlosť otáčania je obmedzená interne, v programe.

Ovládanie drona autopilotom

V prípade ovládania autopilotom je lietanie riadené automaticky tak, aby nedošlo ku kolízii s objektmi vo virtuálnom svete. Na navigáciu drona v priestore používa autopilot body záujmu, medzi ktorými sa pomocou algoritmu pre problém obchodného cestujúceho vygeneruje optimálna trasa medzi stanovenými bodmi, ktorú potom autopilot nasleduje. Trasa je vytvorená tak, aby neprechádzala cez budovy, bola čo najkratšia a aby bol v nej obsiahnutý každý jeden bod záujmu práve jeden krát. Následne autopilot pohybuje dronom, čím sleduje jednotlivé body trasy až do cieľa.

Počas testovania sme si všimli dve obmedzenia. Prvým z nich je, že po vytvorení bodov záujmu sa všekty nachádzajú v rovnakej výške 7 metrov nad zemou. To výrazne zjednodušuje prácu autopilota, keďže nemusí počas letu dochádzať k zmene výškovej hladiny stroja. Druhým obmedzením samotného autopilota je, že dron sa automaticky otáča iba po násobkoch $45°$. Presun k danému bodu potom pripomína posun figúrky kráľovnej na šachovnici. Dôsledkom takéhoto ovládania je nie celkom ideálna trasa k cieľu a taktiež dlhší čas potrebný na dosiahnutie bodu záujmu.

Generovanie sveta a dáta z OpenStreetMap

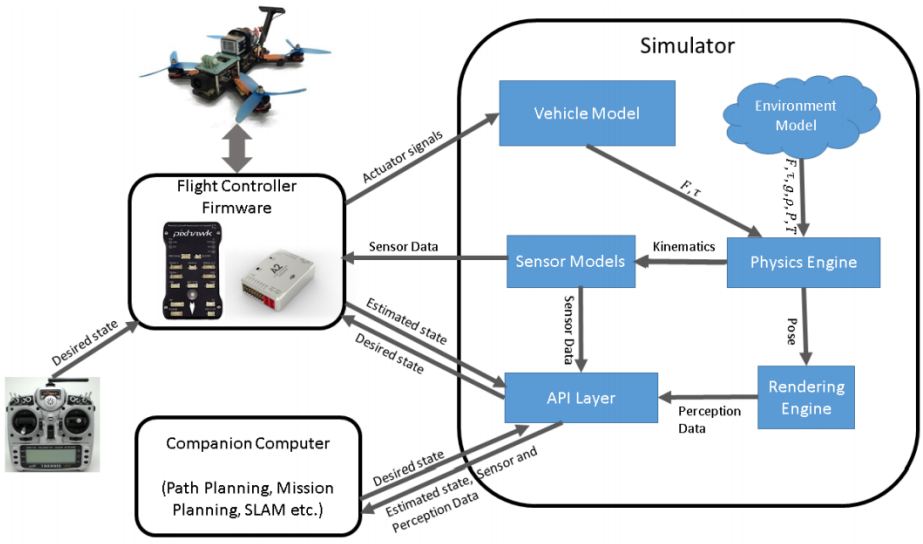
Simulátor využíva dáta o objektoch z *OpenStreetMap* na tvorbu budov a objektov vo virtuálnom prostredí, ktoré zodpovedajú reálnym v danej oblasti. Konkrétne pracuje s dátami vygenerovanými prostredníctvom portálu `openstreetmap.org`. Dáta sú vo formáte XML a obsahujú zoznam bodov a ich pozície v 3D priestore. Tieto body reprezentujú budovy, na základe ktorých potom systém vytvorí 3D objekt vo virtuálnom prostredí. Je dôležité poznamenať, že tieto dáta sú staticky importované do systému. Ak chceme zmeniť oblasť, v ktorej sa nachádzame, je nutné vygenerovať nové dáta na portáli a importovať ich do systému. Z tohto dôvodu nie je možné v systéme lietať mimo vygenerovaný rozsah a očakávať, že sa nové dáta automaticky stiahnu a systém vytvorí nové budovy.

Systém taktiež obsahuje mapu v pravom hornom rohu vycentrovanú na aktuálnu pozíciu drona, ktorá okrem iného slúži aj na nastavovanie bodov záujmu. Pozícia stredu mapy je staticky nastavená a musí zodpovedať oblasti, ktorá bola získaná z portálu `openstreetmap.org`. V opačnom prípade nie je možné použiť autopilota na navigáciu v priestore.

Napriek nedostatkom a absencii fyzikálneho modelu tento simulátor ukazuje, že je možné využiť dáta z otvorených zdrojov, akými sú napríklad služby *OpenStreetMap* na vytvorenie konkrétneho virtuálneho prostredia, kde by bolo možné otestovať drona a jeho autopilota na reálnych údajoch.

3.3.2 AirSim

Tento simulátor dokáže simulovať drony, vozidlá a iné pohyblivé stroje. Je vytvorený v hernom rámci *Unreal Engine* s druhou, zatiaľ experimentálnou verziou, vytvorenou pomocou rámca *Unity*. Simulátor je open-source a vďaka implementácii v týchto rámcoch podporuje viacero platforiem. Použitím dvoch moderných herných rámcov je docielená aj vysoká vizuálna kvalita virtuálneho prostredia, čo je jednou z hlavných výhod tohto systému. Ďalšou veľkou výhodou je integrácia s existujúcim ovládacím firmvérom pre drony. Medzi ne patria známe ovládače *PX4*, *ROSFlight* alebo *Hackflight*. Počas simulácie sú ovládaču dodávané letové údaje zo simulovaných senzorov (akcelerometer, gyroskop, barometer, magnetometer a GPS), na základe ktorých ovládač vypočíta hodnoty pre ovládanie motorov kvadrokoptéry. Následne interný fyzikálny systém na základe síl produkovaných motormi prepočíta zmeny v pohybe. Tento pohyb ovplyvňuje aj prostredie, v ktorom je možné simulovať viacero faktorov, napríklad dážď, sneženie, prach alebo hmlu. Okrem toho je možné nastaviť aj rýchlosť a smer vetra. [22]



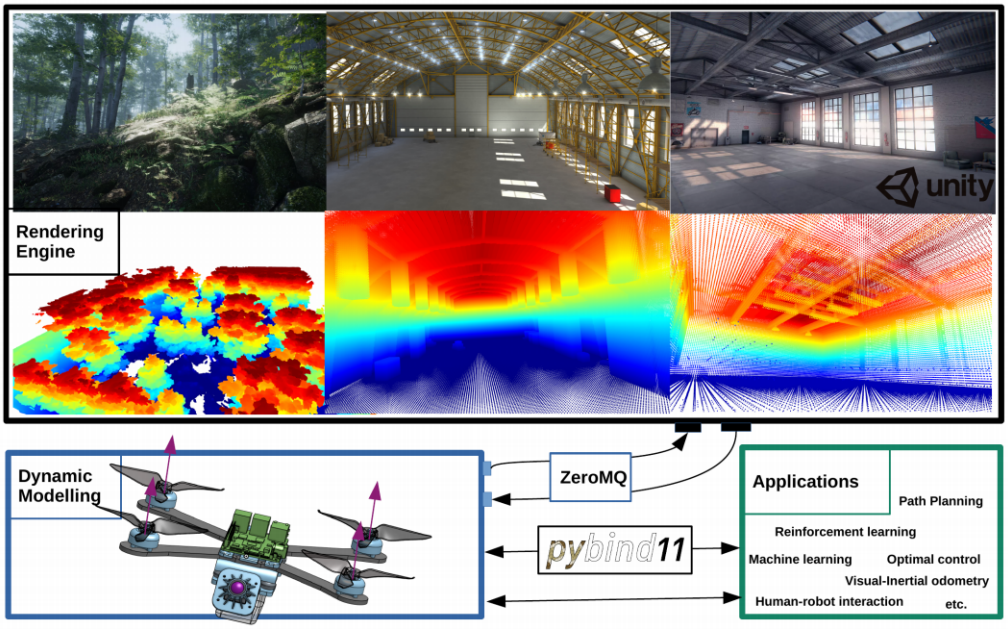
Obr. 3.2: Architektúra systému *AirSim* [22]

Príkazy pre kvadrokoptéru sú dodávané pomocou ovládača pre reálne zariadenia alebo využitím druhého počítača pre generovanie príkazov, napríklad v prípade sledovania konkrétnej trasy a bodov záujmu. Na tento účel poskytuje

AirSim aplikačné rozhranie. Hlavným cieľom tohto simulátora je poskytnúť platformu pre výskum v oblasti umelej inteligencie, konkrétne pre potreby trénovania autonómnych algoritmov.

3.3.3 Flightmare

Druhý simulátor je modulárny a flexibilný so zameraním špeciálne pre drony. Skladá sa hlavne z dvoch nezávislých komponentov - vizualizačného rámca *Unity* a fyzikálneho rámca pre simulovanie pohybu drona. Výhodou oddelenia fyzikálnej simulácie od vizualizácie, je zvýšenie výkonnosti a presnosti simulovanej dynamiky pohybu využitím paralelného programovania. Rozhranie medzi vizualizačným rámcom a fyzikálnou simuláciou je implementované pomocou knižnice pre vysoko-výkonné asynchrónne posielanie správ *ZeroMQ*. Okrem toho autori implementovali rozhranie *OpenAI-Gym* pre nástroje strojového učenia v simulátore. [23]



Obr. 3.3: Architektúra systému Flightmare [23]

Simulátor taktiež obsahuje rozhranie pre prístup k údajom zo senzorov rovnakých, ako v simulátore *AirSim*. Navyše umožňuje extrahovať mračno 3D bodov aktuálnej scény. Tiež obsahuje rozhranie pre simuláciu a trénovanie umelej inteligencie. Jeho efektívna implementácia im umožňuje simulovať naraz stovky dronov. V poslednom rade podporuje zariadenia virtuálnej reality, takže je možné sa do prostredia imersívne zapojiť a interagovať s ním.

3.3.4 Ďalšie menšie open-source simulátory

Ďalej sa pozrieme na menšie projekty, ktorých zdrojové kódy sme našli na verejne dostupných repozitároch na Githube. Uvedené sú v prehľadnej tabuľke 3.1 s popisom jednotlivých systémov. Každý z nich implementuje matematický model kvadrokoptéry v určitej forme. Príklady prác zaoberajúcimi sa matematickým vyjadrením fyzikálneho správania sa dronov sú spomínané v úvode tejto kapitoly. Okrem toho všetky riešenia používali na stabilizáciu kvadrokoptéry regulátory typu PID. Jedná sa o proporcionálny, integračný a derivačný regulátor [24]. Jeho úlohou je priblížiť sa čo najbližšie požadovanej hodnote. Rýchlosť a tvar približovania určujú jeho parametre, ktoré je potrebné nastaviť podľa potreby.

3.4 Záver analýzy

Z prieskumu vyplýva, že v súčasnosti už existuje niekoľko systémov v oblasti simulácie kvadrokoptér. Niektoré sa snažia zaujať fotorealistickým grafickým prostredím, iné poskytovanými funkciami a možnosťami rozšírenia. Okrem analyzovaných systémov existujú aj simulátory poskytované priamo výrobcami komerčných dronov, napríklad *DJI*. Ich simulátory predstavujú špičku v tejto oblasti, avšak v drvivej väčšine prípadov sa jedná o platené riešenia, ktoré neponúkajú prístup k zdrojovému kódu.

Z hľadiska použitej platformy všetky simulátory zahrnuté v tomto prieskume tvoria samostatnú aplikáciu, ktorú je nutné stiahnuť a nainštalovať. Výhodou takéhoto prístupu je vyššia celková výkonnosť simulátora a možnosť použiť presnejšie výpočty na simuláciu fyzikálneho systému. Nevýhodami sú nutnosť aplikáciu inštalovať pred použitím a s tým súvisiace nároky na úložný priestor na zariadení. Ďalšou nevýhodou je nutnosť aplikáciu exportovať pre každú platformu zvlášť. Navyše takýto export môže vyžadovať určité úpravy systému.

Simulácia fyziky je vo väčšine prípadoch docielená implementáciou matematických modelov, opísaných v spomínaných prácach. Na tieto účely systémy využívajú matematické knižnice pre účely riešenia diferenciálnych rovníc pohybu v priestore. Dôsledkom tohto prístupu je, že nevyužívajú všeobecné fyzikálne rámce. Výnimkou však je systém *AirSim*, ktorý matematické modely využíva len na výpočet produkovaného ťahu a krútiaceho momentu motorov a o samotnú simuláciu pohybu sa stará vstavaný fyzikálny rámec.

V oblasti riadenia a stabilizácie kvadrokoptéry v simulovanom prostredí využívajú všetky systémy PID regulátory. Dôvodom je, že ide o spôsob regulácie,

Názov projektu	**Autor**	**Použitý jazyk a knižnice**	**Stručný popis**
Quadcopter 3D Simulator [25]	Peter Huang	*Python*, knižnice {matplotlib} a {SciPy} pre vizualizáciu a fyzikálne výpočty	Projekt používa na ovládanie pohybu drona PD regulátor, ktorý na základe požadovanej a aktuálnej pozície vyhodnotí odchýlku a tá sa použije ako výstup pre ovládanie motorov. Systém vychádza z poznatkov získaných počas kurzu Robotika: Lietajúce roboty [26].
Quadcopter simulator [27]	Abhijit Majumdar	*Python*, knižnice *SciPy*, *Numpy* a *Math* na výpočet fyzikálneho modelu, *matplotlib* na vizualizáciu	Realizácia dynamiky a fyziky sa opiera o práce [20] a [21]. Taktiež využíva vzorec na výpočet statického ťahu generovaného motormi, kde záleží od parametrov lopatiek a ich uhlovej rýchlosti. Na ovládanie a stabilizáciu kvadrokoptéry sa používa PID regulátor.
Multicopter Sim [28]	Simon D. Levy	*C++*, herný rámec *Unreal Engine* a ovládací firmvér podľa výberu, odporúčaný je *Hackflight*, ktorý bol tiež vytvorený týmto autorom	Simulátor sa podobá na *AirSim* spomínaný vyššie, avšak je jednoduchší a zameraný výhradne na simuláciu kvadrokoptér a neobsahuje žiadnu funkcionalitu na prácu s AI. Implementácia dynamiky vychádza z práce [29] a teda simulátor nevyužíva vstavaný fyzikálny systém *Unreal Enginu*.
Quadcopter-Simulator [30]	Mohamed Khair Dimashky, Maher Al-Kassir, Majd Al-Ajlani	*C#*, rámec *XNA framework v4.0* a knižnica *Math.NET* na realizáciu výpočtov	Systém obsahuje simuláciu senzorov kvadrokoptéry (akcelerometer, gyroskop, magnetometer a barometer), ktorých údaje dodáva riadiacemu softvéru. Kaskádovo zapojené PID regulátory sú použité na stabilizáciu drona. Systém sa opiera o poznatky opísané v práci [31].

ktorý nemá zložitú štruktúru, je jednoduchý na pochopenie a funguje spoľahlivo. Mierne odlišný prístup mal len jeden zo systémov, simulátor *Quadcopter 3D Simulator* autora Petra Huanga, ktorý sa rozhodol použiť jednoduchší PD regulátor, ktorý neobsahuje integračnú zložku.

Tieto oblasti nás zaujímali najviac. Ostatné pridané funkcionality konkrétnych simulátorov, ako napríklad podpora rôznych ovládacích firmvérov pre drony, podpora simulácie a trénovania agentov umelej inteligencie, či export dát zo simulátora, sú príkladom ďalších oblastí, v ktorých možno prácu rozvíjať.

Pre implementáciu modelu v našom prostredí sme sa rozhodli použiť rovnaký prístup, aký využíva simulátor *AirSim*, teda kombináciu fyzikálneho rámca a vzťahov na výpočet síl motorov. Taktiež použijeme PID regulátor na stabilizáciu a riadenie kvadrokoptéry.

4 Integrácia riadiaceho softvéru so zdieľaným virtuálnym prostredím

Ešte pred samotným návrhom a implementáciou systému sme vykonali experiment, ktorého cieľom bolo overiť, či a akým spôsobom je možné integrovať prototyp riadiaceho softvéru do prostredia *LIRKIS G-CVE*. Počas integrácie do tohto prostredia sme využili komponent pre komunikáciu medzi entitami, knižnicu *Networked-Aframe* a prototyp riadiaceho softvéru pre robotický vysávač (cBot), ktorý je podrobne opísaný v článku [32]. V závere sme ešte zhrnuli naše postrehy a pripomienky k práci s rámcom *A-Frame* ako takým.

4.1 Komponent pre komunikáciu medzi entitami

Tento komponent vznikol ako výsledok diplomovej práce Michala Ivana [33]. Slúži na zdieľanie informácií o jednotlivých klientoch pripojených do virtuálnej scény. Taktiež umožňuje tieto entity vynútene odpojiť zo scény (prostredníctvom admina) a sledovať ich aktuálnu pozíciu v scéne. Práve sledovanie pozícií sme využili v implementácii vysávacieho robota, aby sme mu vedeli dať informácie o tom, kde a v akých vzdialenostiach od robota sa jednotliví používatelia nachádzajú. Riadiaci softvér tieto údaje potom používa na pohyb po scéne tak, aby robot nenarazil do žiadneho používateľa počas toho, ako sa presúva medzi pozíciami na čistenie a neohrozoval ich čistiacim procesom. Komponent využíva *Networked-Aframe* na zdieľanie týchto informácií cez sieť.

