

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

**Využitie prostriedkov internetu vozidiel v
navigácii mobilných robotov**

Diplomová práca

2023

Bohdan Tanasov

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

**Využitie prostriedkov internetu vozidiel v
navigácii mobilných robotov**

Diplomová práca

Študijný program: Inteligentné systémy
Študijný odbor: Informatika
Školiace pracovisko: Katedra umelej inteligencie (KUI)
Školiteľ: doc. Dr. Ing. Ján Vaščák
Konzultant: ing. Dušan Herich

Košice 2023

Bohdan Tanasov

Abstrakt v SJ

Cieľom tejto práce bolo vyvinúť automatizovaný systém riadenia pre bezpilotné lietadlo, ktorý využíva zabudovanú kameru na navigáciu a zber údajov. Systém bol implementovaný pomocou programovacieho jazyka Python spolu s balíkom OpenCV pre algoritmy počítačového videnia a značkami ArUco pre merania, NodeJS a React. Pomocou systému môžeme manipulovať s viacerými dronmi a so samostatným dronom z webovej stránky a môžeme vidieť údaje dronov a súradnice dronov v priestore pomocou značiek aruco. Na záver možno konštatovať, že systém úspešne dosiahol svoj cieľ, ktorým je adaptácia autonómneho navigačného systému pre bezpilotné lietadlá v interiéri. Následne spracované súbory údajov boli dostatočne dobre filtrované, aby sa mohli považovať za adekvátne merania. Program by sa mohol v budúcnosti vylepšiť tak, aby podporoval viac príkazov pre skupiny dronov, napr. postaviť kolo alebo iné postavy. Takýto systém sa dá použiť aj pre priemyselné bezpilotné lietadlá, kde sú polohy AR markerov vopred známe.

Klúčové slová

UAV, autonómna navigácia, AR marker, určovanie polohy, merania

Abstrakt v AJ

The objective of this work was to develop an automated control system for an unmanned aircraft that uses an embedded camera for navigation and data collection. The system was implemented using the Python programming language along with the OpenCV package for computer vision algorithms and the ArUco tags for measurements, NodeJS and React. Using the system, we can manipulate multiple drones and a single drone from a web page and can see the drone data and drone coordinates in space using aruco tags. In conclusion, the system has successfully achieved its

goal of adapting an autonomous navigation system for drones indoors. The subsequently processed datasets were sufficiently well filtered to be considered as adequate measurements. The program could be improved in the future to support more commands for groups of drones, e.g., build a bike or other figures. Such a system could also be used for industrial drones where the positions of the AR markers are known in advance.

Kľúčové slová v AJ

UAV, autonomous navigation, AR marker, positioning, measurements

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
Katedra kybernetiky a umelej inteligencie

Z A D A N I E
D I P L O M O V E J P R Á C E

Študijný odbor: **Informatika**

Študijný program: **Inteligentné systémy**

Názov práce:

Využitie prostriedkov internetu vozidiel v navigácii mobilných robotov

Use of the Internet of Vehicles Means in the Navigation of Mobile Robots

Študent: **Bc. Bohdan Tanasov**

Školiteľ: **doc. Dr. Ing. Ján Vaščák**

Školiace pracovisko: **Katedra kybernetiky a umelej inteligencie**

Konzultant práce: **Ing. Dušan Herich**

Pracovisko konzultanta: **Katedra kybernetiky a umelej inteligencie**

Pokyny na vypracovanie diplomovej práce:

1. Analýza existujúcich prístupov a technológií pre využitie infraštruktúry internetu vozidiel v navigácii mobilných robotov.
2. Návrh architektúry systému pre využitie prostriedkov internetu vozidiel v navigácii mobilných robotov.
3. Implementácia navrhnutého systému a jeho testovanie v rôznych prostrediach a scenároch.
4. Overenie funkčnosti a spoľahlivosti systému na lietajúcich vozidlách.
5. Porovnanie výkonu a efektivity navrhnutého systému s existujúcimi prístupmi a technológiami.
6. Vypracovať dokumentáciu podľa pokynov vedúceho záverečnej práce.

Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 21.04.2023

Dátum zadania diplomovej práce: 31.10.2022

.....
prof. Ing. Liberios Vokorokos, PhD.

dekan fakulty



Čestné vyhlásenie

Vyhlasujem, že som bakalársku prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice 21. 4. 2023

.....

Vlastnoručný podpis

Poděkovanie

Najskôr by som sa chcel poděkovat vedúcemu bakalárskej práce doc. Dr. Ing. Jánovi Vaščakovi. Príležitosť komunikovať s docentom Vaščakom bola vždy otvorená, kedykoľvek som narazil na problémové miesto alebo mal otázku o svojom výskume alebo písaní. Dôsledne pripúštal, aby tato práca bola mojou vlastnou, ale vždy, keď si myslel, že to potrebujem, ma nasmeroval správnym smerom.

Na záver musím svojim rodičom poděkovat za to, že mi počas rokov štúdia a pri výskume a písaní tejto práce poskytovali neutíchajúcu podporu a neustále povzbudzovanie. Bez nich by tento úspech nebol možný. Ďakujem.

Predhovor

Téma práce sa zaoberá využitím dronov Tello v skupinovej koordinovanej misii, kde bola navrhnutá a implementovaná webová aplikácia, ktorá umožňuje ovládanie viacerých dronov naraz. V práci je tiež popísaná implementácia systému, ktorý umožňuje sledovanie pozície dronov pomocou kamery a detekcie Aruco markerov. Cieľom tejto práce bolo ukázať, ako efektívne ovládať viacero dronov naraz pomocou jednoduchej webovej aplikácie a koordinovaného systému detekcie pozície dronov. V práci sa tiež zaoberáme riešeniami problémov pri použití viacerých dronov naraz a analyzujeme výsledky našich experimentov a testov.

Obsah

Úvod	1
1 Formulácia úlohy	2
2 Prehľad literatúry	4
2.1 Optický princíp senzorov	4
2.2 Vyhýbanie sa prekážkam a orientácia na základe obrazu z kamery	5
2.2.1 S vylepšenou neurónovou sietou	5
2.2.2 Princíp stereosnímania	7
2.2.3 Na základe pohyblivého obrazu kamery	9
2.2.4 Určovanie polohy pomocou značiek	10
2.3 WebSocket	13
3 Design	15
3.1 Výber dronu	15
3.2 Programovací jazyk a súbor nástrojov	16
3.3 Ako ovládať dron tello	18
3.4 Umiestnenie dronu	23
3.4.1 Používanie značkovačov aruco	24
3.4.2 Pravidlá umiestňovania značiek	26
3.5 Prvé testy polohovania pomocou značiek	27
3.5.1 Vyhodnotenie testu a očakávania od systému	28
3.6 Princíp, spresnenie a robustnosť polohovania	29
3.6.1 Perspektívna projekcia, transformácia pnp	30
3.6.2 Konvencie merania	33
3.6.3 Konvencie používané v opencv	35
3.6.4 Prevod polohy kamery do globálneho súradnicového systému .	37
3.7 Webové sokety s komunikáciou s dronom	38
3.7.1 Architektúra	39