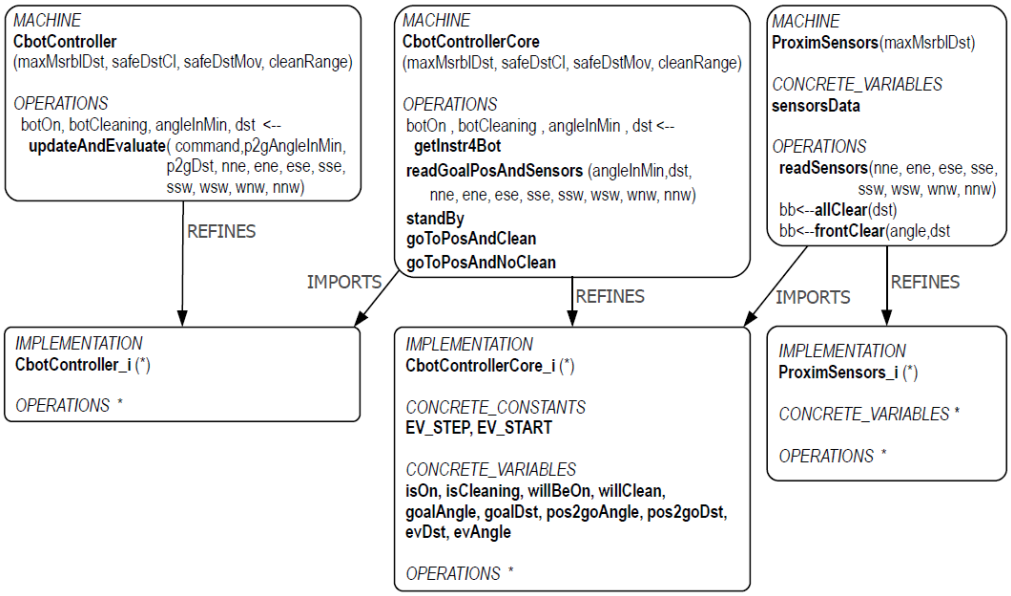
4.2 Networked-Aframe knižnica

Funkcionalitu tejto knižnice využíva aj vyššie spomínaný komponent, ale v čase, keď bol komponent vytváraný používala knižnica rozhranie *EasyRTC*, čo je vlastne nadstavba nad *WebRTC* protokolom, ktorý obstaráva prenos dát cez internet pomocou *UDP*. V najnovšej verzii (v čase písania je to verzia 0.7.1) je predvolene po-

užívaný protokol *WebSocket*, ktorý narozdiel od *WebRTC* využíva *TCP* spojenie. Preto bolo nutné zmeniť adaptér *Networked-Aframe* používaný na komunikáciu po sieti. Zmena je jednoduchá, stačí len stiahnuť implementáciu daného adaptéra a priložiť ho do HTML stránky rovnako, ako iné súbory s kódom v jazyku *Javascript*, avšak zdrojový kód samotnej knižnice sa musí vyskytovať pred týmto adaptérom, aby ju vedel adaptér rozpoznať. Následne je ešte potrebné v konfigurácii komponentu *Networked-Aframe* nastaviť požadovaný adaptér na `easyrtc`. Po vykonaní týchto úprav môžeme používať komponent pre komunikáciu medzi entitami bez nutnosti v ňom vykonávať úpravy, aj s najnovšou verziou knižnice.

4.3 Model ovládača pre cBot-a a generovanie kódu

Pre použitie prototypu ovládača v scéne je najprv nutné použiť vhodný prekladač na vygenerovanie spustiteľného kódu zodpovedajúceho formálnej špecifikácii. Model bol vytvorený a opísaný v článku [32] pomocou jazyka *B* a *B-Metódy*.



Obr. 4.1: Špecifikácia ovládača v jazyku B [32]

Z tejto špecifikácie sa následne vygeneroval kód v jazyku *Java*. Výsledkom prekladu boli štyri súbory, ktoré ako celok tvoria funkčný ovládač:

- `CBotControllerCore.java` - obsahuje jadro ovládača a všetku funkcionalitu pre rôzne stavy vrátane algoritmu vyhýbania sa objektom
- `CBotController.java` - predstavuje rozhranie pre ovládač a obsahuje metódu pre aktualizáciu a vyhodnotenie stavu robota

- `ProximSensors.java` - v ňom sa nachádza funkcionalita pre reprezentáciu senzorov robota a snímania okolia
- `ArrayUtils.java` - obsahuje metódu na inicializáciu poľa

Po vygenerovaní kódu z formálnej špecifikácie je nutné ho ešte preložiť do jazyka *Javascript*, keďže *A-Frame* je webový rámec vytvorený v tomto jazyku. Na preklad z Javy do *Javascriptu* sme použili nástroj *Jsweet*. Konkrétne, tento nástroj prekladá kód z jazyka Java do *Typescript*, ktorý následne pretransformuje do zvolenej verzie *Javascriptu*.

Nástroj sme spúšťali pomocou vývojového prostredia *IntelliJ IDEA*, v ktorom po konfigurácii parametrov sa pri preklade kódu ovládača vygeneroval aj kód v *Javascripte*. Prvotne nastal problém pri preklade súboru `ArrayUtils.java`, keďže nástroj nedokázal správne preložiť metódu používajúcu generické typy. Bolo nutné tento súbor vyradiť z prekladu a miesta, kde sa používal kód v ňom definovaný nahradiť. V tomto prípade sa jednalo iba o jedno volanie metódy v súbore `ProximSensors.java`, ktoré sa používalo na inicializáciu poľa senzorov. Po úprave kódu v *Jave* bol preklad úspešný. Pre istotu sme ešte overili sémantickú korektnosť preloženého kódu spúšťaním hlavnej metódy definovanej v súbore `CBotController.j` pre rôzne vstupné hodnoty v oboch jazykoch. Po overení správnosti nasledovala integrácia ovládača do virtuálnej scény.

4.4 Nasadenie ovládača do virtuálnej scény

Proces nasadenia ovládača do scény pozostával z vytvorenia virtuálnej entity, ktorá bude ovládaná a spracovania vstupov pre ovládač na základe údajov zo scény. Vytvorili sme teda entitu reprezentujúcu automatického robota. Rovnako, ako aj entity používateľov (ich avatar), aj poloha a rotácia tohto robota je pomocou *Networked-Aframe* odosielaná v reálnom čase pre všetkých, ktorí sú pripojení v rovnakej miestnosti.

Funkcionalita sa v *A-Frame* pridáva entitám pomocou komponentov, ktoré vieme dynamicky vyskladať podľa potreby. Na integráciu ovládača do scény sme vytvorili samostatný komponent `cbot-cleaning`, ktorý používa rozhranie vygenerovaného a preloženého ovládača pre robota a na základe jeho výstupných údajov potom upravuje atribúty entity (pozíciu a rotáciu). Tento komponent vieme pridať rôznym entitám a vytvoriť tak z nich robotický vysávač jednoducho a bez nutnosti niečo meniť.

V inicializačnej časti tohto komponentu sa vytvorí inštancia ovládača a inicia-

lizuje sa vstupnými hodnotami. Na tento účel sme použili `schema` komponentu. V nej sme zadefinovali názov, typ a predvolenú hodnotu pre atribúty potrebnými na konfiguráciu ovládača. To nám umožňuje ich ľubovoľne nastaviť iba za pomoci atribútov komponentu priamo v HTML entite, teda bez zásahov do Javascript kódu. Konkrétne sa jedná o hodnoty `maxMsrblDst` (dosah senzorov), `safeDstCl` (bezpečná vzdialenosť pre čistenie), `safeDstMov` (bezpečná vzdialenosť pre pohyb) a `cleanRange` (dosah čistenia). Základnou jednotkou vzdialenosti je v *A-Frame* meter, preto sú absolútne hodnoty relatívne malé (pohybujú sa v rozmedzí od 1 do 15). Predvolené hodnoty sme nastavili podľa testovania vo virtuálnej scéne na `maxMsrblDst` - 15, `safeDstCl` a `safeDstMov` - 3 a `cleanRange` - 0.1.

Úlohou tohto robota je prejsť všetkými pozíciami a vyčistiť ich. Preto sme potrebovali vytvoriť entitu pre takéto body. Na tento účel sme vytvorili ďalší komponent, `cbot-dirty-position`, ktorému vieme rovnako, ako aj v komponente pre robota nastaviť atribúty. Pre tento komponent nastavujeme iba jednu hodnotu a to úroveň špinavosti. Tá predstavuje hodnotu, ktorú robot po príchode na pozíciu znižuje o jedna každým volaním metódy `tick`. Vizuálne je táto entita reprezentovaná nízkym kvádrom oranžovej farby, ktorý automaticky mení svoju priehľadnosť na základe úrovne čistoty. To znamená, že čím je úroveň špinavosti bližšie k 0, tým je entita priehľadnejšia, až ju nakoniec vôbec nevidieť. Až klesne táto hodnota na nulu, entita je zo scény odstránená.

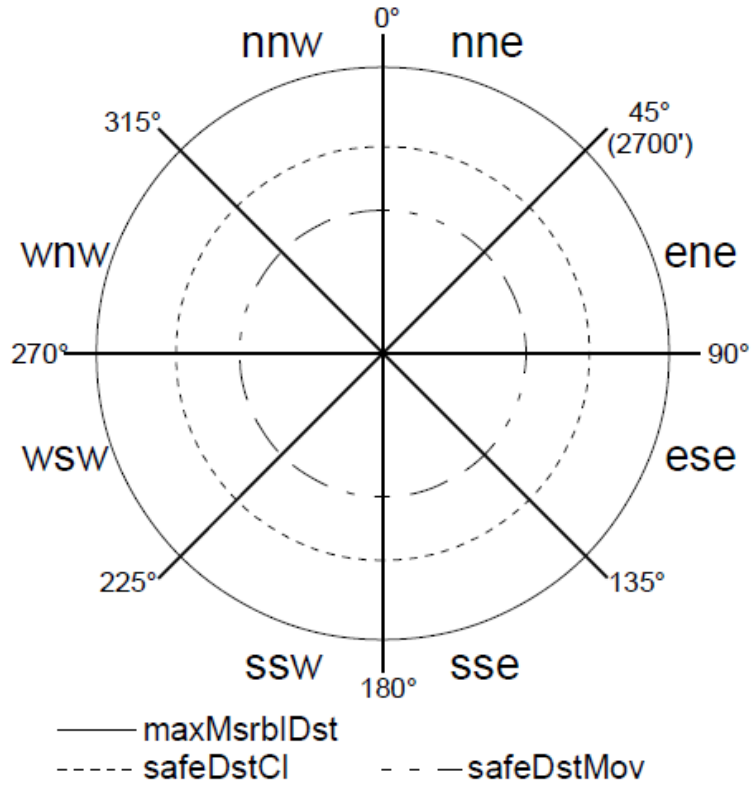
Robot si počas inicializačnej fázy automaticky vyhľadá všetky entity v scéne s týmto komponentom a pridá si ich do zoznamu. Ovládanie robota sme implementovali v metóde `tick` komponentu. Táto metóda je zavolaná vždy pri prekreslení snímky, teda v ideálnom prípade 60-krát za sekundu. V nej sme použili rozhranie ovládača pre získanie riadiacich údajov, ktoré sa potom použijú na zmenu atribútov entity. Postup operácií je nasledovný:

1. Vypočítanie vzdialenosti a uhlu natočenia robota k nasledujúcemu bodu záujmu.

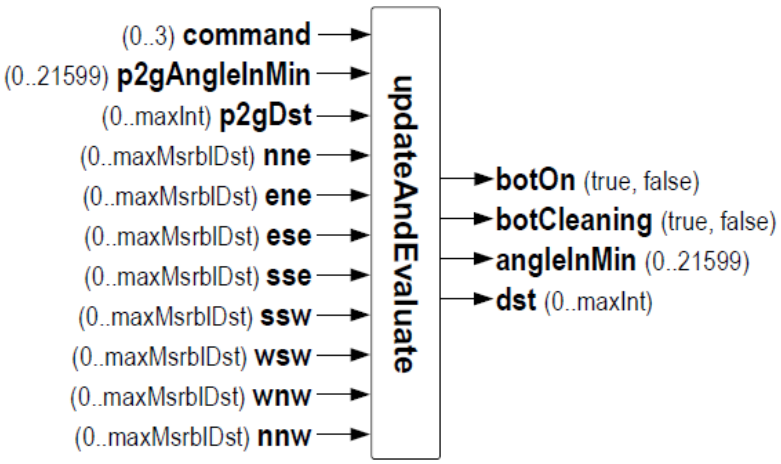
2. Výpočet vzdialeností všetkých používateľov od robota pre každý sektor definovaný na obrázku 4.2.

3. Zavolanie metódy ovládača použitím vypočítaných hodnôt a príkazu pre čistenie (rozhranie a zoznam hodnôt je na obrázku 4.3).

4. Aktualizácia atribútov entity, konkrétne pozície a rotácie, prípadne aktivovanie čistenia bodu, ak sa robot nachádza na správnej pozícii.



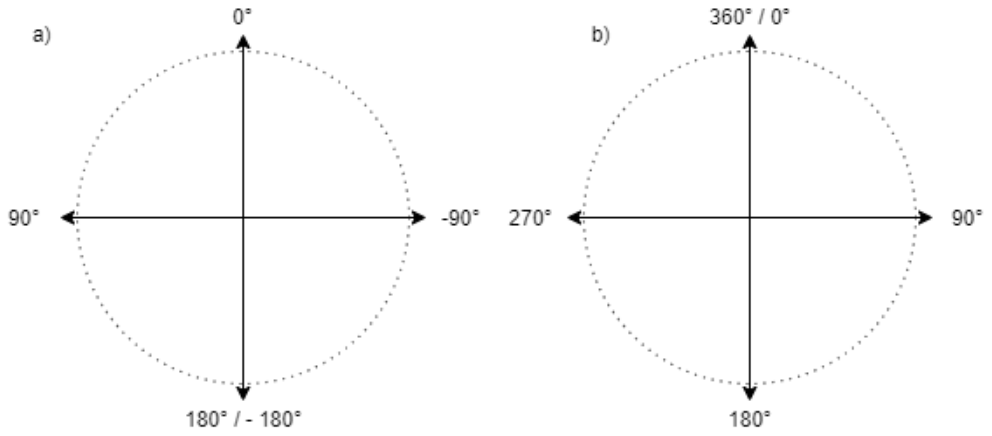
Obr. 4.2: Jednotlivé sektory senzorov pre snímanie okolia robota [32]



Obr. 4.3: Rozhranie ovládača a jeho vstupné a výstupné hodnoty [32]

Na získavanie pozícií všetkých používateľov používame komponent vytvorený v diplomovej práci [33]. Ten nám v reálnom čase ukladá a aktualizuje pozície všetkých používateľov, ktorí sú aktuálne pripojení. Pri pripojení pridá nový záznam a aktualizuje ho, po odpojení ho vymaže. Hodnoty uhlov pri natočení robota k nasledujúcemu bodu a uhlov medzi robotom a používateľmi bolo nutné upraviť, keďže reprezentácia *A-Frame* rotácií a rotácií, ktoré očakáva ovládač sa

líšia v rozsahu. Rozdiely sú vyobrazené na obrázku 4.4. Uhol získaný zo scény bolo nutné preklopiť na vertikálnej osi a normalizovať na rozsah $0°$ až $360°$.



Obr. 4.4: Reprezentácia uhlov v *A-Frame* (a) a reprezentácia uhlov ovládača (b)

Robot postupne prechádza všetkými pozíciami vo svojom zozname, až kým ich všetky nevyčistí. Po vyčistení všetkých pozícií sa vypne a ostane na mieste. Ovládač je navrhnutý tak, aby zabezpečil presun medzi pozíciami a ich čistenie s ohľadom na bezpečnosť ľudí v okolí robota. Svoju aktuálnu činnosť a stav prehľadne zobrazuje tým, že mení farby svojej vrchnej časti. Celkovo môže nastať niekoľko prípadov:

- Robot sa presúva na pozíciu a nikto sa v jeho smere nenachádza - v tomto prípade svieti robot na zeleno, čo značí, že má voľnú cestu a presúva sa.
- Robot sa presúva na pozíciu a v jeho smere sa nachádza používateľ - v tomto prípade začne robot vykonávať úhybný manéver, pričom svieti na červeno, čím dáva najavo, že musel prerušiť svoju činnosť, pretože sa niekto nachádza príliš blízko.
- Robot dorazil na pozíciu a nikto sa v okolí bezpečnej vzdialenosti na čistenie nenachádza - robot začne s čistením pozície a počas toho svieti na oranžovo, čím indikuje, že prebieha čistenie pozície a nikto by sa nemal približovať.
- Robot dorazil na pozíciu a niekto sa nachádza príliš blízko alebo robot už čistí a používateľ sa priblíži - robot preruší čistenie, čo dá najavo zmenou farby na červenú.
- Robot dokončil svoju prácu, teda vyčistil všetky pozície v zozname - v tomto stave sa robot vypne, čo indikuje zmenou farby na bielu.

5 Návrh a implementácia virtuálneho prostredia pre evaluáciu prototypov riadenia dronov

Implementáciu sme realizovali do prostredia, ktoré vzniklo počas diplomovej práce Michala Ivana [33]. Výstupom jeho práce bolo rozšírenie systému *LIRKIS G-CVE* o komponenty obsahujúce funkcionalitu pre zabezpečenie prihlasovania používateľov do systému (role administrátora, používateľa a cBot-a), získavanie informácií o ostatných prihlásených používateľoch a entitách (z hľadiska nášho zamerania využijeme najmä informácie o pozícii a rotácii) a manažment používateľov administrátorským rozhraním. Budeme pokračovať v rozširovaní tohto systému a integrácie do už existujúcej štruktúry.

Pred samotným návrhom a implementáciou sme vytvorili oddelenú HTML stránku, v ktorej budú definované entity pre drona, prostredie a jeho body záujmu. Taktiež sme pridali funkcionalitu prihlásenia a overenia totožnosti serverom, ale aj grafické rozhranie pre prihlásenie, rovnako ako to riešil Michal vo svojej práci. Okrem toho sme do úvodnej stránky systému dodali tlačidlo, ktoré nám umožní sa do prostredia prihlásiť ako dron.

A-Frame poskytuje možnosť definovať atribúty komponentov priamo v HTML bez toho, aby sme zasahovali do kódu. Využijeme túto funkcionalitu na navrhnutie komponentov tak, aby bolo možné nakonfigurovať všetky potrebné parametre pre simuláciu a riadenie drona priamo v HTML entite.

5.1 Typ modelovaného dronu a spôsob jeho ovládania

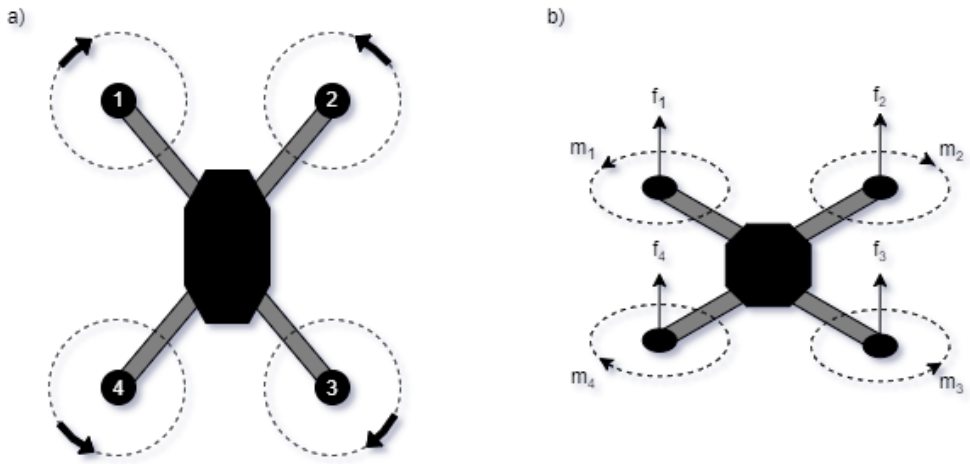
Na začiatok je nutné vytvoriť v prostredí verný model drona, ktorý by zodpovedal svojim správaním a stavbou reálnym strojom, ktoré sú v súčasnosti využí-

vané v civilnom sektore. Ako bolo spomínané v analýze 3.3, najrozšírenejšími typmi komerčných dronov sú kvadrokoptéry. Existujú dva spôsoby konfigurácie, a to konfigurácia "X"a "+". Rozloženie typu "X"v podstate nahradilo druhý typ, pretože takýto dron je obratnejší a záťaž motorov pri vykonávaní pohybu je rovnomernejšia. Preto budeme realizovať návrh a implementáciu simulátora pre túto konfiguráciu.

Grafický model kvadrokoptéry sme získali bezplatne z online databázy *3D Warehouse* [34]. Ten sme si stiahli vo formáte pre program *SkecthUp*, preto sme si ho importovali do modelovacieho programu *Blender*, v ktorom sme vykonali drobné úpravy tak, aby bol vizuálne korektný. *A-Frame* podporuje formáty `obj` a `gltf`, dôsledkom čoho nie je možné úplne presne exportovať model v inom formáte a zachovať všetky nastavenia. V našom prípade sme sa rozhodli použiť formát `gltf`, v ktorom po exporte mali niektoré povrchy nastavenú nesprávnu textúru.

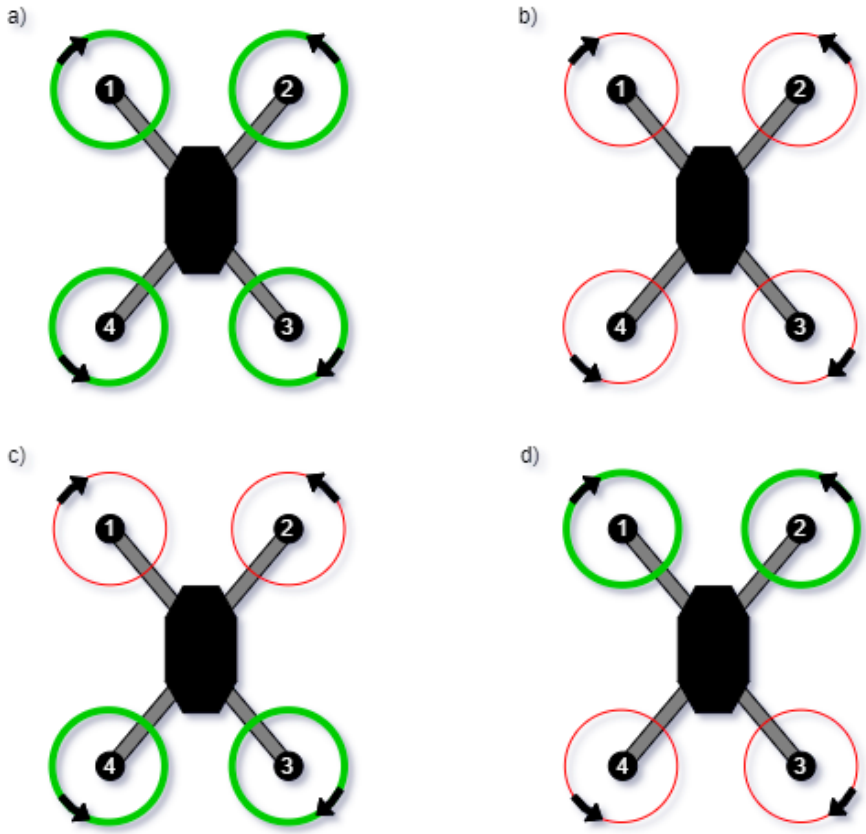
Kvadrokoptéra vykonáva svoj pohyb iba úpravou rýchlosti svojich štyroch motorov, čo z nej robí na prvý pohľad relatívne jednoduchý lietajúci stroj. Pre implementáciu modelu použijeme nasledovné rozloženie a konfiguráciu motorov, ktorá je uvedená na obrázku 5.1. Všetky motory sa po celý čas otáčajú rovnakým smerom. Konkrétne motory s číslami 1 a 3 v smere hodinových ručičiek, zatiaľ čo čísla 2 a 4 rotujú opačným smerom. Dôvodom tejto konfigurácie sú sily, ktorými pôsobia rotujúce sa lopatky na telo kvadrokoptéry. Je zrejmé, že prietok vzduchu generovaný otáčajúcimi sa lopatkami vytvára ťahové sily f_1 až f_4, ktoré sú zobrazené na obrázku 5.1. Tieto sily pôsobia vo vertikálnej osi proti gravitačnej sile. Ak je ich celkový súčet rovný veľkosti gravitačnej sily pôsobiacej na drona, výsledkom je vznášanie sa v rovnakej výške. Tento stav sa nazýva hovering. Okrem vztlakových síl vznikajú aj sily krútiaceho momentu, na obrázku 5.1 sú zobrazené ako m_1 až m_4, ktoré pôsobia v protismere rotácie daných motorov. Sily m_1 a m_3 pôsobia v opačnom smere, ako sily m_2 a m_4. Súčet týchto síl musí byť rovný nule, aby sa kvadrokoptéra nezačala točiť na mieste.

Ovládaním rýchlosti motorov ovládame aj veľkosti spomínaných síl. Kvadrokoptéra sa v 3D priestore môže pohybovať v šiestich rôznych smeroch, inak nazývaných aj šesť stupňov voľnosti (6 degrees of freedom - 6DoF). Konkrétne ide o pohyby smerom hore-dole, doľava-doprava, dopredu-dozadu a rotácie v osiach X, Y a Z. Na obrázkoch 5.2 a 5.3 je zobrazené, ako je možné ovládať pohyb drona pomocou regulácie rýchlosti otáčok jednotlivých motorov. Pohyb drona je možné ovládať tak, že ho nakloníme do požadovaného smeru a vztlak, ktorý vytvárajú motory spolu s náklonom spôsobia, že sa dron začne pohybovať v danom smere.

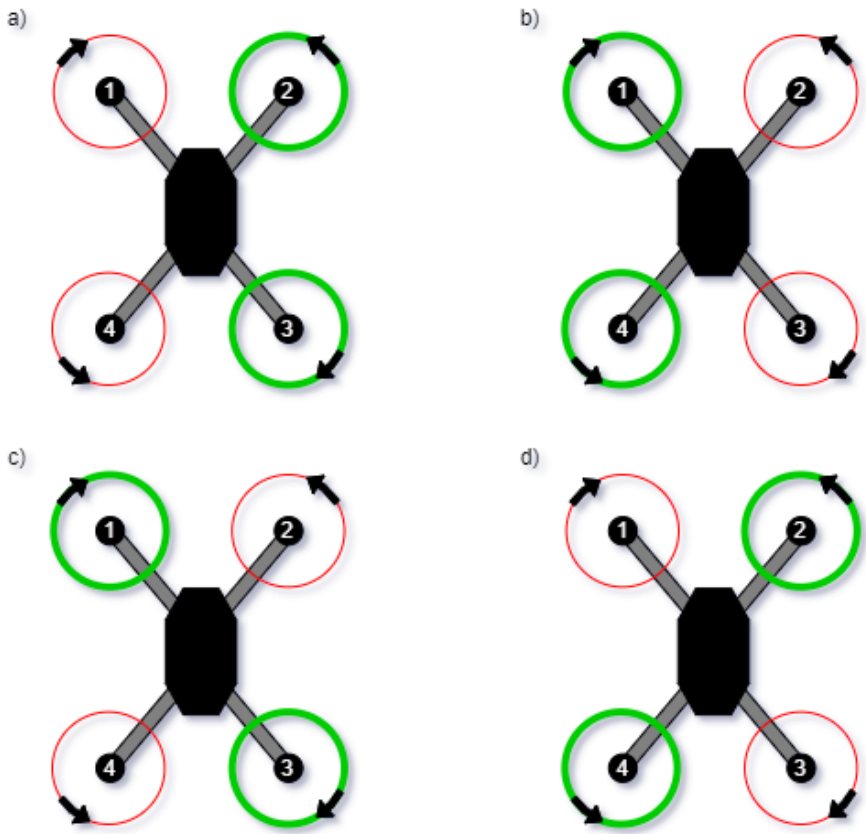


Obr. 5.1: Konfigurácia a smer rotácie motorov modelovanej kvadrokoptéry (a), fyzikálne sily generované motormi kvadrokoptéry (b)

Veľkosť odchýlky určuje, ako rýchlo sa bude dron pohybovať v zvolenom smere. Avšak tento náklon nesmie byť príliš veľký, inak by došlo k nestabilite a pádu kvadrokoptéry.



Obr. 5.2: Ovládanie pohybu kvadrokoptéry. Zelená farba predstavuje zvýšenie, červená farba zníženie výkonu daných motorov. Pohyb smerom nahor (a), nadol (b), dopredu (c) a dozadu (d)



Obr. 5.3: Ovládanie pohybu kvadrokoptéry. Zelená farba predstavuje zvýšenie, červená farba zníženie výkonu daných motorov. Pohyb smerom doľava (a), doprava (b), rotácia vľavo (c) a vpravo (d)

Pre vizualizáciu sme použili 3D model dostupný bezplatne na portáli *3dwarehouse.sketchup.com*. Stiahli sme si ho vo formáte `Collada`, ktorý sme pomocou bezplatného modelovacieho softvéru *Blender* exportovali do formátu `gltf`. Ten je možné priamo použiť v *A-Frame* prostredníctvom zabudovaného komponentu `gltf-model`.

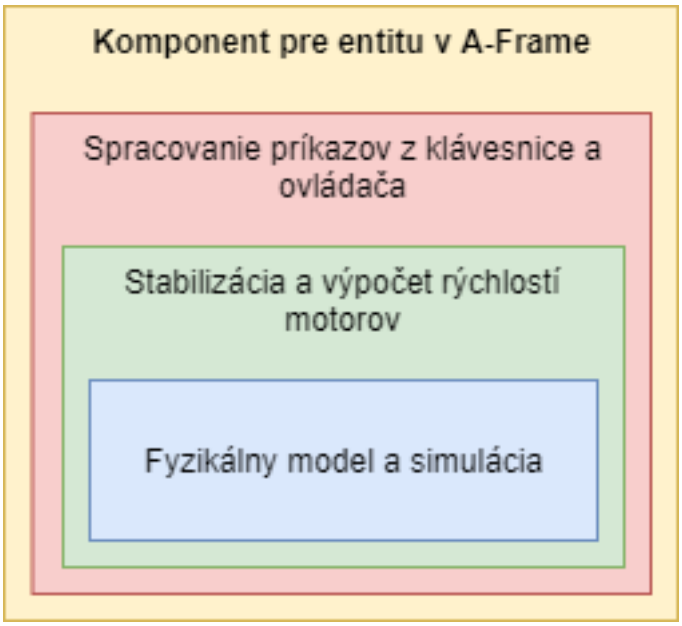
5.2 Štruktúra ovládacích komponentov kvadrokoptéry

Pohybový systém reálnych kvadrokoptér sa skladá z niekoľkých častí:

- Motory a vrtule
- Elektronický regulátor otáčok - toto zariadenie slúži na prevod riadiaceho signálu na otáčky pre motor
- Riadiaca doska so zabudovaným stabilizátorom - úlohou je reagovať na vstupy z diaľkového ovládača a podľa príkazu ovládať rýchlosti štyroch motorov

- Zložitejšie a drahšie kvadrokoptéry obsahujú aj autopilota, ktorý je súčasťou riadiacej dosky - ten umožňuje kvadrokoptéram pohybovať sa autonómne

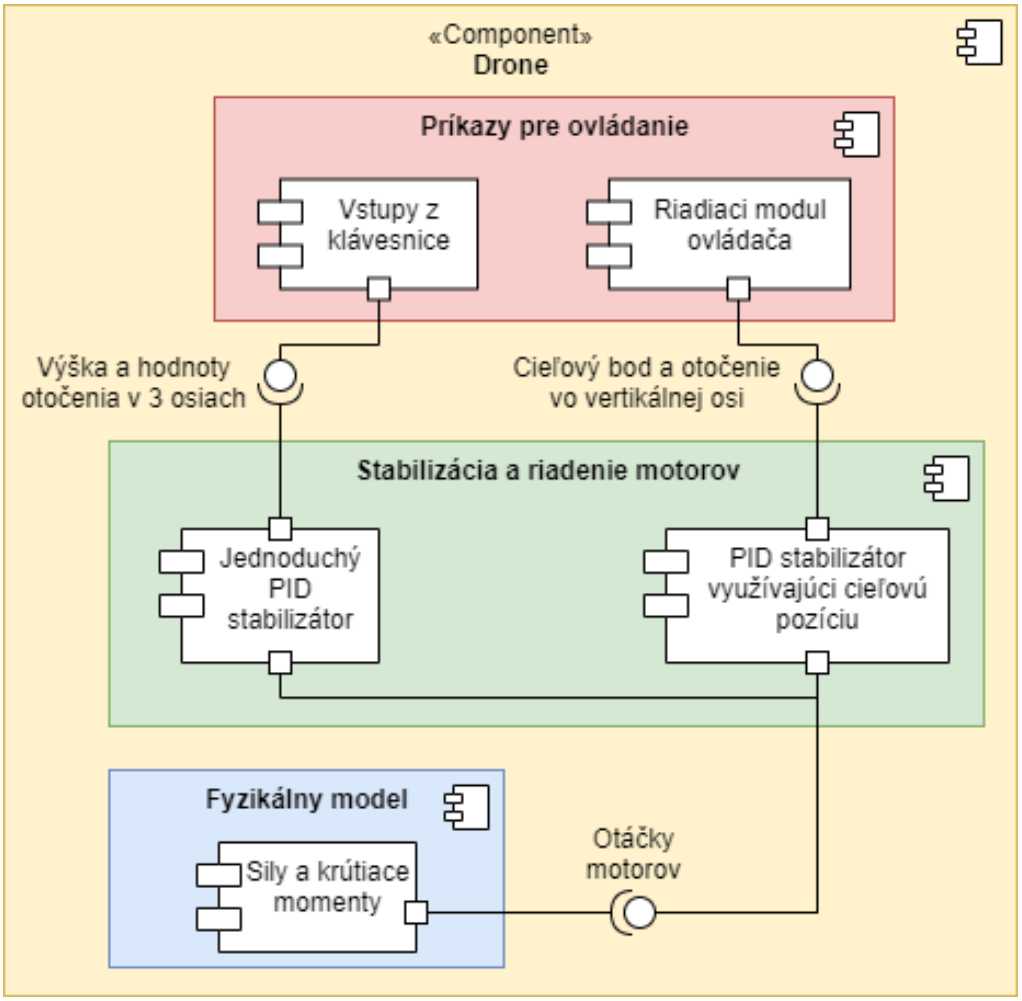
Z hľadiska virtuálneho modelu pre nás nemá význam implementovať elektronický regulátor otáčok a nebudeme sa ním zaoberať. Pri návrhu a implementácii riadiaceho modelu sa budeme riadiť podľa spomínaných častí a vytvoríme niekoľko oddelených modulov, ktoré dokopy tvoria riadiaci systém pre modelovanú kvadrokoptéru. Hierarchické usporiadanie jednotlivých častí je na obrázku 5.4.



Obr. 5.4: Hierarchia jednotlivých ovládacích komponentov

Požiadavkou na riadiaci systém je, aby boli jeho jednotlivé časti od seba nezávislé a tým pádom ich bolo možné zamieňať a testovať tak prototypy rôznych ovládacích softvérov bez nutnosti zasahovať do nízko-úrovňového riadenia a stabilizácie drona. Z toho vyplýva nutnosť implementovať jednotlivé časti riadiaceho systému ako samostatné moduly a vytvoriť k nim rozhranie, pomocou ktorého bude medzi nimi prebiehať komunikácia. Štruktúra a spôsob prepojenia modulov riadiaceho systému je zobrazený v diagrame 5.5.

Riadenie kvadrokoptéry pozostáva z troch hlavných častí. Na najnižšej úrovni sa nachádza fyzikálny model drona. Ten definuje správanie sa kvadrokoptéry podľa reálnych fyzikálnych zákonov, teda ako má teleso reagovať na pôsobiace sily a kolízie v priestore. Nad ním sa nachádzajú PID regulátory, ktoré slúžia na stabilizáciu stroja počas letu a taktiež na regulovanie otáčok motorov podľa rozdielu medzi požadovaným a súčasným stavom. Rozhodli sme sa implementovať dva druhy, ktoré nám umožňujú vytvoriť odlišné spôsoby riadenia. Na najvyššej úrovni sme implementovali tri rôzne metódy pre ovládanie drona, a to manuálne

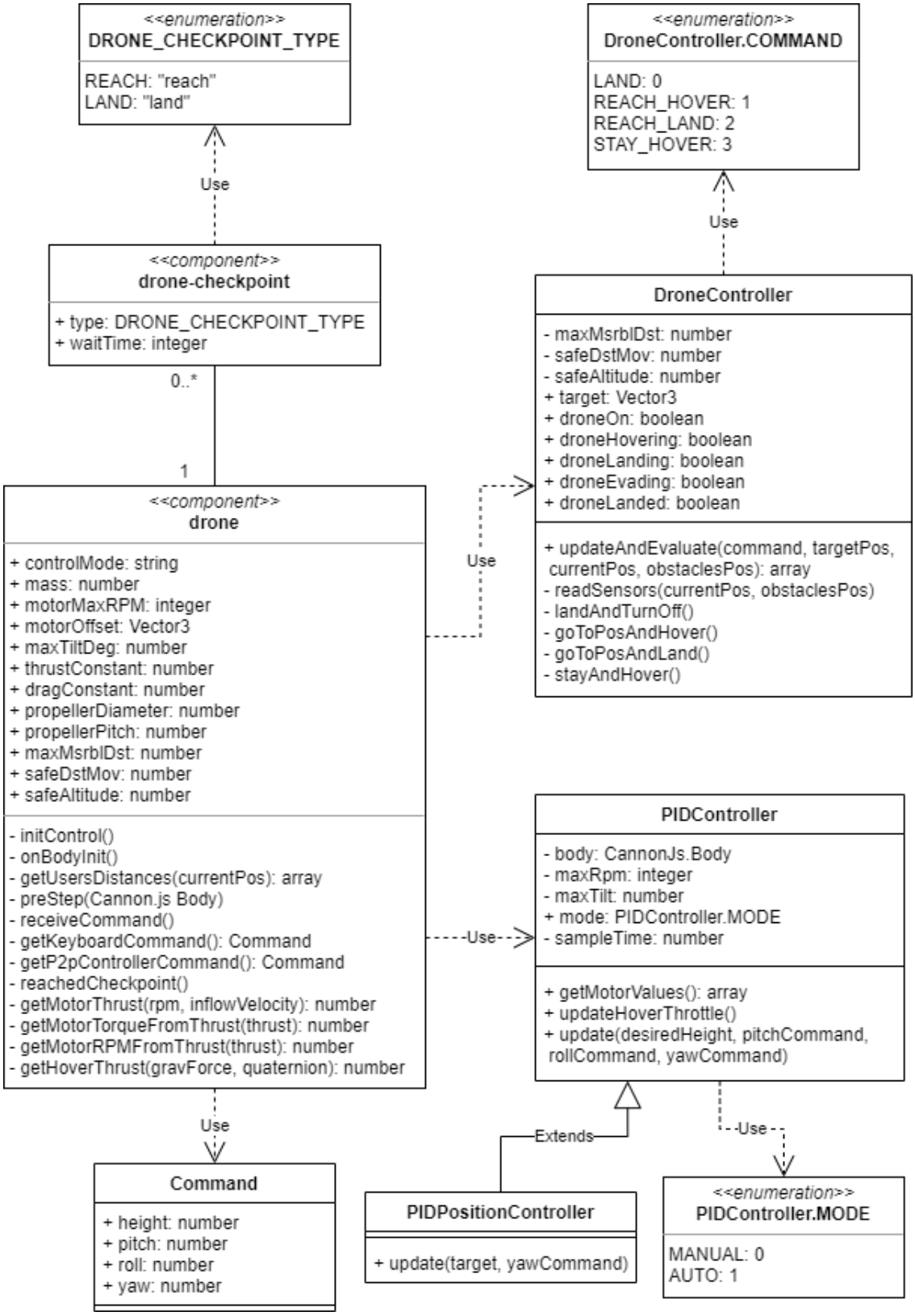


Obr. 5.5: Konceptuálny diagram riadiaceho systému kvadrokoptéry

pomocou klávesnice, alebo automaticky. Pre automatické riadenie sme vytvorili dva moduly. Jeden využíva v práci už spomínaný riadiaci softvér pre robotický vysávač (cBot). Druhý sme implementovali so zameraním špecificky pre použitie v kvadrokoptérach.