3.7.2	Komunikačný protokol	39
3.7.3	Implementácia	41
4	Implementácia projektu	42
4.1	Prehľad štruktúry programov	42
4.2	Programy, ktoré ovládajú dron	43
4.3	Programy, ktoré spracúvajú obrázky	44
4.4	Programy na spracovanie údajov	47
4.4.1	Ukladanie údajov o značkách	47
4.5	Webová aplikácia	49
4.5.1	Frontend	49
4.5.2	Backend	51
5	Simulačné experimenty	52
5.1	Detekcia značiek	53
5.2	Výpočet polohy	55
5.3	Režim viacerých dronov	56
5.4	Kontrola skupiny dronov	57
6	Zhrnutie	59
6.1	Návrhy na ďalší vývoj	62
7	Záver	63
	Zoznam použitej literatúry	64

Zoznam obrázkov

2–1	Načítaný obraz (vľavo), skutočná mapa hĺbky na základe senzorov (vľavo uprostred) a napokon mapa hĺbky vytvorená pomocou Gaussovoho modelu (vpravo uprostred) a Laplaceovho modelu (vpravo)	6
2–2	Faktory zohľadňované pri monokulárnom vnímaní hĺbky	7
2–3	Model virtuálnej kamery. (a) Model virtuálnej kamery systému stereovízie založeného na plochej prizme. (b) Modifikovaný model dierkovej virtuálnej kamery systému stereovízie založeného na voľnej prizme.	8
2–4	Vľavo: Tradičné stereofónne nastavenie predpokladá, že aspoň dva uhly pohľadu zachytávajú scénu v rovnakom čase. Vpravo: Uvažujeme o nastavení, pri ktorom sa kamera aj objekt pohybujú.	9
2–5	Pravdepodobnosť detekcie značky v závislosti od jej oblasti na obrázku pre 3 rôzne uhly naklonenia: 0° , 45° , 70°	12
2–6	Závislosť vplyvu veľkosti oblasti značky na chybe pri zistovaní jej polohy.	12
3–1	Obrázok dronu DJI Tello zobrazujúci jeho komponenty.	15
3–2	Schéma návrhu systému: Webová aplikácia vytvorená pomocou React komunikuje s backendovým serverom Node.js, ktorý posiela príkazy a prijíma telemetrické údaje z dronov pripojených k ASUS Tinkerboard.	17
3–3	Smery definované pri ovládaní dronu pomocou SDK.	22
3–4	Tradičná kalibrácia pomocou vzoru šachovnice. a) Rutiny OpenCV dokážu odhaliť všetky rohy šachovnice. b) Rohy nemožno priradiť k príslušným bodom na vzore šachovnice, ak vzor nie je úplne zackytený.	25
3–5	Schéma procesu kalibrácie.	25

3 – 6	Súradnicový systém kamery a značky interpretovaný programom OpenCV.	30
3 – 7	Hlavné nápravy lietadla podľa normy DIN 9300.	35
3 – 8	Interpretácia vektora natočenia Rodriguesovej osi a uhla.	36

Zoznam tabuliek

5 – 1 Výsledky experimentu detektie značiek pre každý marker	54
5 – 2 Výsledky experimentu výpočtu polohy	55
5 – 3 Výsledky experimentu výpočtu polohy	57
5 – 4 Výsledky experimentu detektie značiek pre každý marker	58

Slovník termínov

Webová aplikácia - softvérová aplikácia typu klient-server, v ktorej klient (alebo používateľské rozhranie) beží vo webovom prehliadači.

Node.js - open-source, multiplatformné, back-endové prostredie pre beh JavaScriptu, ktoré beží na engine V8 a vykonáva kód JavaScriptu mimo webového prehliadača.

BE server - backendový server, ktorý spravuje údaje a logiku aplikácie a spracováva požiadavky z front-endu.

Tinkerboard - jednodoskový počítač malých rozmerov určený pre počítačových nadšencov a domácih majstrov.

ArUco markery - typ fiduciálnych značiek, ktoré možno použiť v aplikáciach rozšírenej reality.

Program pre drony - softvér, ktorý beží na palubnom počítači dronu a ovláda jeho pohyby a senzory.

Detekcia markera - proces detektie a identifikácie markera na obrázku alebo videu.

Výpočet polohy - proces určovania polohy objektu vzhľadom na známy referenčný bod alebo objekt.

Absolútne súradnice - presná poloha objektu v priestore, zvyčajne vyjadrená v súradniaciach X, Y a Z.

Režim viacerých dronov - režim, v ktorom sa súčasne ovláda viacero dronov.

Režim jedného dronu - režim, v ktorom sa súčasne ovláda len jeden dron.

Informácie o stave dronu - informácie o aktuálnom stave dronu vrátane jeho polohy, nadmorskej výšky, úrovne nabitia batérie atď.

Jedinečné ID - jedinečný identifikátor pridelený každému dronu na odlišenie od ostatných.

Úvod

Bezpilotné lietadlá (UAV) alebo drony, ako sú častejšie známe, sú čoraz bežnejšie, pričom jedným z dôvodov je ich miniaturizácia a nová kategória, ktorú vytvorili: MAV (Micro Aerial Vehicle). Kvadrokoptéry, ktoré sa dajú používať v interiéri, sa vďaka svojim malým rozmerom môžu ľahko presúvať medzi rôznymi typmi zariadení. V širokom spektri aplikácií, ako je vizuálny dohľad, monitorovanie alebo dokonca nahrávanie videa, poskytujú malé rozmery lepšiu manévrovateľnosť a možnosť lietať a skúmať menšie, ťažko dostupné oblasti.

Malá velkosť je však na úkor menšieho počtu snímačov, keďže hmotnosť je rozhodujúcim faktorom. Jediná zabudovaná kamera so správnym softvérom môže nahradiť mnohé snímače potrebné na vyhýbanie sa prekážkam, čo je dôležité pre autonómnu prevádzku. V súčasnosti ani palubné počítače kompaktných zariadení nemajú takýto výpočtový výkon, takže na spracovanie obrazu a odosielanie riadiacich údajov môžeme použiť externý počítač.

Cieľom tejto práce je ovládať a riadiť dron a viacero dronov pomocou webovej aplikácie a zverejňovať polohu dronu v priestore pomocou kamery a aruko tagov.