5.3 Implementácia fyziky a fyzikálneho modelu

Aby sme mohli vytvoriť prostredie s realistickou simuláciou lietajúcej kvadrokoptéry a kolízií medzi objektmi, potrebujeme vytvoriť fyzikálny model a určiť, akým spôsobom budú detekované a riešené kolízie. Z hľadiska implementácie fyzikálneho modelu bolo v kapitole 3.3 analyzovaných niekoľko systémov, z ktorých väčšina implementovala fyzikálny model bez použitia fyzikálneho rámca. Výhodou takéhoto prístupu je, že tento model bude realistickejší a viac zodpovedať reálnemu správaniu sa drona. Avšak implementácia je omnoho náročnejšia a vyžaduje si riešenie diferenciálnych rovníc pre výpočty zmeny pozície a rotá-



Obr. 5.6: Diagram tried implementovaného systému

cie objektu. Riešenie týchto výpočtov samostatne, teda bez využitia fyzikálneho rámca, je zložité a vyžaduje si veľké množstvo systémových prostriedkov, čo je

pre použitie vo webovej virtuálnej realite nevhodné. Existujú spôsoby, akými je možné zvýšiť rýchlosť takýchto výpočtov, tým sme sa ale nezaoberali.

5.3.1 Výber a použitie vhodného fyzikálneho rámca

Počas prieskumu existujúcich systémov sme narazili na systém *AirSim* [22], ktorý využíva fyzikálny rámec na prepočítavanie zmien pozície a rotácie pomocou zjednodušených rovníc. Autori implementovali výpočty pre určenie veľkosti síl generovaných motormi a samotnú aktualizáciu fyzikálneho objektu obstaráva fyzikálny rámec. Tento prístup predstavuje kompromis medzi realistickou simuláciou, výkonom a presnosťou simulácie. Okrem toho potrebujeme aj sledovať a realisticky reagovať na kolízie medzi objektmi. Navyše tu existuje možnosť, že by sme chceli rozšíriť prostredie o iné fyzikálne objekty, pre ktoré by sme inak museli implementovať iný fyzikálny model. Z týchto dôvodov sme sa rozhodli použiť samostatný fyzikálny rámec. Na tento účel sme do prostredia pridali už spomínaný komponent `aframe-physics-system`, pomocou ktorého vieme realizovať fyzikálne simulácie v prostredí.

V analýze 1 platformy *LIRKIS G-CVE* sme zistili, že máme na výber z dvoch fyzikálnych rámcov - *Ammo.js* a *Cannon.js*. Vzhľadom na výkonnosť a kvalitu dokumentácie jednotlivých rámcov sme sa rozhodli použiť *Cannon.js*, nakoľko funkcionalitu, ktorú nám ponúka, bohate postačuje na simulovanie lietajúceho drona.

Pre zapnutie fyzikálnej simulácie v prostredí stačí pridať hlavnej scéne v *A-Frame* komponent `physics`. Prostredníctvom jeho atribútov vieme konfigurovať parametre fyzikálneho sveta, ako napríklad počet iterácií počas riešenia kontaktov a pohybu, veľkosť gravitačnej sily a trenia v prostredí, alebo možnosť zapnúť vykonávanie výpočtov na oddelenom procese. My sme využili nastavenie gravitácie na hodnotu $9,80665$. Po pridaní komponentu môžeme definovať objektom, ako sa majú správať.

Na tento účel *Cannon.js* ponúka tri druhy simulovaných objektov:

- statický - objekt sa javí, akoby mal nekončenú hmotnosť, nie je ovplyvnený silami ani gravitáciou, nie je pohyblivý a nekoliduje s inými statickými a kinematickými objektmi
- kinematický - veľmi podobný, ako statický, avšak môže sa pohybovať, no iba priamym nastavením rýchlosti v danom smere
- dynamický - tento typ objektov je plne simulovaný, má určitú hmotnosť a reaguje na pôsobenie síl a gravitácie, následkom ktorých sa pohybuje, koliduje so všetkými typmi objektov

V našom prostredí je zem, alebo podlaha statický typ objektu. Entitám používateľov sme nastavili kinematický typ objektu a jeho pohyb realizujeme nastavením rýchlosti v dvoch osiach. Pre model kvadrokoptéry sme zvolili dynamický typ, vďaka čomu vieme simulovať kolízie s ostatnými objektmi a realizovať realistické správanie sa definovaním síl pôsobiacich na objekt, rovnako, ako v prípade simulátora *AirSim*. Okrem toho mu vieme nastaviť konkrétnu hmotnosť. Rozhodli sme sa použiť hmotnosť jedného z veľmi populárnych komerčne používaných dronov, model *DJI Phantom 4 Pro*, ktorý váži 1.388 kg. Samozrejme je možné túto hodnotu nastaviť podľa uváženia.

5.3.2 Realizácia fyzikálneho modelu

Rozhodli sme sa realizovať výpočty zmien pozície a rotácie kvadrokoptéry využitím fyzikálneho rámca. Na uvedenie modelu drona do pohybu využijeme sily, ktoré je možné v *Cannon.js* aplikovať priamo v konkrétnom bode daného objektu s určitou veľkosťou a smerom. Podľa obrázka 5.1 pôsobí každý motor dvoma silami na telo kvadrokoptéry: vztlakovou silou a silou točivého momentu.

Pre výpočet veľkosti vztlakovej sily využijeme nasledovný vzorec pre výpočet dynamického vztlaku generovaného rotormi [1]:

$$F = \rho * \frac{\pi * (0.0254 * d)^2}{4} * [(RPM * 0.0254 * \frac{p}{60})^2 - (RPM * 0.0254 * \frac{p}{60}) * V_0] * (\frac{d}{3.29546 * p})^{1.5}$$

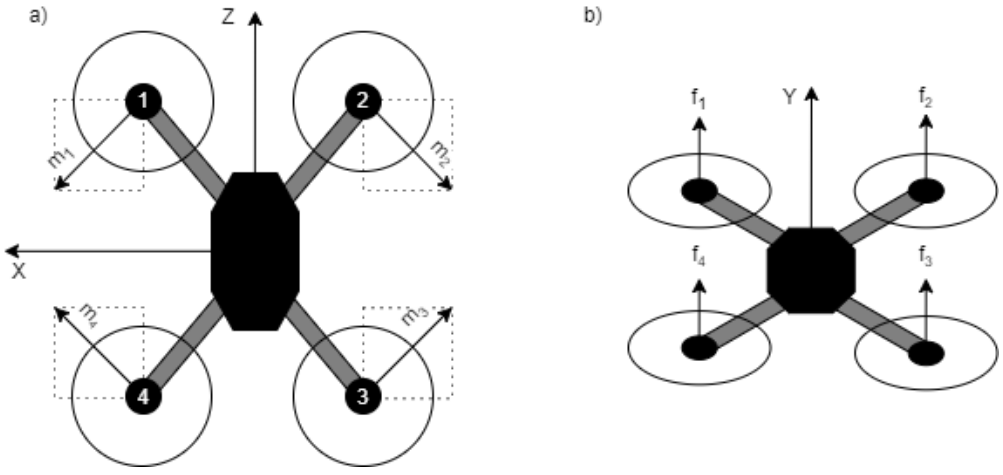
ρ predstavuje hustotu vzduchu, ktorú sme v systéme nastavili na hodnotu rovnú hustote vzduchu pri hladine mora, teda $1.225 \frac{kg}{m^3}$. d predstavuje priemer lopatiek rotora v palcoch. RPM označuje počet rotácií za minútu. p reprezentuje sklon lopatiek v palcoch. Táto hodnota predstavuje vzdialenosť, o ktorú by sa lopatky posunuli dopredu počas jednej otáčky, keby sa pohybovali cez tuhé teleso, napríklad ako sa skrutka pohybuje pri rotovaní do dreva. V_0 je rýchlosť vzduchu vstupujúceho do rotorov, ktorú sme ale z dôvodu zjednodušenia nepoužili pri výpočtoch. Keďže my používame metrické jednotky, vytvorili sme pomocné funkcie pre prevod palcov na centimetre. Rozmery v palcoch si uložíme do premenných, aby sme nemuseli vykonávať prevod jednotiek zakaždým, ak chceme zistiť veľkosť vztlakovej sily. Tento vzorec sme integrovali do komponentu tak, aby bolo možné jednoducho nastaviť parametre lopatiek a simulovať tak rôzne modely kvadrokoptér. My sme ostali pri modeli spomínanom vyššie a použili roz-

[1] Gabriel Staples, 2013. `https://www.electricrcaircraftguy.com/2013/09/propeller-static-dynamic-thrust-equation.html`

mery jedného druhu lopatiek určených pre tento typ dronu. Konkrétne sa jedná o lopatky s priemerom 24 cm a sklonom 13.97 cm.

Na výpočet veľkosti točivého momentu generovaného motormi používame iný prístup, keďže točivý moment závisí od mnohých faktorov, ako napríklad fyzikálne rozmery motora a jeho rotora, stavba elektromotora, veľkosť a rozmery rotujúcej šachty, na ktorú je pripevnený rotor, ale aj veľkosť a rozmery lopatiek. Namiesto toho sme sa rozhodli využiť pomer medzi vztlakovou silou a točivým momentov, ktorý je pre každý stroj iný a je definovaný pomerom medzi faktormi pre vztlak a krútiaci moment. Tieto faktory je potrebné experimentálne zmerať na reálnom zariadení. Pre naše účely sme použili hodnoty jedného z analyzovaných simulátorov, konkrétne systému *MulticopterSim*. Používateľ si ich ale môže upraviť podľa toho, aký model kvadrokoptéry chce simulovať. Pomocou získaného pomeru medzi týmito silami vieme prostredníctvom vypočítanej vztlakovej sily určiť veľkosť generovaného momentu pre každý z motorov.

Posledným krokom je aplikovanie síl v danom smere a bode na simulovaný objekt kvadrokoptéry. Na tento účel sme použili funkciu *Cannon.js* - `applyLocalForce`, ktorá aplikuje silu na lokálnu pozíciu objektu. Jednotlivé pozície sme nastavili tak, aby zodpovedali vizuálnemu modelu drona, ktorý sme použili. Taktiež je ale možné tieto pozície nastaviť ľubovoľne prostredníctvom atribútov komponentu. Funkcia `applyLocalForce` má dva parametre: vektor sily a pozícia miesta pôsobenia. V tomto vektore sme skombinovali vztlakovú a točivú silu. Výsledkom sú vektory na obrázku 5.7.



Obr. 5.7: Smer aplikovania síl krútiaceho momentu (a) a vztlakovej sily (b) na model kvadrokoptéry vzhľadom na osi vo virtuálnom prostredí

Keďže vektor sily je trojrozmerný, definujeme v ňom veľkosti pre jednotlivé osi. Krútiaci moment pôsobí na kvadrokoptéru v rovine XZ, preto sme museli

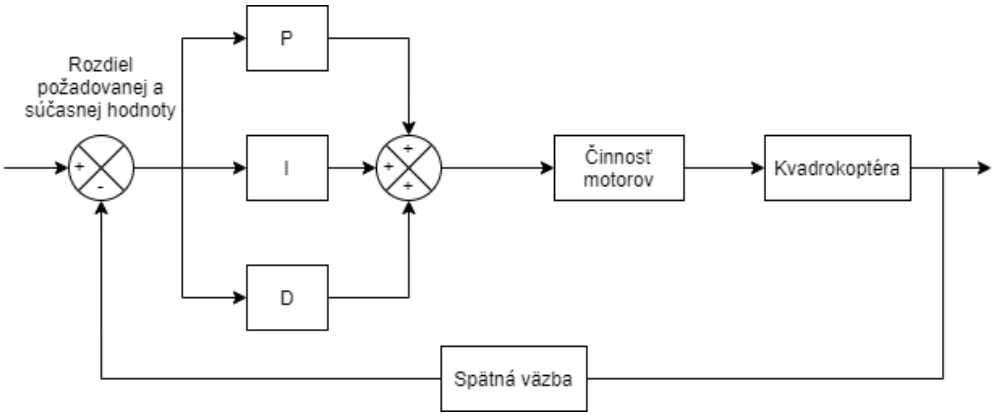
výslednú hodnotu upraviť použitím Pytagorovej vety, čo znamená, že sme ju vydelili druhou odmocninou čísla dva. Celkovo sme takto dosiahli celkom realistickú simuláciu fyzikálneho modelu drona, ktorý je ovplyvňovaný parametrami, ako sú napríklad rozmery a tvar lopatiek a hustota vzduchu.

5.4 Stabilizácia kvadrokoptéry

Z výsledkov analýzy v časti 3.3 sme sa dozvedeli, že jednoduchý a spoľahlivý spôsob, ako implementovať stabilizáciu kvadrokoptéry je využitím PID regulátora. Štruktúra takéhoto regulátora je na obrázku 5.8. Výpočet pozostáva z troch samostatných zložiek, ktoré sa vo výsledku sčítajú. Ide o zložku proporcionálnu (P), integračnú (I) a derivačnú (D). Každá z nich má odlišný vplyv na správanie sa regulátora:

- Proporcionálna zložka - určuje veľkosť vplyvu rozdielu medzi cieľovou a aktuálnou hodnotou
- Integračná zložka - redukuje odchýlky pre dosiahnutie stabilného stavu
- Derivačná zložka - znižuje čas potrebný na dosiahnutie stabilného stavu a redukuje prekročenie cieľovej hodnoty

Pre zjednodušenie, proporcionálna zložka pracuje s veľkosťou rozdielu medzi požadovaným a aktuálnym stavom. Integračná zložka sleduje, ako dlho sme ešte nedosiahli požadovaný stav a derivačná zložka sleduje, ako rýchlo sa k požadovanému stavu približujeme. Veľkosť a vplyv týchto zložiek sa určuje konštantami, ktoré je potrebné nastaviť tak, aby sa regulátor dostatočne rýchlo priblížil k cieľovej hodnote, ale neprekračoval ju, alebo nezačal oscilovať.



Obr. 5.8: Štruktúra PID regulátora so spätnou väzbou

Typ regulátora, ktorý je na obrázku 5.8 použijeme na výpočet otáčok jednotlivých motorov potrebných na dosiahnutie určitého definovaného stavu. Stav kvadrokoptéry v priestore je definovaný niekoľkými premennými:

- pozícia v trojrozmernom priestore (súradnice X, Y a Z)
- rýchlosť pohybu v každej z osí
- uhly natočenia v trojrozmernom priestore (uhol v osiach X, Y a Z)
- rýchlosť otáčania v každej z osí

Z tohto dôvodu musíme použiť niekoľko PID regulátorov, konkrétne tri pre prácu s pozíciami a rýchlosťou pohybu a ďalšie tri pre rotácie a rýchlosti otáčania. Tieto regulátory tvoria dokopy kompletný stabilizátor, na ktorého vstupe je požadovaný stav a výstupom sú rýchlosti motorov potrebné na dosiahnutie tohto stavu. Zo štruktúry riadiaceho systému kvadrokoptéry (obrázok 5.5) je zrejmé, že nami navrhnutý systém používa dva rôzne stabilizátory. Ako prvý si popíšeme jednoduchý PID stabilizátor.

5.4.1 Jednoduchý stabilizátor

Tento stabilizátor pracuje iba so štyrmi stavovými premennými. Týmito hodnotami sú požadovaná letová hladina a uhol natočenia v každej z troch osí. Preto tento stabilizátor pozostáva iba zo štyroch samostatných PID regulátorov, narozdiel od druhého stabilizátora, ktorý ich má šesť.

Úlohou prvého regulátora je na základe požadovanej letovej hladiny určiť veľkosť ťahu potrebného na dosiahnutie danej výšky. Funguje tak, že využíva hodnotu ťahu potrebného na udržanie sa vo vzduchu (stav nazývaný hovering) a k nemu potom pripočítava alebo od neho odrátava určitú hodnotu na základe toho, či má dron stúpať alebo klesať. V prostredí *A-Frame* je výška reprezentovaná osou Y a rozmery sú udávané v metroch. Regulátor teda pracuje s aktuálnou pozíciou a rýchlosťou pohybu v tejto osi. Cieľová letová hladina a aktuálna pozícia predstavujú vstupné údaje pre regulátor, ktorý na základe nich určí aktuálnu odchýlku. Táto odchýlka je potom použitá pri výpočtoch zložiek P a I regulátora. Veľkosť zložky D určujeme prostredníctvom rýchlosti pohybu v danom smere, keďže derivačná zložka sleduje veľkosť zmeny hodnoty za určitý čas. Výpočet jednotlivých zložiek tohto regulátora je nasledovný:

- proporcionálna zložka: $P = K_P * error$
- integračná zložka: $I = I + K_I * error$

- derivačná zložka: $D = K_D * vel\_y$

Hodnoty K_P, K_I a K_D predstavujú konštanty pre jednotlivé zložky spomínané vyššie. Hodnoty týchto konštánt sme nastavovali ručne po dokončení implementácie celého stabilizátora. Tento proces sa nazýva kalibrácia. Výstupom tohto regulátora je súčet hodnôt z jednotlivých zložiek a spomínanej hodnoty potrebnej pre udržanie sa vo vzduchu. Tento výsledok predstavuje celkový ťah potrebný na dosiahnutie cieľovej výšky v danom časovom okamihu.

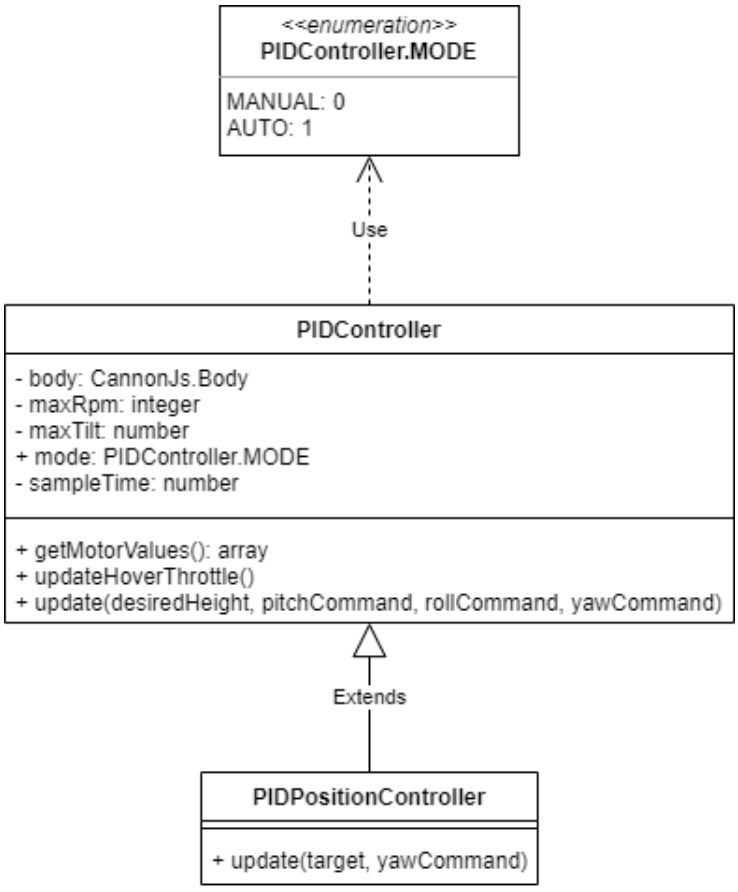
Ďalšie tri regulátory sú funkčne rovnaké a pracujú s rotáciou drona v jednej osi. Rotácie sú v *A-Frame* reprezentované quaterniónmi, preto sme si ich pre potreby výpočtov previedli na reprezentáciu pomocou Eulerových uhlov udávané v radiánoch. Každý z regulátorov obstaráva stabilizáciu v jednej z osí. Vstupnou hodnotou sú požadovaný uhol náklonu v danej osi a aktuálny uhol náklonu a podobne, ako pri regulátore pre letovú hladinu je rozdielom týchto dvoch hodnôt získaná aktuálna odchýlka. Túto odchýlku ešte normalizujeme na rozsah uhlov od $-\pi$ do π a obmedzíme hodnotou pre maximálny možný uhol náklonu v danej osi. Po úprave určíme veľkosť jednotlivých zložiek rovnakým spôsobom, ako pri prvom regulátore. Získame tak tri rôzne výstupy, z ktorých každý obsahuje úpravu rýchlosti motorov pre vykonanie pohybu v danom smere.

Všetky štyri výstupné hodnoty z jednotlivých regulátorov skombinujeme do výslednej hodnoty tak, že pokiaľ danému motoru potrebujeme zvýšiť otáčky na pohyb v kladnom smere osi, jeho zložku pripočítame ako kladné číslo. V opačnom prípade pričítame zápornú hodnotu. Napríklad ak sa dron má pohybovať dopredu, motorom 1 a 2 pripočítame zápornú hodnotu a motorom 3 a 4 kladnú. Výsledkom bude, že sa dron nakloní dopredu a zmena smeru vztlakovej sily spôsobí, že sa bude pohybovať dopredu (viď. obrázok 5.2 pre vysvetlenie zmeny rýchlostí motorov pre dané smery pohybu). Hodnoty pre jednotlivé motory sú nasledovné:

- $Motor1 = vztlak + x\_val - z\_val + y\_val$
- $Motor2 = vztlak - x\_val - z\_val - y\_val$
- $Motor3 = vztlak - x\_val + z\_val + y\_val$
- $Motor4 = vztlak + x\_val + z\_val - y\_val$

Hodnoty x, y a z_val predstavujú výstupy z regulátorov pre otočenia v danej osi. Výstupné hodnoty sme ešte obmedzili tak, aby nepresiahli definovanú maximálnu hodnotu, teda maximálne otáčky, ktoré môže motor dosiahnuť.

Pre reprezentáciu PID stabilizátora v systéme sme vytvorili triedu `PIDController`, ktorá obsahuje metódy potrebné pre interné výpočty, ale aj rozhranie pre nastavovanie a získavanie hodnôt, či manipuláciu so stabilizátorom. V konštruktore sa inicializujú všetky potrebné premenné. Pomocou jeho parametrov je možné zadefinovať obmedzenia pre maximálne otáčky motorov a maximálny uhol náklonu pre všetky osi. Okrem toho sme implementovali dva módy pre tento stabilizátor - *AUTO* a *MANUAL*. Mód AUTO používa modul pre automatické riadenie využitím ovládača pre cBot-a, zatiaľ čo MANUAL sa používa, ak je dron riadený ručne pomocou klávesnice. Rozdiel je v ovládaní rotácie drona v osi Y, kde v automatickom móde sa používa ako vstup cieľový uhol natočenia, zatiaľ čo pri manuálnom móde iba požadovaná veľkosť zmeny rotácie. Stabilizátor taktiež podporuje možnosť nastaviť frekvenciu výpočtov. Štruktúra triedy `PIDController` je zobrazená pomocou diagramu 5.9.



Obr. 5.9: Diagram tried zobrazujúci štruktúru implementovaných stabilizátorov

Výpočty hodnôt sú realizované v metóde `update`. Na vstupe dostane stabilizátor požadovaný stav kvadrokoptéry. Volanie tejto metódy má na starosti trieda, ktorá tento stabilizátor používa. Výsledok výpočtov získame pomocou metódy

`getMotorValues`.

5.4.2 Stabilizátor využívajúci cieľové pozície

Druhý implementovaný stabilizátor využíva na rozdiel od predchádzajúceho ako vstup cieľovú pozíciu v trojrozmernom priestore a požadovaný uhol otočenia v ypsilonovej osi. Z tohto dôvodu pracuje so šiestimi stavovými premennými. Oproti predošlému stabilizátoru pribudli premenné pre polohu a rýchlosť v osiach X a Z. Princíp fungovania a výpočtov je veľmi podobný, ako v prvom prípade. Odlišnosti sú v tom, že najprv tri regulátory vypočítajú hodnoty pre odchýlky od cieľovej pozície pre každú z osí a následne sa získané hodnoty použijú na výpočet uhlov náklonu v každej z osí pre vykonanie pohybu do daného smeru. Taktiež platí, že sa hodnoty ošetrujú, aby nepresiahli definovaný maximálny uhol náklonu a maximálne otáčky motorov. Výsledné hodnoty pre jednotlivé motory sú rovnaké, ako v jednoduchom stabilizátore.

Keďže sa implementácia odlišuje iba vo výpočtoch, ktoré sú implementované v metóde `update`, rozhodli sme sa využiť prvky OOP, konkrétne dedičnosť. Tento stabilizátor je rozšírením jednoduchého stabilizátora a preťažuje iba metódu `update`, ako je vidieť na diagrame 5.9. Využívame ho v module pre automatické riadenie využitím ovládača navrhnutého pre kvadrokoptéru.

5.5 Modul pre spracovanie ovládania pomocou klávesnice

Jednou z požiadaviek na návrh a implementáciu riadiaceho systému je modularita a nezávislosť jednotlivých komponentov. Z tohto dôvodu musíme vytvoriť jednotné rozhranie pre spracovávanie príkazov pre kvadrokoptéru. Na tento účel sme vytvorili štruktúru `Command`, ktorá reprezentuje príkaz odoslaný stabilizátorom na vykonanie daného pohybu. Táto štruktúra obsahuje štyri atribúty:

- `height` - požadovaná výška v metroch
- `pitch` - uhol otočenia v smere osi X určený v radiánoch
- `roll` - uhol otočenia v smere osi Z určený v radiánoch
- `yaw` - uhol otočenia v smere osi Y určený v radiánoch

Ako už bolo spomenuté, tento modul využíva na stabilizáciu kvadrokoptéry jednoduchý stabilizátor v `MANUAL` móde. Na zachytenie vstupov z klávesnice vy-

užívame rozhranie *WebAPI*, pomocou ktorého pridáme funkcie pre udalosti stlačenia a pustenia klávesy (“keydown”a “keyup”). V týchto funkciách na základe typu stlačenej klávesy určíme hodnoty štruktúry `Command`. Ovládanie kvadrokoptéry je realizované klávesami I, K, J, L pre ovládanie pohybu v rovine XZ, klávesami U a O sa ovláda otočenie v osi Y. Stúpanie a klesanie je ovládané klávesami M a N. Podľa toho, aké klávesy boli stlačené, nastavíme značky pre daný príkaz na hodnotu `true`. Po uvoľnení sa hodnota značky nastaví na `false`. Zakaždým pred vykreslením snímky sa podľa hodnôt značiek určia vstupné hodnoty pre stabilizátor nasledovne:

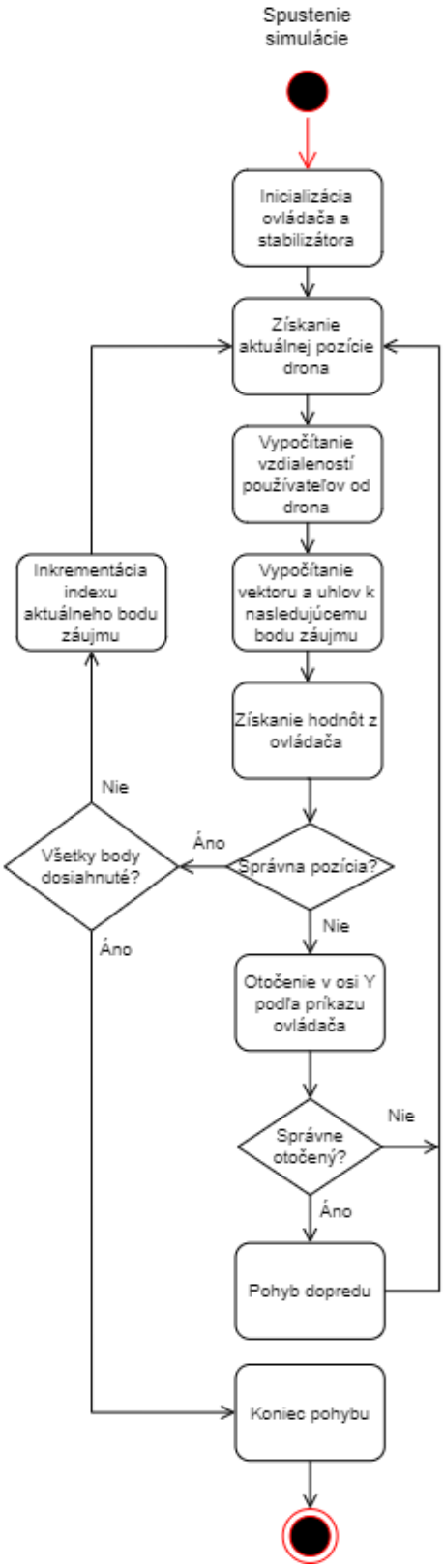
- Pohyb dopredu a dozadu - hodnota `pitch` je nastavená na $-20°$, respektíve $20°$ a prepočítaná na radiány
- Pohyb doľava a doprava - hodnota `roll` je nastavená na $-20°$, respektíve $20°$ a prepočítaná na radiány
- Otáčanie vľavo a vpravo - hodnota `yaw` je nastavená na $-180°$, respektíve $180°$ a prepočítaná na radiány
- Stúpanie a klesanie - hodnota `height` je nastavená s pomocou premennej, ktorej hodnota sa zväčšuje alebo zmenšuje

Výsledný príkaz sa použije ako vstup pre stabilizátor.

5.6 Modul pre automatické riadenie využitím riadiaceho systému pre kvadrokoptéru

Na začiatok sme sa rozhodli v tomto module použiť už existujúci ovládač, ktorý sme vytvorili a opísali v kapitole 4, a zistiť, či bude vhodný pre naše potreby. Ten sme nijakým spôsobom neupravovali. Nevyužili sme ale atribúty `safeDstCl` (bezpečná vzdialenosť pre čistenie) a `cleanRange` (dosah čistenia), keďže v prípade lietajúceho drona nemajú význam. Zvyšné atribúty je možné nastaviť rovnako, ako sme to urobili pre model robotického vysávača, pomocou atribútov HTML entity.

Pre komunikáciu medzi ovládačom a stabilizátorom sme využili rovnakú štruktúru `Command`, ako modul pre klávesnicu. Hodnoty pre túto štruktúru získavame podľa výstupov z ovládača. Tie potom odovzdávame rovnakému stabilizátoru, ako v prípade modulu pre klávesnicu, avšak tentoraz v móde `AUTO`. Princíp fungovania tohto modulu s využitím spomínaného ovládača je zobrazený v diagrame 5.10.



Obr. 5.10: Diagram aktivít zobrazujúci proces riadenia drona pomocou ovládača pre cBot-a

Modul obdrží od ovládača prikázaný uhol natočenia a vzdialenosť, v ktorej sa

má presúvať v danom smere. Uhol natočenia je možné stabilizátoru poslať priamo ako hodnotu `yaw`, keďže funguje v móde `AUTO`. Okrem toho modul nastavuje iba príkaz pre `pitch`, a to na hodnotu $-20°$, ak je dron správne natočený.

Avšak po integrácii spomínaného ovládača sme objavili niekoľko nedostatkov:

- ovládač pracuje iba v rovine XZ, nerozlišuje teda rozdiel medzi výškami a kvôli tomu nevie relevantne reagovať na okolitých používateľov
- ovládač neberie do úvahy rýchlosť a zotrvačnosť kvadrokoptéry, tým pádom nevie ostať stáť na mieste
- ovládanie prebieha iba zadaním cieľovej rotácie a vzdialenosti, čoho dôsledkom je iba schopnosť rotovať na mieste a pohybovať sa dopredu

Kvôli tomu sme sa rozhodli implementovať odlišný ovládač, ktorý by tieto nedostatky odstránil. Aby sme dosiahli to, že ovládač bude brať do úvahy rýchlosť a zotrvačnosť, potrebovali sme použiť druhý stabilizátor, ktorý pracuje s inými vstupnými hodnotami. Aby bolo zachované jednotné rozhranie, zmenili sme iba sémantiku jednotlivých hodnôt štruktúry `Command`:

- `height` - pozícia cieľového bodu v Y-ovej osi
- `pitch` - pozícia cieľového bodu v X-ovej osi
- `roll` - pozícia cieľového bodu v Z-ovej osi
- `yaw` - uhol otočenia v smere osi Y určený v radiánoch

Ovládač sme navrhli a implementovali priamo v jazyku *Javascript*, špecificky pre použitie na ovládanie kvadrokoptéry. Požiadavkou bolo, aby bol schopný ovládať drona tak, aby prešiel všetkými definovanými bodmi záujmu a zároveň sa vyhýbal kolíziam s používateľmi. Na tento účel definujeme počas inicializácie ovládača tri konfigurovateľné atribúty: `maxMsrblDst` (dosah senzorov), `safeDstMov` (bezpečná vzdialenosť pre pohyb) a `safeAltitude` (bezpečná letová výška). Prvé dva atribúty majú rovnaký význam, ako v ovládači pre cBot-a. Atribút bezpečnej letovej výšky určuje ovládaču, v akej výške by sa mal dron udržiavať počas presunu medzi pozíciami alebo uhýbaniu sa používateľom. Rozhodli sme sa použiť takýto prístup, pretože predstavuje jednoduchý spôsob, akým sa môže dron efektívne a bezpečne (s ohľadom na používateľov) presúvať na určené miesta.

Pre potreby simulovania rôznych scenárov lietania kvadrokoptéry podporuje ovládač štyri druhy príkazov:

- `LAND` - dron preruší aktuálnu činnosť a bezpečne pristane na aktuálnej pozícii. Ak sa vo vzdialenosti menšej ako `safeDstMov` niekto nachádza, dron preruší pristávaciu procedúru a vráti sa do bezpečnej letovej hladiny, kde počká, kým sa používateľ vzdiali na vzdialenosť väčšiu, ako je hodnota `safeDstMov`.
- `REACH_HOVER` - dron sa presúva k zadanej pozícii v bezpečnej letovej výške. Po dosiahnutí pozície sa ostane vznášať na mieste.
- `REACH_LAND` - dron sa presúva k zadanej pozícii rovnako, ako v prípade `REACH_HOVER`. Po dosiahnutí pozície pristane na danej pozícii rovnako, ako v prípade príkazu `LAND`.
- `STAY_HOVER` - dron preruší aktuálnu činnosť a ostane sa vznášať na mieste v bezpečnej letovej hladine.

Taktiež sme ošetrili prípad, keď sa dron nachádza na zemi a potrebuje odštartovať. V tomto prípade taktiež kontroluje, či sa niekto nachádza bližšie, než je bezpečná vzdialenosť. Ak sa v danej vzdialenosti nikto nenachádza, dron odštartuje a pokračuje podľa zadaného príkazu. V opačnom prípade počká, až kým priestor v jeho okolí nebude prázdny. Pre zachovanie modularity sme implementovali tento ovládač tak, aby poskytoval rovnaké rozhranie, ako v prípade druhého spomínaného ovládača. To znamená, že po inicializácii s ním pracujeme už iba volaním metódy `updateAndEvaluate`. Vstupnými hodnotami sú jeden zo štyroch spomínaných príkazov, cieľová a aktuálna pozícia drona a zoznam pozícií používateľov, ktorý získavame prostredníctvom komponentu pre komunikáciu medzi entitami. Výstupom funkcie je cieľová pozícia, ktorá sa použije v štruktúre `Command` ako vstup pre stabilizátor. Okrem toho vracia informácie o aktuálnom stave drona, konkrétne či je dron zapnutý, či sa vznáša, či prebieha úhybný manéver, či dron pristáva a či už je na zemi. Princíp fungovania tohto modulu je zobrazený v diagrame 5.11.