1 Formulácia úlohy

Analýza existujúcich prístupov a technológií pre využitie infraštruktúry internetu vozidiel v navigácii mobilných robotov. Táto časť bakalárskej práce bude obsahovať analýz existujúcich prístupov a technológií pre využitie infraštruktúry internetu vozidiel v navigácii mobilných robotov. Tento bod sa bude venovať prehľadu existujúcich riešení a technológií, ktoré umožňujú mobilným robotom využívať internetové pripojenie vozidiel a ich infraštruktúru na navigáciu a pohyb.

Návrh architektúry systému pre využitie prostriedkov internetu vozidiel v navigácii mobilných robotov. V tejto časti sa budeme zaoberať návrhom architektúry systému pre využitie prostriedkov internetu vozidiel v navigácii mobilných robotov. V rámci tohto bodu sa bude riešiť návrh architektúry, ktorá umožní efektívne využívanie prostriedkov internetu vozidiel pre navigáciu mobilných robotov.

Implementácia navrhnutého systému a jeho testovanie v rôznych prostrediach a scenároch. Tato časť bude zahŕňať implementáciu navrhnutého systému a jeho testovanie v rôznych prostrediach a scenároch. Bude zahŕňať praktickú implementáciu navrhnutého systému a jeho otestovanie v rôznych reálnych prostrediach a podmienkach.

Overenie funkčnosti a spoľahlivosti systému na lietajúcich vozidlách. Táto kapitola sa bude zaoberať overením funkčnosti a spoľahlivosti systému na lietajúcich vozidlách. V rámci tohto bodu sa bude testovať funkčnosť a spoľahlivosť systému na lietajúcich vozidlách, aby sa zabezpečilo, že systém dokáže bezchybne fungovať aj v náročných podmienkach.

Porovnanie výkonu a efektivity navrhnutého systému s existujúcimi prístupmi a technológiami. Tento bod sa bude zaoberať porovnaním výkonu a efektivity navrhnutého systému s existujúcimi prístupmi a technológiami. Tento bod bude zahŕňať porovnanie navrhnutého systému s existujúcimi riešeniami a technológiami,

aby sa zistilo, ako dobre funguje navrhnutý systém v porovnaní s konkurenčnými riešeniami.

Vypracovať dokumentáciu podľa pokynov vedúceho záverečnej práce. Posledný bod sa bude zaoberať vypracovaním dokumentácie podľa pokynov vedúceho záverečnej práce. Tento bod bude zahŕňať vypracovanie dokumentácie, ktorá bude obsahovať kompletný popis navrhnutého systému, jeho implementáciu, testovanie a výsledky porovnania s existujúcimi riešeniami.

2 Prehľad literatúry

Pred výberom metódy optického polohovania a navigácie sme museli preskúmať rôzne možnosti. Pozreli sme sa na metódy, ktoré by mohli byť vhodné na navigáciu dronov. Skúmala sa ich použiteľnosť pre mikrodrony.

2.1 Optický princíp senzorov

Snímače s optickým princípom sa bežne používajú v mobilnej robotike na navigáciu, lokalizáciu a mapovanie. Tieto senzory zachytávajú informácie z prostredia analýzou svetla a jeho interakcie s povrchmi, čím poskytujú bohaté údaje na presné určovanie polohy robota a plánovanie trajektórie. V tejto časti sa budeme zaoberať rôznymi typmi optických principiálnych snímačov a ich aplikáciami v mobilnej robotike [1].

Jedným z najčastejšie používaných optických snímačov je kamera, ktorá zachytáva obrázky a videá prostredia. Kamery poskytujú bohaté vizuálne informácie vrátane farby, textúry a tvaru, ktoré možno využiť na detekciu, sledovanie a rozpoznávanie objektov. V našej práci využívame kamery namontované na bezpilotných lietadlách Tello na zisťovanie polohy značiek Aruco a odhadovanie ich súradníc v 3D priestore [1].

Ďalším dôležitým optickým senzorom je senzor LIDAR (Light Detection and Ranging), ktorý vysiela laserové lúče a meria ich odraz na vytvorenie 3D mapy prostredia. Senzory LIDAR sa bežne používajú v autonómnych vozidlách na vyhýbanie sa prekážkam a mapovanie. Sú však drahé a vyžadujú vysoký výpočtový výkon, čo ich robí menej vhodnými pre malú mobilnú robotiku [2].

Okrem kamier a LIDAR-u patria medzi ďalšie typy snímačov na optickom princípe infračervené snímače, ultrazvukové snímače a snímače času letu. Infračervené senzory merajú odraz infračerveného svetla na detekciu objektov a meranie ich vzdialnosti, zatiaľ čo ultrazvukové senzory vysielajú vysokofrekvenčné zvukové vlny a

merajú ich ozvenu na odhad vzdialenosťi. Snímače času letu využívajú svetlo na meranie času, za ktorý sa signál odrazí späť, čím poskytujú presné merania vzdialnosti.

Senzory na optickom princípe celkovo zohrávajú kľúčovú úlohu v mobilnej robotike a poskytujú bohaté informácie na navigáciu a lokalizáciu robota. Pochopením rôznych typov senzorov a ich aplikácií môžeme navrhnúť a implementovať efektívne robotické systémy, ktoré môžu fungovať v rôznych prostrediach a scenároch.

2.2 Vyhýbanie sa prekážkam a orientácia na základe obrazu z kamery

Jedným z najdôležitejších úloh mobilných robotov je schopnosť vyhnúť sa prekážkam a správne sa orientovať v prostredí. K dispozícii sú rôzne senzory, ktoré umožňujú robotom rozpoznať a vyhnúť sa prekážkam. Okrem senzorov ako LIDAR, ultrazvukových a infračervených senzorov sa dá využiť aj obraz z kamery.

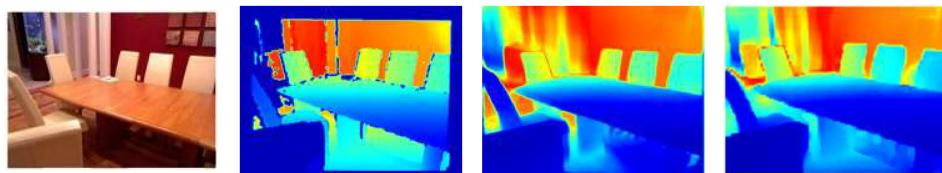
Kamera poskytuje bohaté vizuálne informácie o prostredí a umožňuje robotom získať informácie o prekážkach a teréne. Existujú rôzne spôsoby, ako využiť obraz z kamery pre navigáciu a vyhýbanie sa prekážkam.

2.2.1 S vylepšenou neurónovou sieťou

Jednou zo skúmaných metód na vyhýbanie sa prekážkam a orientáciu pomocou kamery bolo strojové učenie pomocou neurónových sietí. Nakoniec však bola zavrhnutá kvôli časovo a hardvérovo náročným procesom učenia. Táto metóda zahŕňa použitie naučenej neurónovej siete na priradenie približných hodnôt hĺbky každému pixelu na základe polohy objektov z jedného obrazu. Vzory sa dajú naučiť z dvojice LIDAR a digitálnych kamier umiestnených vedľa seba. Zadaním dostatočného počtu týchto naučených dvojíc môže neurónová sieť vytvoriť vzťah medzi pixelmi na obrázkoch a hodnotami hĺbky získanými z LIDAR-u alebo stereokamery.