5.7 Hlavný komponent pre entitu kvadrokoptéry

Tento komponent s názvom *drone* kombinuje vyššie spomínané moduly do jedného celku a obstaráva inicializáciu modelu kvadrokoptéry, fyzikálneho modelu, dvoch rôznych stabilizátorov a ovládača pre kvadrokoptéru. Okrem toho poskytuje rozhranie pre konfiguráciu simulovaného modelu prostredníctvom už spomínaných HTML atribútov. Zoznam konfigurovateľných hodnôt je nasledovný:

- `controlMode` - `keyboard` pre ovládanie klávesnicou, `auto` pre automatické riadenie pomocou ovládača pre cBot-a a `p2pauto` pre automatické riadenie pomocou ovládača pre kvadrokoptéru
- `mass` - hmotnosť kvadrokoptéry v kilogramoch
- `motorMaxRPM` - maximálne otáčky motorov
- `motorOffset` - pozície motorov vzhľadom na stred kvadrokoptéry
- `maxTiltDeg` - maximálny uhol náklonu v stupňoch
- `thrustConstant` - faktor pre vztlak motorov
- `dragConstant` - faktor pre krútiaci moment motorov
- `propellerDiameter` - priemer lopatiek rotorov v centimetroch
- `propellerPitch` - sklon lopatiek v centimetroch
- `maxMsrblDst` - maximálny dosah senzorov pre automatické riadenie v metroch
- `safeDstMov` - bezpečná vzdialenosť pre pohyb v metroch
- `safeAltitude` - bezpečná letová hladina v metroch

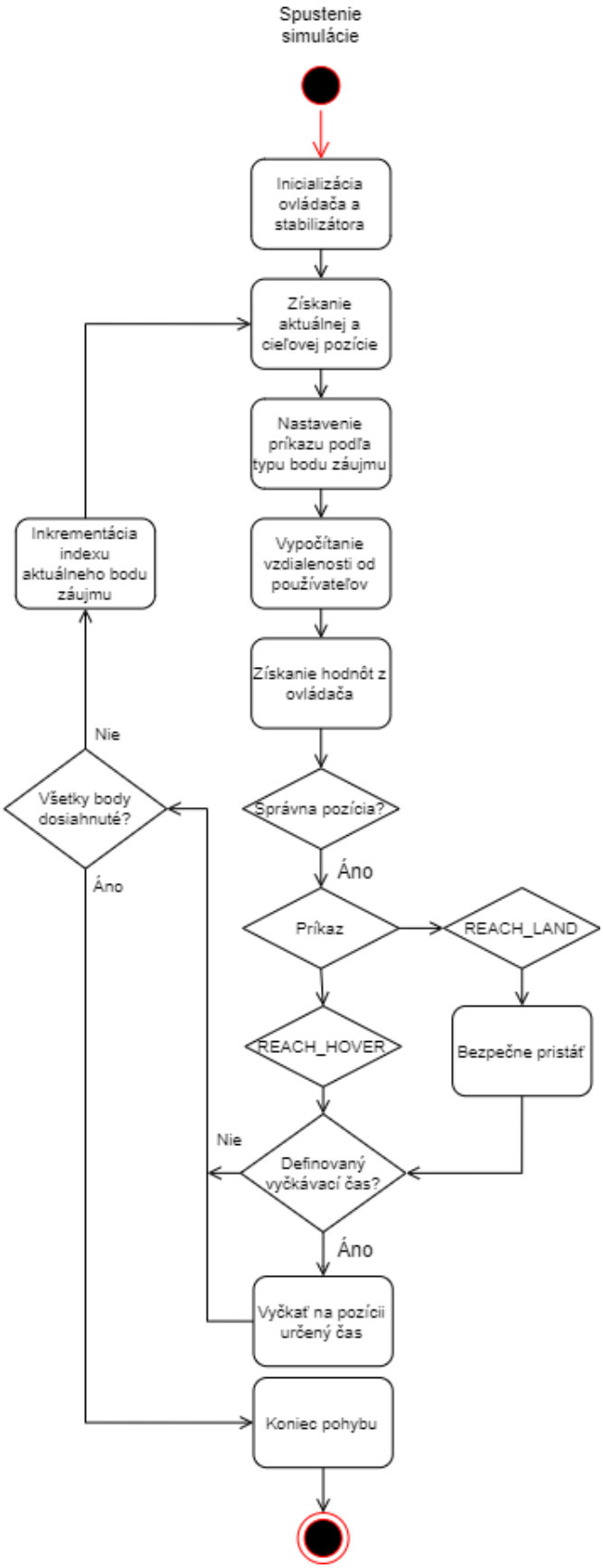
Navyše počas inicializácie si tento komponent automaticky vytvorí zoznam bodov záujmu, ktoré má dron nasledovať. Jeho aktualizáciu má na starosti modul automatického riadenia. Využíva na to metódu `reachedCheckpoint` tohto komponentu. Navrhli sme tento komponent a všetky moduly tak, aby bolo možné ich v prípade potreby jednoducho meniť. Dosiahli sme to definovaním rozhraní, pomocou ktorých jednotlivé moduly komunikujú. Tento komponent modularitu implementuje tak, že využíva štruktúru `Command` na oddelenie stabilizátorov od ovládačov. Na tento účel sme vytvorili metódu `receiveCommand`, ktorá podľa nastaveného atribútu `controlMode` získava údaje od relevantného modulu. Tie sú potom odovzdané stabilizátorom. Po výpočte rýchlostí motorov ich potom hlavný komponent použije vo fyzikálnom modeli. Tento komponent je možné pridať rôznym entitám vo virtuálnom prostredí a vytvoriť tak z nich fyzikálne simulovaného drona.

5.8 Komponent bodov záujmu pre drona

Aby bolo možné jednoducho definovať body záujmu, ktoré má dron dosiahnuť, potrebovali sme vytvoriť ďalší komponent *drone-checkpoint*, ktorý by definoval entitu ako pozíciu pre drona. V analýze 3 sme zistili rôzne možnosti využitia komerčných dronov. Aby sme vedeli takéto rôzne prípade použitia simulovať v našom prostredí, vytvorili sme dva druhy značiek:

- `reach` - označený zelenou farbou, tento typ bodu určuje dronu, že je potrebné doraziť na danú pozíciu a v prípade, že ďalšia značka neexistuje tam aj ostať
- `land` - označený červenou farbou, tento typ bodu určuje, že na tomto mieste je potrebné pristáť

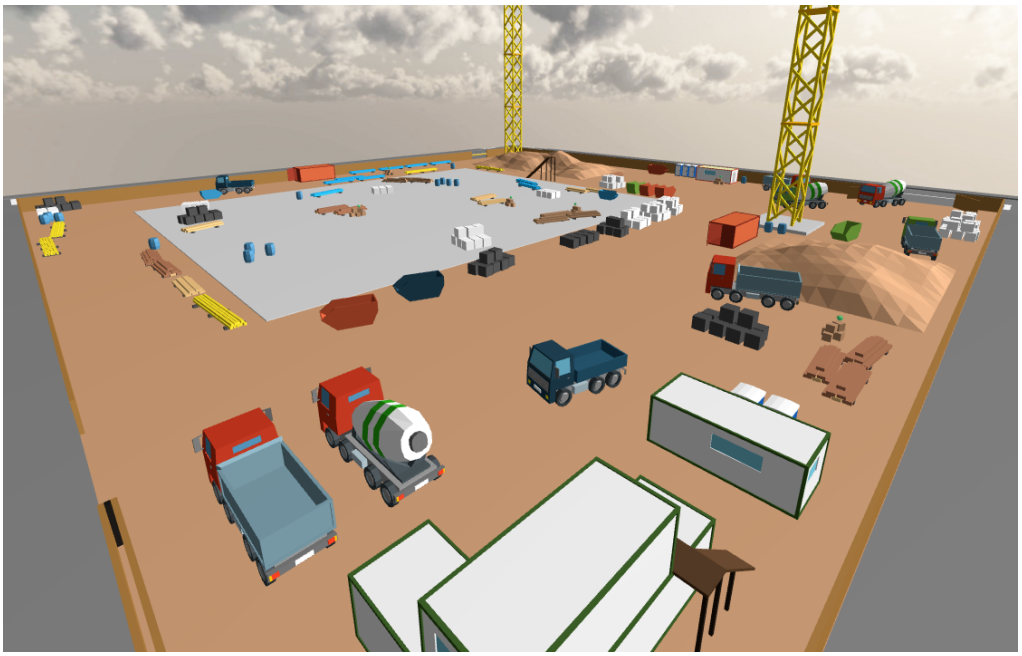
Okrem toho sme implementovali atribút `waitTime`, ktorým môžeme zadefinovať, ako dlho má dron na danej pozícii počkať. V prípade značky typu `reach` ostane dron čakať vo vzduchu. Pri značke typu `land` vyčká na zemi. Pre pridanie novej značky stačí pridať do scény novú entitu s týmto komponentom a nastaviť jej typ a čas.



Obr. 5.11: Diagram aktivít zobrazujúci proces riadenia drona pomocou ovládača navrhnutého pre kvadrokoptéru

6 Zhodnotenie implementovaného systému

Aby sme mohli overiť použiteľnosť nami navrhnutého a implementovaného systému, vytvorili sme konkrétny scenár, ako pre používateľa, tak aj drona. Rozhodli sme sa ich zasadiť do prostredia prebiehajúcej výstavby novej budovy. Scéna obsahuje množstvo objektov, ktoré sa bežne vyskytujú v takomto prostredí, ako napríklad stavebný materiál, nákladné vozidlá, statické žeriavy, či kontainery náradie a stroje. Všetky tieto objekty sú nastavené tak, aby cez nich používateľ, ani dron nevedeli prechádzať. 3D model prostredia sme získali online z `https://majadroid.itch.io/3d-house-construction-site`, ktorý sme po drobných úpravách nasadili do scény. Model kvadrokoptéry už bol opísaný v implementácii.



Obr. 6.1: Vytvorené virtuálne prostredie použité pri vyhodnotení riešenia

Dron sa v tomto prostredí presúva autonómne medzi šiestimi pozíciami, kde na určitú dobu pristane a potom pokračuje ďalej. Vizuálnu odozvu pre používa-

teľov sme implementovali zmenou farieb tela drona. Zelená signalizuje presun na pozíciu, oranžová vykonávanie úkonu po pristátí a červená prerušenie aktuálnej činnosti a reakcia na používateľa, ktorý sa nachádza príliš blízko.

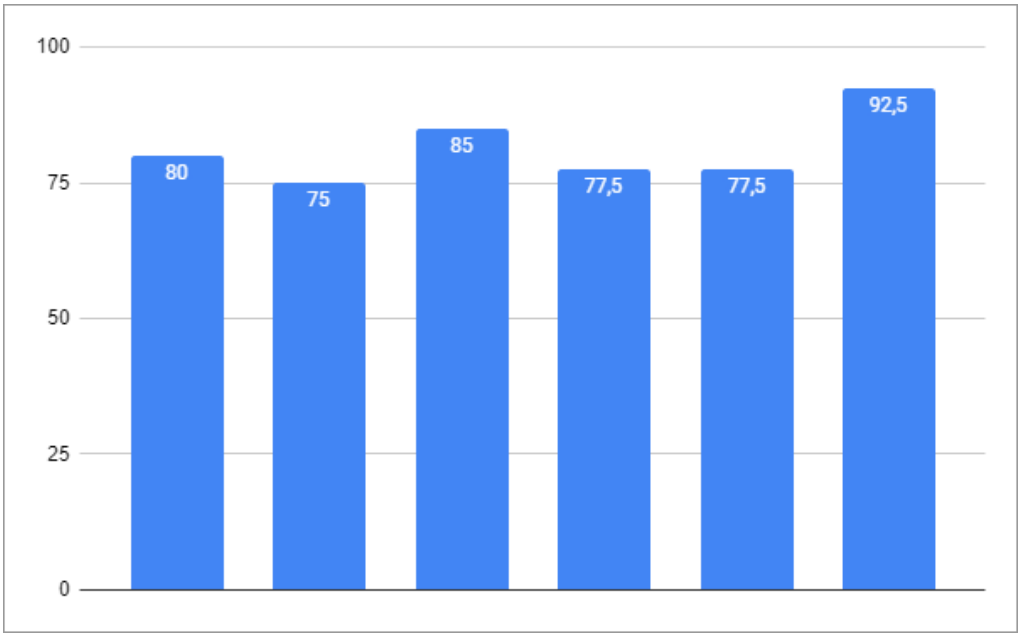
Úlohou používateľa je pohybovať sa v prostredí a nájsť 6 objektov rozmiestnených tak, aby sa nachádzali v blízkosti pozícií, kde pristáva dron. Takto vieme overiť, či dron dodržuje bezpečný odstup od používateľov počas vykonávania svojej činnosti.

Na zhodnotenie kvality nami navrhnutého a implementovaného virtuálneho prostredia sme sa rozhodli použiť metodológiu hodnotenia použiteľnosti systému (SUS). Tá je vhodná z niekoľkých dôvodov, najmä jej všeobecnosť a subjektívnosť, ktorá je pre relevantné zhodnotenie používateľského zážitku počas používania systému veľmi dôležitá. Pozostáva z desiatich otázok týkajúcich sa interakcie používateľa so systémom a jeho osobného zážitku. Každá z nich ponúka päť možných odpovedí očíslovaných 1 až 5, od rozhodne nesúhlasím - 1, až po určite súhlasím - 5. V prípade nepárnych otázok sa od čísla odpovede odčíta 1, pre párne otázky sa od čísla 5 odčíta hodnota odpovede. Výsledné čísla spočítame a vynásobíme číslom $2,5$. Týmto získame celkové SUS skóre. Priemerná hodnota je 68. Aby bol systém hodnotený ako výborný, je potrebné dosiahnuť hodnotu vyššiu, ako $80,3$. Naopak, nedostatočné systémy majú skóre nižšie ako 51.

Ďalšou výhodou metódy SUS je, že poskytuje relevantné výsledky aj pri malom počte vzoriek. To je pre nás vhodné, vzhľadom na aktuálnu epidemiologickú situáciu, keďže nie je možné ponúknuť systém na testovanie širokému okruhu ľudí.

Celkovo sa do testovania zapojilo 6 ľudí, z ktorých väčšina boli študenti vo veku od 21 do 25 rokov. Jeden z respondentov bol vo veku 50 rokov. Testovanie prebiehalo tak, že si používateľ zobrazil hlavnú stránku, ktorá obsahovala inštrukcie a informácie potrebné pre pochopenie úlohy. Navyše sa v nej nachádzali obrázky opisujúce vzhľad drona vzhľadom na jeho aktuálny stav a vzhľad objektov, ktoré má používateľ nájsť. Po prečítaní inštrukcií sa pripojil do prostredia, v ktorom sa už dron nachádzal. Nasledovalo hľadanie a zbieranie objektov. Ich aktuálny počet sa zobrazuje v hornej časti pohľadu. Po tom, čo boli všetky objekty nájdené, používateľ opustil scénu a vyplnil dotazník.

Výsledkom testovania je celkové priemerné SUS skóre implementovaného systému, ktoré dosiahlo hodnotu $81,25$, čo znamená, že systém splnil ciele použiteľnosti z hľadiska jednoduchosti, konzistentnosti a zrozumiteľnosti. Okrem klasických otázok sme však zisťovali informácie o tom, či používatelia v scéne videli drona a či s ním nejakým spôsobom interagovali. Až 5 používateľov ho videlo a



Obr. 6.2: Výsledné hodnoty SUS skóre jednotlivých používateľov

dron na nich určitým spôsobom reagoval. Jednému používateľovi sa však dron nezobrazil kvôli chybe prehliadača.

7 Záver

Účelom tejto diplomovej práce bolo navrhnúť a vytvoriť zdieľané prostredie pre virtuálnu realitu, v ktorom je možné testovať funkčnosť a použiteľnosť prototypov riadiacich systémov pre drony.

Počas analýzy sme zistili, že virtuálne prostredia sú vytvárané za pomoci jedného z dvoch hlavných softvérových rámcov - *A-Frame* a *Unity*. Rámec *Unity* používajú systémy, ktoré vyžadujú detailné grafické spracovanie a realistickú simuláciu prostredia, napríklad na trénovanie doktorov pri rôznych lekárskych zákrokoch, alebo na simuláciu správania sa ľudí počas nešťastia. Naproti tomu *A-Frame* našiel využitie skôr v oblastiach, ktoré kládli dôraz nie na vizuálne spracovanie, presnosť a kvalitu simulácie, ale na širokú podporu zariadení a jednoduchosť práce s rámcom. Preto sa rámec objavoval skôr v oblasti vzdelávania žiakov a študentov, alebo pri tvorbe jednoduchých virtuálnych svetov pomocou virtuálnej reality.

V našom prípade sme pracovali s platformou *LIRKIS G-CVE*, postavenou na *A-Frame*, rozšírenou o rozhranie pre mobilné zariadenia a komponent pre manažment používateľov. Na základe výhod *A-Frame* a podpore fyzikálnej simulácie prostredníctvom rozšírenia `aframe-physics-system` sme preukázali, že je táto platforma pre naše účely vhodná.

Následne sme analyzovali existujúce systémy, ktoré simulujú pohyb drona vo virtuálnom prostredí. Väčšina systémov vychádza z matematických modelov pre kinematiku a dynamiku, ktoré sú náročné na výpočtový výkon. Našli sme ale aj systémy, ktoré na to používajú vstavaný fyzikálny rámec. Vzhľadom na výkonnosť webového prostredia sme sa rozhodli použiť rovnaký prístup. Taktiež sme zistili, že riadenie drona sa skladá z niekoľkých častí, a to pohonných jednotiek, stabilizácie pomocou PID regulátora, a riadiaceho modulu.

Pred návrhom a implementáciou sme experimentálne overili, že pomocou nástroja *Jsweet* je možné preložiť kód z jazyka *Java* do *Javascript* a integrovať tak už vytvorený model riadiaceho systému autonómneho stroja do nášho prostredia. Správnosť procesu sme overili porovnávaním výstupov kódu v oboch jazykoch a

vytvorením prostredia s robotickým vysávačom, ktorý úspešne vykonával svoju činnosť s ohľadom na bezpečnosť používateľov v jeho okolí.

Na základe analýzy sme vytvorili model drona. Pomocou fyzikálneho rámca sme simulovali pôsobenie síl, ktorých veľkosť sme zistili využitím vzťahov na výpočet vztlakovej sily a krútiaceho momentu. Potom sme implementovali PID regulátor na stabilizáciu a riadenie otáčok motorov podľa príkazov z riadiaceho systému. Nakoniec sme vytvorili riadiaci systém, ktorý zabezpečoval autonómny presun medzi pozíciami, pristávanie a štartovanie, s ohľadom na bezpečnosť používateľov. Všetko sme umiestnili do hierarchickej štruktúry a prepojili pomocou rozhraní, aby bolo možné jednotlivé časti meniť a testovať tak rôzne riadiace systémy.

Výsledkom je komponent, ktorý je možné priradiť rôznym entitám v prostredí a urobiť tak z nich fyzikálne simulovaného a autonómne riadeného drona. Jeho parametre je možné jednoducho nastaviť prostredníctvom atribútov.

Na konci sme overili použiteľnosť implementovaného systému vo virtuálnom prostredí, ktorý predstavuje reálne prostredie stavby budovy, kde dron roznáša materiál medzi stanovenými pozíciami. Realizovali sme používateľské testovanie a výsledok vyhodnotili pomocou SUS dotazníka. Používatelia boli s prostredím a jeho funkcionalitou veľmi spokojný.