Siet sa môže učiť pomocou učenia založeného na LIDAR-e s dohľadom a učenia založeného na stereopároch bez dohľadu na základe pravdepodobnostného princípu. Učenie pod dohľadom je však často príliš prísne a učenie bez dohľadu poskytuje nepresné výsledky. Preto sa odporúča používať poloprevádzkové učenie alebo porovnávať výsledky získané pomocou siete s kalibrovaným systémom detekcie hĺbky počas prevádzky [3].

Táto metóda získava hodnoty hĺbky z okolia a umiestnenia objektov (množiny pixelov) a vytvára mapu hĺbky podobnú tej, ktorá je znázornená na obrázku 1. Jej presnosť je zatiaľ experimentálna a takýto systém je stále vystavený mnohým chybám. Napriek tomu bol použitý v aplikáciách na riadenie dronov. Hĺbkové mapy na obrázku 1 ilustrujú rozdiel medzi rôznymi metódami a presnosť tejto metódy [4, 3].



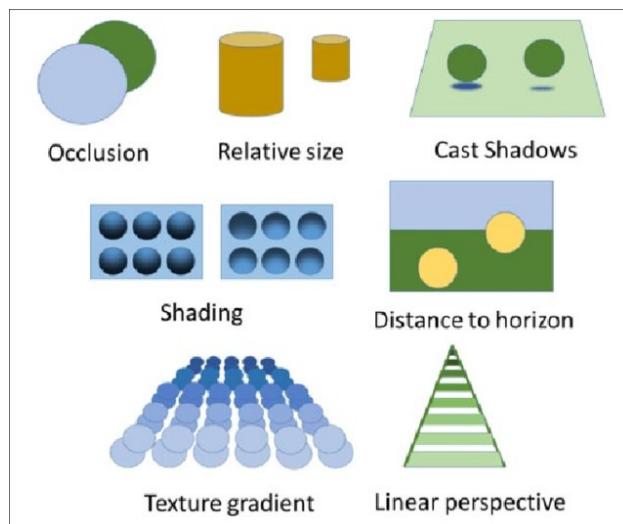
Obrázok 2–1 Načítaný obraz (vľavo), skutočná mapa hĺbky na základe senzorov (vľavo uprostred) a napokon mapa hĺbky vytvorená pomocou Gaussovoho modelu (vpravo uprostred) a Laplaceovho modelu (vpravo)

Neurónová siet berie do úvahy niekoľko faktorov:

- **oklúzia**
- **relatívna veľkosť**
- **vrhaný tieň**
- **tieňovanie**
- **vzdialenosť od horizontu**
- **gradient textúry**

- lineárna perspektíva

Učenie siete je časovo a zdrojovo náročné. Po vyškolení je výzvou zabezpečiť, aby sieť fungovala v reálnom čase na riadenie dronov bez toho, aby sa stratilo príliš veľa údajov v dôsledku zrýchlenia kalibrácie, čo môže viest ku kolíziám. Počas pomalého letu sa siet stále dokáže vyhýbať prekážkam, ale považuje ich len za súbor bodov s relatívnou polohou, nie za presné obrysy [5].



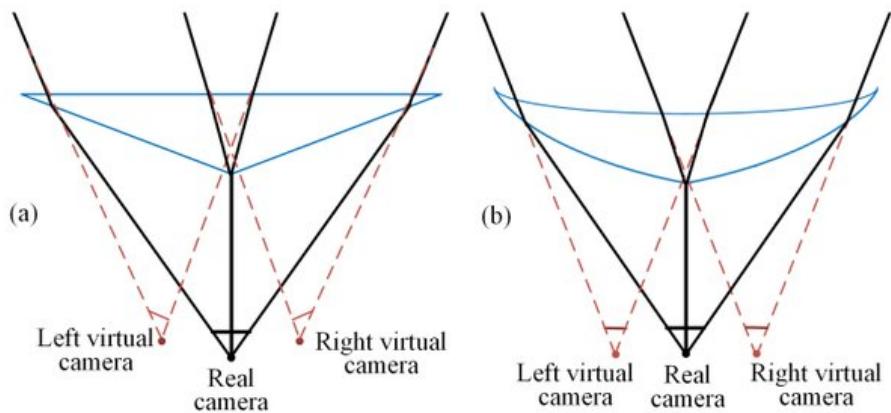
Obrázok 2 – 2 Faktory zohľadňované pri monokulárnom vnímaní hĺbky

2.2.2 Princíp stereosnímania

Princíp stereosnímania je široko používaná metóda vnímania hĺbky v počítačovom videní, robotike a príbuzných oblastiach. Zahŕňa použitie dvoch alebo viacerých kamier na snímanie obrazov tej istej scény z rôznych uhlov pohľadu a následné použitie rozdielov v obrazoch na výpočet vzdialenosí objektov na scéne. Tento princíp sa uplatňuje pri rôznych úlohách v robotike vrátane rozpoznávania objektov, navigácie a vyhýbania sa prekážkam [6].

V kontexte bezpilotných lietadiel môže byť princíp stereočočky užitočným nástrojom na navigáciu a vyhýbanie sa prekážkam. Výhodou použitia stereolitografie je, že je k dispozícii mnoho volne použiteľných riešení, vrátane knižnice OpenCV. Po-

užíva sa aj na ovládanie dronov, ale jeho spôsobnosť pri vysokých rýchlosťach je veľmi nízka. Na vytvorenie hĺbkovej mapy je potrebná dokonalá kalibrácia a pevná fixácia kamery, inak systém poskytne chybné údaje. Naš dron DJI Tello má však napríklad len jednu kameru. V takomto prípade možno obraz z kamery rozdeliť na dve virtuálne kamery, ktoré sa potom môžu použiť na simuláciu stereovidenia, ako je znázornené na obrázku 2-3. Tento prístup má však svoje obmedzenia a výzvy [6].



Obrázok 2 – 3 Model virtuálnej kamery. (a) Model virtuálnej kamery systému stereovízie založeného na plochej prizme. (b) Modifikovaný model dierkovej virtuálnej kamery systému stereovízie založeného na voľnej prizme.

Jednou z hlavných výziev je, že rozlíšenie virtuálnych kamier je nižšie ako rozlíšenie skutočnej kamery. Toto zníženie rozlíšenia môže ovplyvniť presnosť vnímania hĺbky, najmä v prípade vzdialených objektov. Ďalšou výzvou je, že základná vzdialenosť medzi dvoma virtuálnymi kamerami je pevná a nedá sa upraviť, čo môže obmedziť rozsah vzdialenosťí, ktoré možno presne odhadnúť [7].

Okrem toho výkonnosť princípu stereosnímania môžu ovplyvniť rôzne faktory, ako sú svetelné podmienky, kalibrácia kamery a oklúzie. Tieto faktory môžu mať za následok chyby vo vnímaní hĺbky, čo môže viesť ku kolíziám alebo nepresnej navigácii [7].

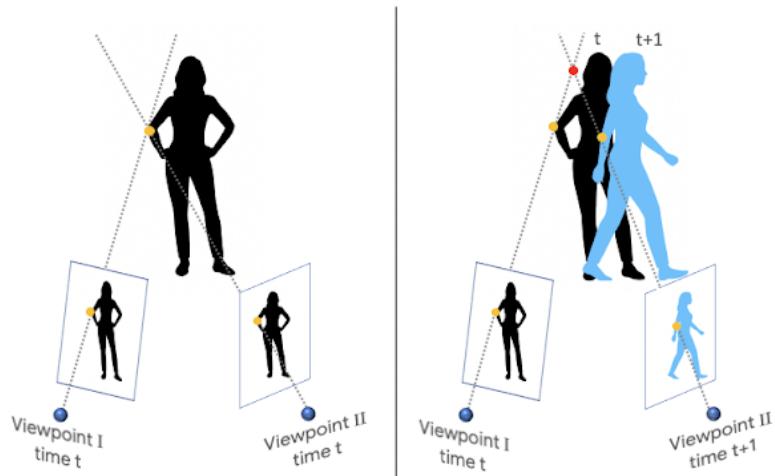
Vzhľadom na tieto problémy je pochopiteľné, prečo princíp stereosnímania nemusí

byť vždy najlepším riešením pre navigáciu dronov. Namiesto toho môžu byť na tieto úlohy vhodnejšie iné techniky.

Na záver možno konštatovať, že hoci je princíp stereosnímania výkonnou metódou na vnímanie hĺbky a úspešne sa uplatňuje v rôznych oblastiach, jeho použitie v kontexte dronov, ako je Tello, je obmedzené z dôvodu použitia iba jednej kamery. V dôsledku toho môže byť potrebné použiť iné techniky snímania v spojení s týmto princípom, aby sa dosiahla presná a spoľahlivá navigácia dronov.

2.2.3 Na základe pohyblivého obrazu kamery

Ďalším populárnym prístupom k dosiahnutiu vnímania hĺbky v bezpilotných lietadlách je použitie metód založených na snímaní pohyblivou kamerou. Táto metóda zahŕňa snímanie sekvencie obrazov pomocou jednej kamery namontovanej na dron a následné použitie algoritmov na odhad informácií o hĺbke na základe pohybu kamery.



Obrázok 2–4 Vľavo: Tradičné stereofónne nastavenie predpokladá, že aspoň dva uhly pohľadu zachytávajú scénu v rovnakom čase. Vpravo: Uvažujeme o nastavení, pri ktorom sa kamera aj objekt pohybujú.

Ide o riešenie, o použití ktorého v našej práci sa uvažovalo dlho. Myšlienka spočíva

v tom, že ak nemáte dve kamery, môžete dronom pohybovať a fotografovať objekt z viacerých uhlov, ako je znázornené na obrázku 2-4. Na rozdiel od stereolitografie zhodujeme desiatky snímok namiesto dvoch a porovnávame pixely medzi sebou a s polohou kamery v čase zhodenia snímky. Takto môžeme získať hĺbkový obraz na základe viacerých uhlov pohľadu a pozícií kamery. Týmto spôsobom môžeme získať hĺbkový obraz na základe viacerých uhlov pohľadu a pozícií kamery. Prekvapujúce bolo, že na základe štúdie Alvareza [8] chyby polohovania dronu počas vznášania poskytujú dostatočný posun na vytvorenie mapy hĺbky a na získanie potrebných údajov stačí 30 snímok.

V porovnaní s predchádzajúcimi uvedenými metódami má prístup založený na snímaní pohyblivou kamerou tú výhodu, že nevyžaduje žiadne ďalšie snímače alebo zariadenia. Okrem toho umožňuje väčšiu flexibilitu, pokiaľ ide o umiestnenie a pohyb dronu.

Tento prístup však nie je bez obmedzení. Presnosť informácií o hĺbke získaných touto metódou do veľkej miery závisí od kvality použitých algoritmov na odhad pohybu. Okrem toho je táto metóda náchylná na chyby spôsobené faktormi, ako je rozmazenie pohybu kamery, zlé svetelné podmienky a oklúzia. V tomto prípade bola táto myšlienka zavrhnutá aj kvôli šumu z akcelerometrov a chybám, ktoré môžu vzniknúť.

Napriek týmto obmedzeniam sa metóda založená na snímaní pohyblivej kamery úspešne používa v rôznych aplikáciách dronov, ako je vyhýbanie sa prekážkam a mapovanie. V porovnaní s ostatnými už uvedenými metódami môže byť tento prístup pre svoju jednoduchosť a flexibilitu vhodnejší pre určité prípady použitia.

2.2.4 Určovanie polohy pomocou značiek

Určovanie polohy pomocou značiek je populárna metóda na presný odhad polohy (pozície a orientácie) kamery vzhľadom na prostredie. Táto metóda zahŕňa umiest-

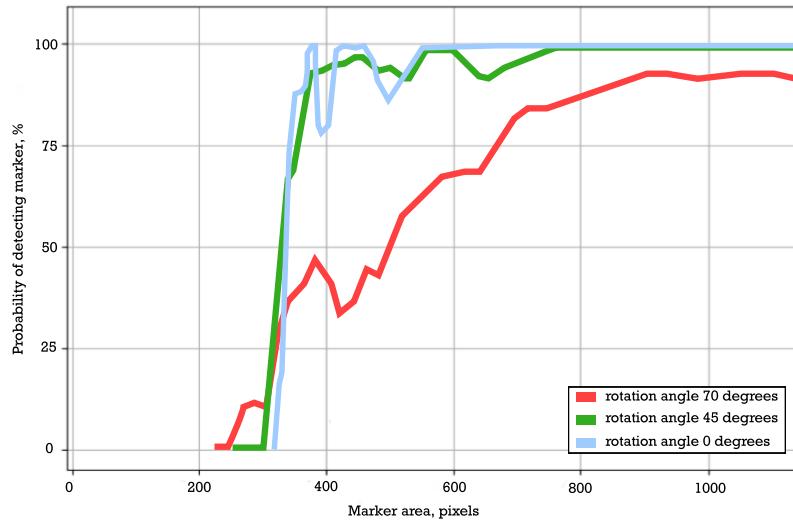
nenie značiek so znáomou geometriou v prostredí a ich detekciu v obrazoch kamery. Značky Aruco, ktoré sú založené na štvorcových čiernobielych vzoroch s jedinečným kódom, sú oblúbenou voľbou pre túto metódu vďaka ich vysokej presnosti a odolnosti voči rôznych svetelných podmienkam [9].