Fyzikálna simulácia drona je postačujúca, avšak je možné využiť spomínané matematické modely na zlepšenie kvality simulácie. Okrem toho by bolo vhodné simulovať a riadiť sa na základe senzorov nachádzajúcich sa v reálnych strojoch, napríklad gyroskop, akcelerometer, barometer a magnetometer. Ďalším možným rozšírením systému je podpora pre formálne špecifikovaný riadiaci systém a jeho automatická integrácia do prostredia, napríklad prostredníctvom odkazu na repozitár, alebo načítaním súboru s modelom.

Nedostatkom našej fyzikálnej simulácie, konkrétne stabilizácie kvadrokoptéry, je jej závislosť na frekvencii vykresľovania snímkov. Parametre PID regulátora je nutné adekvátne nakonfigurovať. Momentálne sme ich nastavili pre prostredie pracujúce pri 60 snímkoch za sekundu. V budúcnosti by bolo vhodné implementovať niekoľko variantov pre rôzne frekvencie.

Literatúra

1. SNOWDON, David; CHURCHILL, Elizabeth; MUNRO, Alan. Collaborative Virtual Environments: Digital Spaces and Places for CSCW: An Introduction. In: 2001. ISBN 978-1-85233-244-0. Dostupné z DOI: 10.1007/978-1-4471-0685-2_1.
2. HUDÁK, Marián; KOREČKO, Štefan; SOBOTA, Branislav. LIRKIS Global Collaborative Virtual Environments: Current State and Utilization Perspective. *De Gruyter*. 2020, roč. 11, č. 1, s. 1–8.
3. MIT. *A-Frame VR env*. 2015. Dostupné tiež z: https://aframe.io/. [cit. 01.04.2021].
4. LEE, Hayden. *Networked-Aframe*. 2017. Dostupné tiež z: https://github.com/networked-aframe/networked-aframe. [cit. 01.04.2021].
5. CAPECE, N.; ERRA, U.; LOSASSO, G.; D'ANDRIA, F. Design and Implementation of a Web-Based Collaborative Authoring Tool for the Virtual Reality. In: *2019 15th International Conference on Signal-Image Technology Internet-Based Systems (SITIS)*. 2019, s. 603–610.
6. SCAVARELLI, A.; ARYA, A.; TEATHER, R. J. Circles: exploring multi-platform accessible, socially scalable VR in the classroom. In: *2019 IEEE Games, Entertainment, Media Conference (GEM)*. 2019, s. 1–4.
7. PAIVA, Paulo V. F.; MACHADO, Liliane S.; VALENÇA, Ana Maria G.; BATISTA, Thiago V.; MORAES, Ronei M. SimCEC: A Collaborative VR-Based Simulator for Surgical Teamwork Education. *Comput. Entertain.* 2018, roč. 16, č. 2. Dostupné z DOI: 10.1145/3177747.
8. RUKANGU, A.; FRANZLUEBBERS, A.; TUTTLE, A.; MATTINGLY, K.; O'NEAL, C.; ROBINSON, D.; AHN, S. J. G.; JOHNSEN, K. Virtual Family Room: Bridging the Long Distance. In: *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. 2020, s. 52–56.

9. CHHEANG, V.; SAALFELD, P.; HUBER, T.; HUETTL, F.; KNEIST, W.; PREIM, B.; HANSEN, C. Collaborative Virtual Reality for Laparoscopic Liver Surgery Training. In: *2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. 2019, s. 1–17.

10. SHARMA, S.; RAJEEV, S. P.; DEVEARUX, P. An immersive collaborative virtual environment of a university campus for performing virtual campus evacuation drills and tours for campus safety. In: *2015 International Conference on Collaboration Technologies and Systems (CTS)*. 2015, s. 84–89.

11. SHARAD, Sharma; PHILLIP, Devreaux; DAVID, Scribner; JOCK, Grynovicki; PETER, Grazaitis. Megacity: A Collaborative Virtual Reality Environment for Emergency Response, Training, and Decision Making. *IS and T International Symposium on Electronic Imaging Science and Technology*. 2017, s. 70–77. Dostupné z DOI: 10.2352/ISSN.2470-1173.2017.1.VDA-390.

12. SHARMA, S.; BODEMPUDI, S.; ARROLLA, M.; UPADHYAY, A. Collaborative Virtual Assembly Environment for Product Design. In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2019, s. 606–611.

13. PLAZA, Juan. *What is the Value of the European Drone Market?* Dostupné tiež z: https://www.commercialuavnews.com/europe/value-european-drone-market. [cit. 07.01.2021].

14. NGUYEN, V. T.; JUNG, K.; DANG, T. DroneVR: A Web Virtual Reality Simulator for Drone Operator. In: *2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. 2019, s. 257–2575. Dostupné z DOI: 10.1109/AIVR46125.2019.00060.

15. LIU, H.; BI, Z.; DAI, J.; YU, Y.; SHI, Y. UAV Simulation Flight Training System. In: *2018 International Conference on Virtual Reality and Visualization (ICVRV)*. 2018, s. 150–151. Dostupné z DOI: 10.1109/ICVRV.2018.00052.

16. ZHANG, Z.; XIONG, M.; XIONG, H. Monocular Depth Estimation for UAV Obstacle Avoidance. In: *2019 4th International Conference on Cloud Computing and Internet of Things (CCIOT)*. 2019, s. 43–47. Dostupné z DOI: 10.1109/CCIOT48581.2019.8980350.

17. SARKAR, Mrinmoy; HOMAIFAR, Abdollah; EROL, Berat A.; BEHNIAPOOR, Mohammadreza; TUNSTEL, Edward. PIE: a Tool for Data-Driven Autonomous UAV Flight Testing. *J Intell Robot Syst*. 2019, s. 421–438. Dostupné z DOI: https://doi.org/10.1007/s10846-019-01078-y.

18. (OSRF), Open Source Robotics Foundation. *Gazebo - Robot simulation made easy*. 2014. Dostupné tiež z: http://gazebosim.org/. [cit. 21.04.2021].

19. PATELLI, Andrea; MOTTOLA, Luca. Model-Based Real-Time Testing of Drone Autopilots. In: Singapore, Singapore: Association for Computing Machinery, 2016, s. 11–16. DroNet '16. ISBN 9781450344050. Dostupné z DOI: 10.1145/2935620.2935630.

20. GIBIANSKY, Andrew. *Quadcopter Dynamics and Simulation*. Dostupné tiež z: https://andrew.gibiansky.com/downloads/pdf/Quadcopter%20Dynamics,%20Simulation,%20and%20Control.pdf. [cit. 21.03.2021].

21. BEARD, Randal. Quadrotor Dynamics and Control Rev 0.1. In: Faculty Publications, Ira A. Fulton College of Engineering, Technology, Electrical a Computer Engineering, 2008. Dostupné tiež z: http://hdl.lib.byu.edu/1877/624.

22. SHAH, Shital; DEY, Debadeepta; LOVETT, Chris; KAPOOR, Ashish. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In: *Field and Service Robotics*. 2017. Dostupné z eprint: arXiv:1705.05065.

23. SONG, Yunlong; NAJI, Selim; KAUFMANN, Elia; LOQUERCIO, Antonio; SCARAMUZZA, Davide. Flightmare: A Flexible Quadrotor Simulator. 2020.

24. KIAM HEONG ANG Gregory Chong, Yun Li. PID Control System Analysis, Design, and Technology. *IEEE Transactions on Control Systems Technology*. 2007, roč. 13, s. 559–576.

25. HUANG, Peter. *Quadcopter 3D Simulator*. 2019. Dostupné tiež z: https://github.com/hbd730/quadcopter-simulation.

26. KUMAR, Vijay. *Robotics: Aerial Robotics*. 2016. Dostupné tiež z: https://www.coursera.org/learn/robotics-flight.

27. MAJUMDAR, Abhijit. *Quadcopter simulator*. 2018. Dostupné tiež z: https://github.com/abhijitmajumdar/Quadcopter_simulator.

28. LEVY, Simon D. *Multicopter Sim*. 2021. Dostupné tiež z: https://github.com/simondlevy/MulticopterSim.

29. BOUABDALLAH, Samir; MURRIERI, Pierpaolo; SIEGWART, Roland. Design and Control of an Indoor Micro Quadrotor. In: 2004, zv. 5, 4393–4398 Vol.5. ISBN 0-7803-8232-3. Dostupné z DOI: 10.1109/ROBOT.2004.1302409.

30. DIMASHKY, Mohamed Khair; AL-KASSIR, Maher; AL-AJLANI, Majd. *Quadcopter-Simulator*. 2017. Dostupné tiež z: `https://github.com/dimashky/Quadcopter-Simulator`.

31. LUUKKONEN, Teppo. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*. 2011, roč. 22, s. 22.

32. KOREČKO, Štefan; SOBOTA, Branislav; JACHO, Ladislav. Can Game Engines Improve Development of Formally Verified Software? 2019.

33. IVAN, Michal. *Kolaborácia vo virtuálnej realite: komunikacná a riadiaca cast*. 2020. Dipl. pr. Technická univerzita v Košiciach.

34. TRIMBLE. *3D Warehouse*. 2021. Dostupné tiež z: `https://3dwarehouse.sketchup.com/`.

Zoznam príloh

Príloha A Používateľská príručka

Príloha B Systémová príručka

Príloha C CD médium – záverečná práca v elektronickej podobe

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Používateľská príručka

Príloha A

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Štefan Korečko PhD.
Konzultant:	

Košice 2021 **Bc. Dávid Gula**

Obsah

1 Funkcia programu **1**

2 Inštalácia programu **2**

2.1 Platforma Glitch 2

2.2 Lokálna inštalácia 2

3 Použitie programu **4**

4 Obmedzenia programu **5**

Literatúra **6**

Zoznam obrázkov

2.1 Spustenie terminálu v *Glitch* 3

1 Funkcia programu

Systém poskytuje zdieľané virtuálne prostredie podporujúce pripojenie a interakciu medzi viacerými používateľmi naraz. Obsahuje simuláciu dvoch autonómne ovládaných zariadení - robotického vysávača a lietajúceho drona. Obe zariadenia reagujú na prítomnosť používateľov v prostredí a upravujú svoju činnosť tak, aby ich neohrozili. Scéna prostredia predstavuje miesto prebiehajúcej stavby a obsahuje rôzne predmety, vozidlá, stavebný materiál, kontainery s materiálom a stavebné žeriavy. Úlohou systému je poskytnúť možnosť testovať a overovať prototypy autonómne ovládaných strojov vo virtuálnom priestore, s možnosťou interakcie s reálnymi používateľmi.

2 Inštalácia programu

V tomto prípade máme na výber z dvoch možností, podľa toho, či chceme mať systém spustený lokálne na našom počítači, alebo prostredníctvom bezplatnej platformy *Glitch* [1]. V oboch prípadoch systém spúšťame v prehliadači, je teda nutné mať nainštalovaný a aktualizovaný jeden z moderných podporovaných prehliadačov, napríklad *Google Chrome*, *Mozilla Firefox*, *Opera* alebo *Microsoft Edge*.

2.1 Platforma Glitch

V prípade použitia platformy *Glitch* je postup priamočiary a jednoduchý. Je potrebné najprv na platforme vytvoriť nový projekt, do ktorého budeme nahrávať súbory. Všetky súbory v projekte je potrebné pred nahrávaním odstrániť. Potom prejdeme do priečinka `assets` a nahráme zdrojové súbory vo formáte `zip`. Po nahratí je potrebné kliknúť na súbor a skopírovať webovú adresu. Následne si otvoríme terminál, ku ktorému sa dostaneme cez tlačidlo `Tools`.

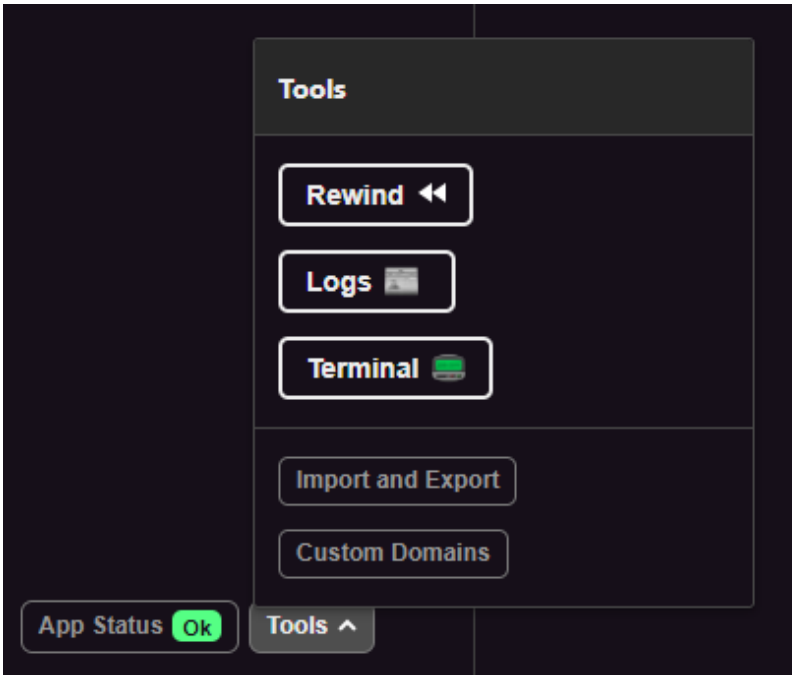
Do neho zadáme nasledujúce príkazy, ktorými rozbalíme zdrojové súbory do projektu:

1. `$ wget -O file.zip https://url-adresa-zip-suboru`
2. `$ unzip file.zip -d .`
3. `$ rm file.zip`
4. `$ refresh`

Potom si už môžeme projekt spustiť pomocou tlačidla `Show` v hornej časti obrazovky.

2.2 Lokálna inštalácia

Pre spustenie lokálnej verzie systému je nutné mať nainštalovaný *Node.js* [2]. Postup inštalácie je opísaný v `https://nodejs.dev/learn/how-to-install-nodejs`.



Obr. 2.1: Spustenie terminálu v *Glitch*

Po úspešnej inštalácii stačí prejsť do priečinka s projektom a v príkazovom riadku zadať `node server.js`. Systém je potom dostupný na adrese `http://localhost:8081` v prehliadači.

3 Použitie programu

Po otvorení spomínanej adresy v prehliadači sa nám zobrazí stránka s popisom scenára pre používateľov. Po kliknutí na tlačidlo v spodnej časti obrazovky sa do systému prihlásime ako bežný používateľ. Zobrazí sa nám textové pole, do ktorého zadáme meno pre nášho avatara. Potom sa už môžeme po prostredí pohybovať tak, ako je vysvetlené na úvodnej stránke.

Do systému sa okrem bežného používateľa môžeme prihlásiť aj ako títo používatelia:

- Administrátor - dostupný na adrese `http://localhost:8081/admin.html`, prihlásenie vyžaduje zadanie hesla `welcome`
- Robotický vysávač cBot - dostupný na adrese `http://localhost:8081/cbot.html`, prihlásenie vyžaduje zadanie hesla `hello`
- Lietajúci dron - dostupný na adrese `http://localhost:8081/drone.html`

V rozhraní administrátora môžeme sledovať pripojených používateľov, ich atribúty a prípadne ich zo scény vyhodiť. V prípade prihlásenia ako cBot sa nám zobrazuje jeho pohľad z prvej osoby. Nevieme sa pohybovať, iba ovládať smer, kam sa pozeráme. V prípade drona sa vieme pohybovať aj ovládať pohľad rovnako, ako pri bežnom používateľovi.

4 Obmedzenia programu

Keďže sa jedná o zdieľané prostredie, všetci používatelia sa navzájom vidia, iba ak sú pripojení. Napríklad ak chceme mať v prostredí spusteného drona a zároveň sa v ňom pohybovať ako používateľ, je potrebné byť naraz prihlásený aj ako dron, aj ako používateľ.

Na to, aby dochádzalo k aktualizácii pohybu pri dronovi a cBotovi, je nutné mať kartu, v ktorej beží daná inštancia, aktívnu a v popredí, aby ju prehliadač aktualizoval.

Taktiež je potrebné mať v prípade drona systém, ktorý dokáže udržiavať jeho simuláciu pri 60 snímkoch za sekundu. V opačnom prípade je dron nestabilný a nelieta správne.

Literatúra

1. FCS, Fog Creek Software. *Glitch*. 2021. Dostupné tiež z: `https://glitch.com/`.
2. JOYENT, OpenJs Foundation. *Node.js*. 2021. Dostupné tiež z: `https://nodejs.org/en/`.

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Systémová príručka

Príloha B

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Štefan Korečko PhD.
Konzultant:	

Košice 2021 **Bc. Dávid Gula**

Obsah

1 Funkcia programu **1**

2 Popis programu **2**

2.1 Zdrojový kód cBota 3

2.1.1 Komponent cbot-cleaning 4

2.1.2 Komponent cbot-dirty-position 5

2.2 Zdrojový kód drona 5

2.2.1 PID regulátory 5

2.2.2 Ovládač pre drona 7

2.2.3 Komponent drone 8

2.2.4 Komponent drone-checkpoint 11

3 Zhodnotenie riešenia **12**

Literatúra **13**

Zoznam obrázkov

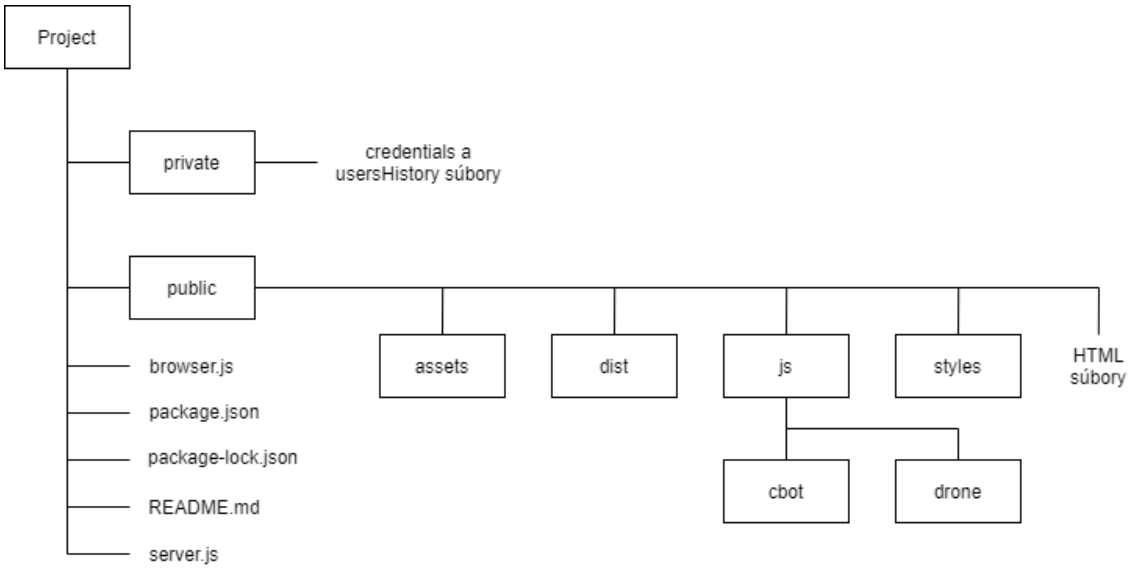
2.1 Štruktúra projektu 2

1 Funkcia programu

Systém poskytuje zdieľané virtuálne prostredie podporujúce pripojenie a interakciu medzi viacerými používateľmi naraz. Obsahuje simuláciu dvoch autonómne ovládaných zariadení - robotického vysávača a lietajúceho drona. Obe zariadenia reagujú na prítomnosť používateľov v prostredí a upravujú svoju činnosť tak, aby ich neohrozili. Scéna prostredia predstavuje miesto prebiehajúcej stavby a obsahuje rôzne predmety, vozidlá, stavebný materiál, kontainery s materiálom a stavebné žeriavy. Úlohou systému je poskytnúť možnosť testovať a overovať prototypy autonómne ovládaných strojov vo virtuálnom priestore, s možnosťou interakcie s reálnymi používateľmi.