Proces používania značiek Aruco na určovanie polohy zahŕňa detekciu značiek v obraze kamery dronu, použitie znáomej veľkosti a polohy značiek na výpočet polohy a orientácie dronu vzhľadom na značky a následné použitie týchto informácií na navigáciu dronu na požadované miesto alebo vykonanie konkrétnej úlohy.

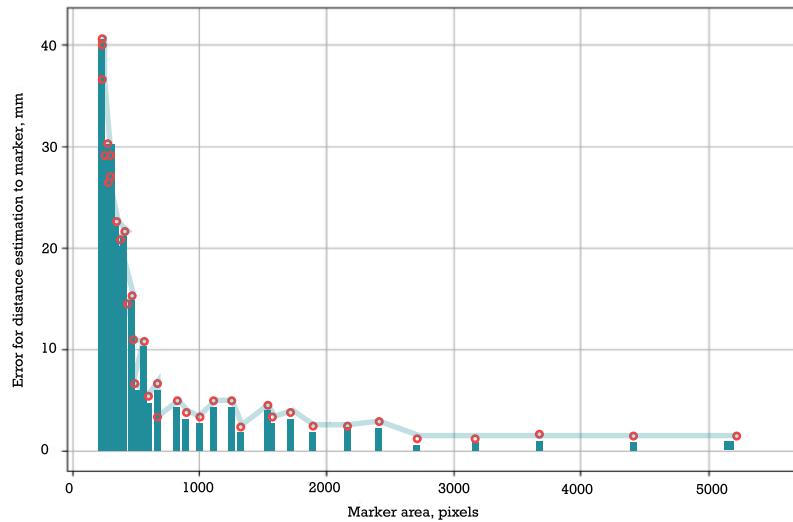
V porovnaní s predchádzajúcimi metódami, ktoré ste spomenuli, má určovanie polohy pomocou markerov tú výhodu, že je veľmi presné a spoloahlivé, a to aj v náročných podmienkach. Nevyžaduje si ani samostatné nastavenie stereokamery alebo zložité trénovanie neurónovej siete, čo z nej robí relatívne jednoduchú a efektívnu metódu na implementáciu.

Vo svojej predchádzajúcej práci som použil metódu určovania polohy pomocou markerov s markermi Aruco na detekciu a sledovanie polohy robota v kontrolovanom prostredí. Výsledky ukázali, že metóda výborne funguje v rôznych situáciách a dokáže poskytnúť presné a spoloahlivé odhady polohy robota. Pokiaľ ide o presnosť nameraných polôh značiek, zistili sme, že sme mohli zistiť značky v širokom rozsahu s priemernou presnosťou $\pm 0,05$ m, a to aj v prípade malých bočných značiek (0,03 m).

V mojej súčasnej diplomovej práci budeme používať rovnakú metódu na odhad polohy dronu vybaveného jednou kamerou. Keďže metóda sa spolieha na detekciu značiek na snímkach z kamery, je dôležité zabezpečiť, aby boli značky na snímkach ľahko rozlíšiteľné a viditeľné. Ukázalo sa, že markery Aruco dobre fungujú v rôznych prostrediach a za rôznych svetelných podmienok, a preto sú vhodnou voľbou pre našu prácu [10].



Obrázok 2 – 5 Pravdepodobnosť detekcie značky v závislosti od jej oblasti na obrázku pre 3 rôzne uhly naklonenia: 0° , 45° , 70° .



Obrázok 2 – 6 Závislosť vplyvu veľkosti oblasti značky na chybe pri zistovaní jej polohy.

V porovnaní s predchádzajúcimi metódami, o ktorých sme hovorili, je výhodou metódy určovania polohy pomocou značiek vysoká presnosť a spoloahlivosť. Jednou z potenciálnych nevýhod používania značiek je však to, že sa musia do prostredia umiestniť vopred, čo nemusí byť praktické vo všetkých situáciách. Okrem toho sa

samotné markery môžu zakryť alebo premiestniť, čo by mohlo spôsobiť problémy s dronom. Napriek týmto obmedzeniam som sa rozhodol použiť túto metódu vo svojej práci kvôli jej overenej presnosti a robustnosti.

2.3 WebSocket

WebSocket je protokol, ktorý poskytuje plne duplexný komunikačný kanál prostredníctvom jedného spojenia TCP. Umožňuje komunikáciu v reálnom čase medzi klientom a serverom, vďaka čomu je vhodný pre aplikácie, ktoré vyžadujú časté aktualizácie alebo nízku latenciu, ako sú hry v reálnom čase, chatové aplikácie a živé dátové kanály.

V kontexte tohto projektu poskytuje použitie technológie WebSocket spôsob komunikácie dronu so serverom v reálnom čase. To je obzvlášť dôležité pre aplikácie, ktoré vyžadujú, aby dron rýchlo reagoval na príkazy alebo napríklad streamovanie videa v reálnom čase.

Jednou z hlavných výhod používania technológie WebSocket je jej nízka latencia. Na rozdiel od tradičných požiadaviek HTTP, ktoré si vyžadujú samostatnú požiadavku a odpoveď pre každú informáciu vymieňanú medzi klientom a serverom, spojenia WebSocket zostávajú otvorené a umožňujú nepretržitú komunikáciu medzi nimi. To umožňuje rýchlejšiu a efektívnejšiu komunikáciu, ktorá je ideálna pre aplikácie v reálnom čase.

Existuje niekoľko alternatívnych technológií, ktoré možno použiť na komunikáciu v reálnom čase, napríklad dlhé dotazovanie, udalosti odosielané serverom (SSE) a WebRTC. Dlhé dotazovanie zahŕňa nepretržité dotazovanie servera na aktualizácie, zatiaľ čo SSE umožňuje serveru posielat aktualizácie klientovi. WebRTC je komunikačný protokol typu peer-to-peer, ktorý umožňuje prenos zvuku a videa v reálnom čase.

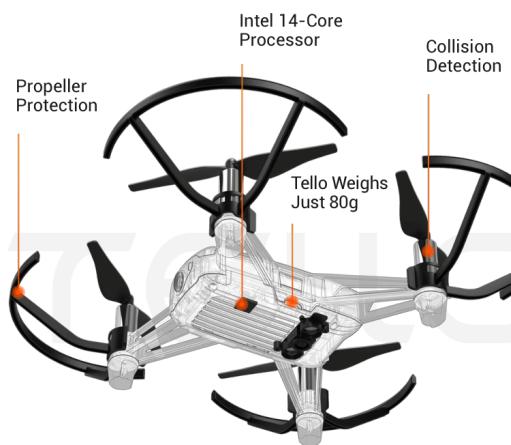
WebSocket je však často uprednostňovanou voľbou na komunikáciu v reálnom čase vďaka nízkej latencii a jednoduchému používaniu. Okrem toho je WebSocket široko podporovaný modernými webovými prehliadačmi a možno ho ľahko integrovať do webových aplikácií pomocou populárnych rámcov, ako sú Node.js alebo Django.

3 Design

V tejto časti sa budeme zaoberať návrhom nášho systému riadenia dronov vrátane hardvérových a softvérových komponentov a ich vzájomnej interakcie. Cieľom je poskytnúť komplexný prehľad návrhu systému vrátane technických špecifikácií a funkcií, aby ostatní mohli systém pochopiť a replikovať. Budeme tiež diskutovať o rôznych konštrukčných úvahách, ktoré boli súčasťou vývoja systému, a o tom, ako sme ich riešili.

3.1 Výber dronu

Po preskúmaní funkcií možných programovateľných dronov je najlepším zariadením kvadrokoptér Tello, ktorú vyvinula spoločnosť Ryze, ale podporuje ju spoločnosť DJI. Dron Tello je malý quadcopter určený na lietanie v interiéri, ktorý má procesor Intel. Má rozmerы len 98 x 92,5 x 41 mm a váži len 87 gramov, čo uľahčuje jeho prepravu a manévrovanie v stiesnených priestoroch. Dron je vybavený 5 MP kamerou, ktorá dokáže zachytíť video s rozlíšením 720p pri 30 snímkach za sekundu, takže je vhodný na základné fotografovanie a videografiu. Má tiež optický snímač toku smerujúci nadol a infračervený snímač vzdialenosť, ktoré mu pomáhajú udržiavať stabilný let v interiéri.



Obrázok 3 – 1 Obrázok dronu DJI Tello zobrazujúci jeho komponenty.

Dron Tello možno naprogramovať pomocou rôznych softvérových vývojových súprav (SDK). Jedným z populárnych SDK na programovanie dronu Tello je DJI Tello SDK, ktorý poskytuje súbor API na ovládanie letu, kamery a ďalších funkcií dronu. Súbor Tello SDK je k dispozícii pre viaceré programovacie jazyky vrátane jazykov Python, Java a Swift, vďaka čomu je prístupný vývojárom s rôznym vzdelením a úrovňou zručnosti .

Okrem DJI Tello SDK je k dispozícii aj niekolko SDK a knižníc tretích strán na programovanie dronu Tello vrátane knižnice TelloPy pre Python a knižnice Node.js Tello pre Node.js. Tieto knižnice poskytujú ďalšie funkcie na ovládanie dronu a spracovanie údajov z jeho senzorov a môžu byť užitočné pre pokročilejšie projekty .

Celkovo je dron Tello všeobecne a cenovo dostupnou možnosťou pre interiérové aplikácie dronov a jeho podpora viacerých programovacích jazykov a SDK ho sprístupňuje vývojárom s rôznym vzdelením a úrovňou zručnosti.

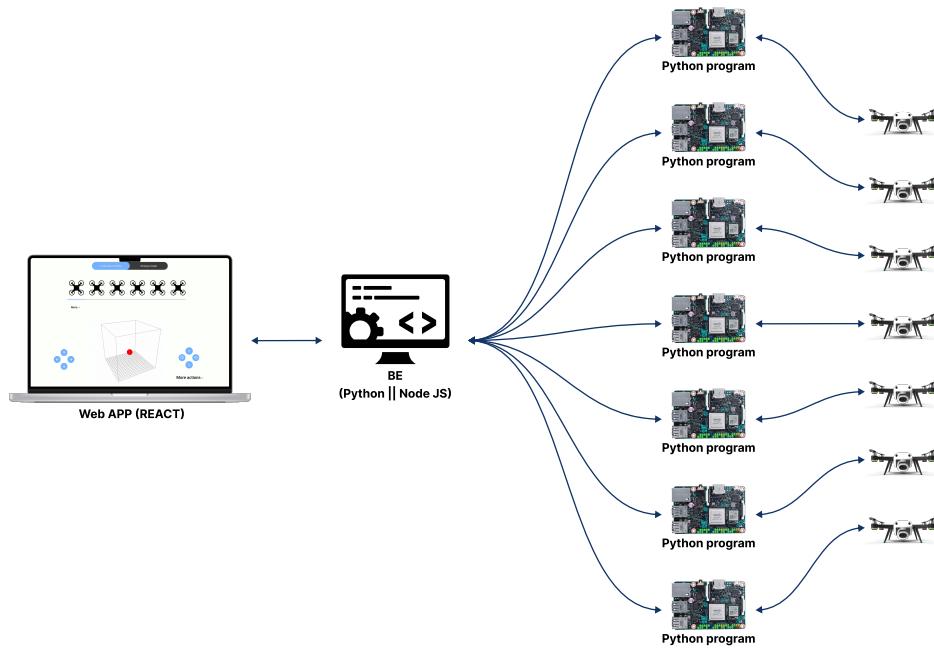
3.2 Programovací jazyk a súbor nástrojov

Systém je založený na webovej aplikácii vyvinutej pomocou React, ktorá umožňuje používateľom ovládať drony v reálnom čase. Webová aplikácia komunikuje s backendovým serverom vytvoreným pomocou Node.js, ktorý funguje ako sprostredkovateľ medzi používateľom a dronmi [11].

Frontend webovej aplikácie je vytvorený pomocou React, populárnej JavaScriptovej knižnice na vytváranie používateľských rozhraní. React umožňuje vytvárať opakovane použiteľné komponenty používateľského rozhrania, ktoré možno ľahko spravovať a aktualizovať. Okrem React využíva frontend aj ďalšie webové technológie, ako sú HTML5 a CSS3 [11].

Backendový server je vytvorený pomocou Node.js, populárneho prostredia na spušťanie JavaScriptu. Node.js umožňuje vytvárať škálovateľné a vysoko výkonné sieťové

aplikácie, preto je vhodnou voľbou na vybudovanie backendu systému. Backendový server komunikuje s frontendom prostredníctvom rozhrania RESTful API a spojení WebSocket.



Obrázok 3 – 2 Schéma návrhu systému: Webová aplikácia vytvorená pomocou React komunikuje s backendovým serverom Node.js, ktorý posielá príkazy a prijíma telemetrické údaje z dronov pripojených k ASUS Tinkerboard.

Na ovládanie dronov a prijímanie ich telemetrických údajov je backendový server pripojený k doske ASUS Tinker Board. Tinker Board je výkonný jednodoskový počítač, na ktorom beží upravená verzia systému Linux. Na každej doske Tinker Board je pre každý dron spustený program v jazyku Python. Tieto programy v jazyku Python sú zodpovedné za prijímanie príkazov z backendového servera a ich odosielanie do dronov, ako aj za prijímanie telemetrických údajov z dronov a ich odosielanie späť na backendový server.