2 Popis programu

Program bol implementovaný v jazyku *HTML* a *Javascript*. Pre vývoj na lokálnom počítači je potrebné si nainštalovať *Node.js*. Je ale možné vyvíjať systém aj pomocou online platformy *Glitch*, kde stačí do nového projektu nahrať a rozbaliť zdrojové súbory. Štruktúra programu je na obrázku 2.1.



Obr. 2.1: Štruktúra projektu

V priečinku `private` sa nachádzajú *JSON* súbory s prihlasovacími heslami pre jednotlivých používateľov a taktiež súbor s históriou používateľov.

V priečinku `public` sa nachádzajú zdrojové súbory *HTML* stránok, *CSS* štýly a *Javascript* kód systému. Taktiež sú tam uložené modely a textúry použité v programe.

Význam jednotlivých podpriečinkov je nasledovný:

- `assets` - použité modely a textúry
- `dist` - použité knižnice v *Javascript*
- `js` - zdrojový kód systému - `cbot` obsahuje kód pre ovládač a pomocné

triedy pre robotický vysávač cBot, `drone` obsahuje kód pre ovládač, stabilizátor a komponent drona

- `styles` - *CSS* štýly pre stránky

Okrem toho sa v priečinku `public` nachádzajú všetky *HTML* súbory s nasledovným významom:

- `admin.html` - stránka pre administrátorské rozhranie
- `cbot.html` - stránka pre rozhranie robotického vysávača cBot
- `drone.html` - stránka pre rozhranie drona
- `hololens.html` - stránka pre rozhranie používateľa s prilbou pre rozšírenú realitu *Hololens*
- `index.html` - hlavná stránka systému
- `user.html` - stránka pre rozhranie používateľa pripojeného cez počítač
- `user-phone.html` - stránka pre rozhranie používateľa pripojeného cez mobilné zariadenie

Funkcionalita implementovaná v tejto diplomovej práci sa nachádza v podpriečinkoch `cbot` a `drone`. Zvyšné súbory vznikli počas diplomovej práce Michala Ivana [1].

2.1 Zdrojový kód cBota

Priečinok `cbot` obsahuje nasledovné súbory:

- `cBotComponents.js` - komponenty *A-Frame* pre cBota a jeho pozície
- `cBotLogin.js` - funkcionalita pre prihlásenie cBota do prostredia, kód sa len mierne odlišuje od implementácie iných používateľov, ktoré riešil Michal vo svojej práci
- `controller.js` - kód ovládača preložený z jazyka *Java* pomocou nástroja *Jsweet*
- `controllerCore.js` - jadro ovládača taktiež preloženého z *Javy*
- `proximSensors.js` - kód senzorov cBota preložený z *Javy*

V tejto príručke bude opísaný len súbor `cBotComponents.js`, ktorý sme implementovali počas tejto diplomovej práce.

2.1.1 Komponent cbot-cleaning

Tento komponent obsahuje implementáciu riadiaceho softvéru robotického vysávača. Obsahuje nasledovné konfiguračné parametre definované v `schema`:

- `command` - príkaz pre ovládač
- `maxMsrblDst` - maximálny dosah senzorov
- `safeDstCl` - minimálna bezpečná vzdialenosť od používateľa, aby bolo možné spustiť čistenie
- `safeDstMov` - minimálna bezpečná vzdialenosť od používateľa pre pohyb
- `cleanRange` - vzdialenosť od pozície pre zahájenie čistenia

Metódy definované v komponente:

- **`function`** `init()` - inicializácia premenných, ovládača a pozícií pre čistenie
- **`function`** `tick()` - získavanie pozícií používateľov, kontrola dosiahnutia pozície a aktualizácia ovládača a na základe jeho výstupov aj pozície a rotácie cBota
- **`function`** `cleanPosition(entity)` - aktualizácia súčasnej pozície (parameter `entity`), ak práve prebieha čistenie a odstránenie pozície, keď je čistenie hotové - prebieha to znižovaním hodnoty `dirtiness` danej entity o 1 pri každom volaní
- **`function`** `getUsersDistances(botPos)` - získanie aktuálnych pripojených klientov z globálnej premennej `clientPositions` a prepočítanie ich pozície vzhľadom na pozíciu cBota (parameter `botPos`)
- **`function`** `updateColor()` - aktualizácia farby objektu hlavy cBota podľa jeho aktuálneho stavu
- **`function`** `normalizeAngleDeg(angle)` - prepočítanie uhlu v stupňoch na rozsah $0°$ až $360°$
- **`function`** `normalizeAngleRad(angle)` - prepočítanie uhlu v radiánoch na rozsah 0 až 2π
- **`function`** `isVector3Zero(vector)` - vráti `true`, ak je 3D vektor nulový, inak `false`

- `function mapValueToRange(value, fromMin, fromMax, toMin, toMax)` - vráti hodnotu z rozsahu od `fromMin` do `fromMax` na hodnotu z rozsahu od `toMin` do `toMax`

2.1.2 Komponent cbot-dirty-position

Tento komponent slúži na definovanie pozície na čistenie pre cBota. Obsahuje nasledovné konfiguračné parametre definované v `schema`:

- `dirtiness` - celé číslo reprezentujúce špinavosť danej pozície
- `isClean` - značka, či je pozícia čistá

Metódy definované v komponente:

- `function init()` - vytvorenie materiálu entity
- `function update()` - kontrola, či je hodnota špinavosti vyššia, ako 0 - ak nie, entita nastaví značku `isClean` na `true`

2.2 Zdrojový kód drona

Priečinok `drone` obsahuje nasledovné súbory:

- `droneComponents.js` - komponenty *A-Frame* pre drona a jeho pozície
- `droneController.js` - kód ovládača pre automatické ovládanie drona
- `droneLogin.js` - funkcionalita pre prihlásenie drona do prostredia, kód sa len mierne odlišuje od implementácie iných používateľov, ktoré riešil Michal vo svojej práci
- `pidController.js` - kód PID regulátorov pre stabilizáciu drona

V tejto príručke nebudeme opisovať súbor `droneLogin.js`, keďže jeho implementácia bola prevzatá z kódu, ktorý vytvoril Michal počas svojej diplomovej práce.

2.2.1 PID regulátory

V súbore `pidController.js` sú definované dve triedy:

- `PIDController` - trieda pre základný PID regulátor

- `PIDPositionController` - trieda pokročilého PID regulátora

Okrem toho je tam definovaná enumerácia `PIDController.MODE = {MANUAL: 0, AUTO: 1}` pre dva módy regulátora. Štruktúra triedy `PIDController`:

- `let PIDController = function (body, maxRpm, maxTilt, sampleTime)` - konštruktor triedy
- `PIDController.prototype.getTime = function ()` - pomocná funkcia na získanie aktuálneho systémového času
- `PIDController.prototype.wrapAngle = function (angle, min, max)` - pomocná funkcia pre prevod uhlu do rozsahu od `min` po `max`
- `PIDController.prototype.getMotorValues = function ()` - vracia pole s hodnotami otáčok pre jednotlivé motory
- `PIDController.prototype.resetIntegrals = function ()` - vynuluje interné počítadla
- `PIDController.prototype.getMode = function ()` - vráti aktuálne nastavený mód regulátora
- `PIDController.prototype.setMode = function (m)` - nastaví mód regulátoru
- `PIDController.prototype.updateHoverThrottle = function ()` - nastaví veľkosť ťahu pre udržanie sa v rovnakej výške
- `PIDController.prototype.update = function (desiredHeight, pitchCommand, rollCommand, yawCommand)` - výpočet otáčok pre motory podľa zadaných vstupov - ak je nastavený čas `sampleTime`, vykonáva sa len ak ubehol dlhší časový úsek od posledného volania, ako nastavený čas

Trieda `PIDPositionController` je potomkom triedy `PIDController` a dedí väčšinu jej metód. Opísané budú len odlišné funkcie:

- `let PIDPositionController = function (body, maxRpm, maxTilt, sampleTime)` - konštruktor triedy

- `PIDPositionController.prototype.update = function (target, yawCommand)` - výpočet otáčok pre motory podľa zadaných vstupov - ak je nastavený čas `sampleTime`, vykonáva sa len ak ubehol dlhší časový úsek od posledného volania, ako nastavený čas

2.2.2 Ovládač pre drona

Funkcionalita je implementovaná v triede `DroneController`, ktorá sa nachádza v súbore `droneController.js`. Okrem toho súbor obsahuje aj enumeráciu

```
DroneController.COMMAND =
{LAND: 0, REACH_HOVER: 1, REACH_LAND: 2, STAY_HOVER: 3}
```

pre jednotlivé príkazy ovládača.
Štruktúra triedy `DroneController` je nasledovná:

- `let DroneController = function (maxMsrblDst, safeDstMov, safeAltitude)` - konštruktor triedy
- `DroneController.prototype.updateAndEvaluate = function (command, targetPos, currentPos, obstaclesPos)` - aktualizácia stavov drona a implementácia automatického riadenia podľa zadaného príkazu
- `DroneController.prototype.readSensors = function (currentPos, obstaclesPos)` - kontrola bezpečných vzdialeností používateľov od drona
- `DroneController.prototype.landAndTurnOff = function (currentPos)` - funkcia použitá pre príkaz `LAND` - bezpečné pristátie s dronom tak, aby nedošlo k ohrozeniu používateľov
- `DroneController.prototype.goToPosAndHover = function (currentPos, targetPos)` - funkcia použitá pre príkaz `REACH_HOVER` - presun drona k pozícii a zotrvanie v jej výške
- `DroneController.prototype.goToPosAndLand = function (currentPos, targetPos)` - funkcia použitá pre príkaz `REACH_LAND` - presun drona k pozícii a bezpečné pristátie s ohľadom na používateľov
- `DroneController.prototype.stayAndHover = function (currentPos)` - funkcia použitá pre príkaz `STAY_HOVER` - zastavenie a zotrvanie na aktuálnej pozícii v bezpečnej výške

- `DroneController.prototype.setPosTo = function (target, pos)` - nastavenie výstupnej pozície, ktorá je použitá pre PID regulátor
- `DroneController.prototype.distanceBetween = function (pos1, pos2)` - vracia vzdialenosť medzi dvoma pozíciami v 3D priestore

2.2.3 Komponent drone

Tento komponent slúži na implementáciu fyzikálnej simulácie, autonómneho a manuálneho riadenia a stabilizácie drona. Nachádza sa v súbore `droneComponents.js`.

Globálne konštanty definované v súbore:

- `const ro = 1.225` - hustota vzduchu
- `const g = -9.80665` - gravitácia
- `const sqrt2 = Math.sqrt(2)` - pomocná premenná pre druhú odmocninu čísla 2

Globálne metódy definované v súbore:

- `function RPMtoRADs(rpm)` - prevod otáčok za minútu na uhlovú rýchlosť
- `function inToCm(inches)` - prevod palcov na centimetre
- `function cmToInches(cm)` - prevod centimetrov na palce
- `function wrapAngle(angle, min, max)` - prevod uhla na rozsah od `min` po `max`
- `function isVector3Zero(vector)` - vráti `true`, ak je 3D vektor nulový, inak `false`
- `function mapValueToRange(value, fromMin, fromMax, toMin, toMax)` - vráti hodnotu z rozsahu od `fromMin` do `fromMax` na hodnotu z rozsahu od `toMin` do `toMax`

Komponent používa nasledujúcu štruktúru na zjednotenie všetkých rôznych metód ovládania:

`let Command = function (height, pitch, roll, yaw)`

Komponent obsahuje nasledovné konfiguračné parametre definované v `schema`:

- `template` - id šablóny pre *Networked-Aframe* [2] a vytvorenie zdieľanej entity

- `attachTemplateToLocal` - hodnota `true` znamená, že model drona bude viditeľný aj pre vlastníka, inak bude viditeľný len pre ostatných pripojených používateľov
- `controlMode` - nastavuje mód ovládania drona - jeden z `keyboard` (manuálne ovládanie), `auto` (testovacie ovládanie pomocou ovládača pre cBota), `p2pauto` (autonómne ovládanie pomocou ovládača)
- `mass` - váha drona v kg
- `motorMaxRPM` - maximálne otáčky motorov
- `motorOffset` - 3D vektor pozícií motorov vzhľadom na model
- `maxTiltDeg` - maximálny náklon drona počas letu v stupňoch
- `thrustConstant` - konštanta ťahu motorov
- `dragConstant` - konštanta krútiaceho momentu motorov
- `propellerDiameter` - priemer lopatiek drona v cm
- `propellerPitch` - sklon lopatiek drona v cm
- `maxMsrblDst` - maximálny dosah senzorov
- `safeDstMov` - minimálna bezpečná vzdialenosť pre pohyb
- `safeAltitude` - bezpečná letová hladina

Metódy definované v komponente:

- `function init()` - inicializácia premenných, pozícií pre drona, metód pre ovládanie a spracovanie vstupov z klávesnice, sieťového a fyzikálneho komponentu
- `function update(oldData)` - aktualizácia veľkosti ťahu potrebného na udržanie rovnakej výšky, ak dôjde k zmene atribútov modelu
- `function initControl()` - inicializácia jednotlivých ovládačov a nastavenie udalostí pre vstupy z klávesnice
- `function onBodyInit()` - funkcia na spracovanie udalosti, keď je dokončená inicializácia fyzikálneho komponentu, incializuje PID regulátory
- `function onKeyDown()` - spracovanie stlačenia klávesy

- `function onKeyUp()` - spracovanie uvoľnenia klávesy
- `function updateKeys()` - aktualizácia príkazu pre pohyb podľa stlačených kláves
- `function setupKeyboardControls()` - inicializácia metód pre udalosť stlačenia klávesy
- `function getUsersDistances(currentPos)` - výpočet vzdialeností jednotlivých používateľov od drona (parameter `currentPos`)
- `function preStep(body)` - aktualizácia pozície a rotácie drona podľa fyzikálneho modelu; výpočet síl, ktorými motory pôsobia na telo drona a aplikovanie týchto síl; aktualizácia príslušného ovládača a PID regulátora
- `function receiveCommand()` - naplnenie štruktúry `Command` hodnotami z aktuálneho ovládača podľa zvoleného módu
- `function getControllerCommand()` - získanie hodnôt z ovládača pre cBota, kontrola dosiahnutia pozície
- `function getP2pControllerCommand()` - získanie hodnôt z ovládača pre drona, kontrola dosiahnutia pozície, aktualizácia farby podľa aktuálneho stavu
- `function reachedCheckpoint()` - volá sa pri dosiahnutí pozície - zmena nasledujúcej pozície a aktualizácia indexu pozície
- `function updateColor()` - nastavenie farby telu drona podľa aktuálneho stavu
- `function getKeyboardCommand()` - získanie hodnôt pre štruktúru `Command` podľa stlačených klávesov
- `function getGamepadCommand()` - momentálne neimplementovaná metóda na získavanie hodnôt pre štruktúru `Command` podľa vstupu z joystickového ovládača
- `function getMotorThrust(rpm, inflowVelocity)` - výpočet vztlakovej sily produkovanej motormi s danými otáčkami a vstupnou rýchlosťou vzduchu
- `function getMotorTorqueFromThrust(thrust)` - výpočet krútiaceho momentu motora podľa pomeru medzi konštantami pre ťah a krútiaci moment

- `function getMotorRPMFromThrust(thrust)` - výpočet otáčok motorov podľa generovaného ťahu
- `function getHoverThrust()` - výpočet veľkosti ťahu motorov potrebného pre udržanie drona v rovnakej výške
- `function getDeltaTime()` - výpočet času od posledného vykreslenia snímky
- `function remove()` - zrušenie počúvania na udalosti pre klávesnicu a inicializáciu fyzikálneho komponentu

2.2.4 Komponent drone-checkpoint

Tento komponent slúži na definovanie bodu záujmu pre drona, ktorý týmito bodmi bude prechádzať. Existujú dva druhy pozície, ktoré definuje enumerácia:

```
const DRONE_CHECKPOINT_TYPE = {REACH: "reach", LAND: "land"}
```

Komponent obsahuje nasledovné konfiguračné parametre definované v `schema`:

- `type` - jeden z druhov podĺa enumerácie `DRONE_CHECKPOINT_TYPE` - `REACH` značí, že dron po dosiahnutí pozície má ostať a vznášať sa na jej mieste; `LAND` značí, že dron po dosiahnutí pozície má pristáť
- `waitTime` - čas v ms, počas ktorého dron vyčká na danej pozícii predtým, než bude pokračovať na ďalšiu

Metódy definované v komponente:

- `function init()` - inicializácia 3D modelu, farby materiálu a premenných

3 Zhodnotenie riešenia

Vytvorené komponenty je možné integrovať do akéhokoľvek virtuálneho prostredia v `A-Frame` [3] a vytvoriť tak fyzikálne simulovaného a autonómne riadeného drona. Podmienkou je, aby bola v prostredí zapnutá fyzikálna simulácia a aspoň jedna definovaná pozícia pre drona. Taktiež sú tieto komponenty závislé od iných, ktoré vznikli počas diplomovej práce Michala Ivana [1].

Počas implementácie sa nám z časových dôvodov nepodarilo implementovať podporu pre spracovanie vstupov z joystickového ovládača. Jedným z obmedzení tohto systému je závislosť PID regulátora, a tým pádom aj celej stabilizácie drona od frekvencie snímkov za sekundu. Momentálne sme to optimalizovali pre 60 snímkov za sekundu. Avšak v prípade, ak bude toto číslo väčšie, či menšie, dron bude nestabilný a bude dochádzať k pomalým reakciám a nepresnej navigácii.

Literatúra

1. IVAN, Michal. *Kolaborácia vo virtuálnej realite: komunikacná a riadiaca cast*. 2020. Dipl. pr. Technická univerzita v Košiciach.
2. LEE, Hayden. *Networked-Aframe*. 2017. Dostupné tiež z: `https://github.com/networked-aframe/networked-aframe`. [cit. 23.04.2021].
3. MIT. *A-Frame VR env*. 2015. Dostupné tiež z: `https://aframe.io/`. [cit. 23.04.2021].