Nakoniec systém využíva aj značky Aruco na zistovanie polohy a orientácie dronov, ktoré sa používajú na riadenie pohybu dronov. Detekcia a sledovanie značiek Aruco

sa vykonáva pomocou populárnej knižnice počítačového videnia OpenCV. Systém je tiež vybavený kamerou namontovanou na každom drone, ktorá zachytáva živý videozáznam, ktorý sa v reálnom čase prenáša späť do webovej aplikácie pomocou WebSocketov.

3.3 Ako ovládať dron tello

V dokumentácii poskytnutej výrobcom je podrobne opísané, ako ovládať dron pomocou programu Python. [19] Po pripojení k vlastnej sieti wifi dronu je potrebné otvoriť spojenie UDP na ovládanie dronu prostredníctvom adresy 192.168.10.1 a portu 8889. Pred spustením ovládania je vždy potrebné poslať dronu príkaz "command", ktorý sa prepne do stavu, v ktorom čaká na príkazy [11]. Z dronu je tiež možné čítať stav letového ovládača, ktorý využíva aj firmvér zariadenia, otvorením servera 0.0.0.0 na porte 8890. Ten nebol súčasťou vyššie uvedenej knižnice GitHub, preto sme ho do programu pridali. Otvorili sme nové spojenie so soketom UDP so spomínaným serverom a údajmi na porte bežiacim v samostatnom vlákne, podobne ako pri prijímaní odpovedí z kontroly dronu na pozadí.

Code Listing 1 Úryvok kódu 1 ukazuje obsluhu vlákna obsiahnutú v jazyku Python

```
# Run Tello command responses UDP receiver on background
client_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
response_receiver_thread = Thread(target=Tello.
                                    udp_response_receiver)
response_receiver_thread.daemon = True
response_receiver_thread.start()

# Run state UDP receiver on background
state_receiver_thread = Thread(target=Tello.udp_state_receiver)
state_receiver_thread.daemon = True
state_receiver_thread.start()

threads_initialized = True
```

Threading library, ktorý možno priradit konkrétnnej funkcií, ktorá bude bežať na vlákne oddelenom od hlavného programu. Vlákno, ktoré číta stav, ukazuje na funkciu `get_read_state()` triedy `Tello` a spúšta ju v samostatnom vlákne, takže beží na pozadí. Mali by ste zapnúť voľbu `daemon = True`, aby sa vlákno spúšťalo ako daemon, t. j. ak sa ukončí program vyššej úrovne, ukončí sa aj vlákno daemon. Ak táto možnosť nie je zapnutá, budete musieť ukončenia vlákien riešiť samostatne [11].

Stavy vysiela dron na spomínanom serveri, ich čítanie nebude spomaľovať čítanie obrazu z kamery dronu, pretože dron bude vysielat retazec stavov po vstupe do príkazového režimu, aj keď ho nebudeme čítať. V oficiálnej dokumentácii je uvedený formát, v ktorom dron vysiela stavy svojich senzorov. Vysiela jeden súvislý, stredníkmi ohraňčený textový súbor ASCII cez bezdrôtové spojenie UDP na prijímajúci server v nasledujúcim formáte: `"pitch:%d;roll:%d;yaw:%d;vgx:%d;vgy:%d;vgz:%d;temp1:%d;temph:%d;t of:%d;h:%d;bat:%d; baro:%.2f;time:%d;agx:%.2f;agy:%.2f;agz:%.2f;\r\n"` Kde sú uvedené hodnoty jednotlivých stavov (interpretácia súradníc je uvedená na obrázku 8):

- **pitch, roll, yaw:** vlastná rotácia dronu okolo osí x, y a z, meraná ako celý uhol, kladný v smere hodinových ručičiek, podľa pravidla ľavej ruky
- **vgx, vgy, vgz:** rýchlosť dronu nastavená v smeroch x, y, z v cm/s (nie sú to skutočné merateľné hodnoty rýchlosťi, ale hodnota priradená k rýchlosťi dronu, ktorá sa mení počas riadenia).
- **temp:** najnižšia teplota v C°
- **temph:** najvyššia teplota v C°
- **t of:** výška kamery času letu v cm v spodnej časti dronu
- **h:** výška v cm

- **bat:** zostávajúca kapacita batérie dronu v celých percentách
- **baro:** hodnota výšky na základe nainštalovaného barometra v cm
- **time:** čas zapnutia motorov (čas letu) v sekundách
- **agx, brain, agz:** zrýchlenie dronu v smere x, y, z zo snímača zrýchlenia interpretované v 0,001 g, kde g je gravitačné zrýchlenie

Toto sa dá rozdeliť na pole pozdĺž stredníkov pomocou funkcie Python string.split(" ; ") a potom sa hodnoty za dvojbodkou v poli prevedú na číselný typ v príslušnom formáte. Na číslo sa prevedú len požadované hodnoty a potom sa hodnoty načítajú do zoznamu, ktorý sa odovzdá do riadku. V Pythone je v rámci obsluhy vlákien len typ Queue, schopný zabezpečiť bezpečnu komunikáciu medzi vláknenami; ak sa nepoužíva na výmenu údajov medzi vláknenami v programe, program môže zamrznúť.

Code Listing 2 Funkcia na čítanie stavov dronov

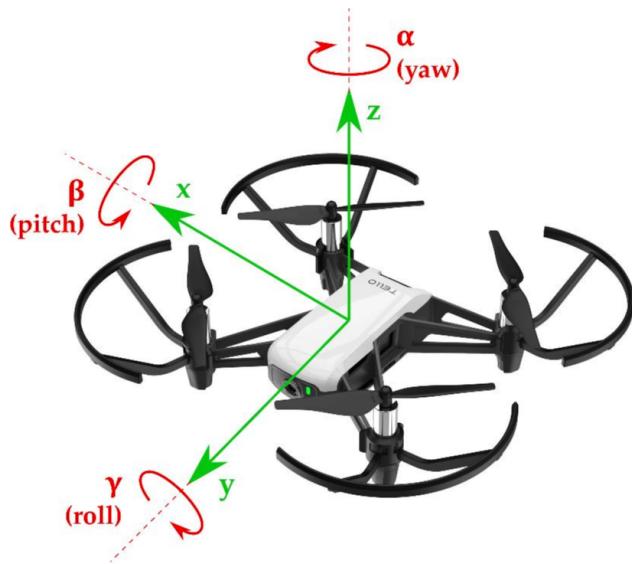
```
def get_read_state(self):
    while True:
        time.sleep(1/25)
        try:
            state_temp, _ = self.stateSocket.recvfrom(1024)
            self.state = state_temp.decode('ASCII').split(" ; ")
            pitch = -int(self.state[0][self.state[0].index(":") + 1:])
            )
            roll = -int(self.state[1][self.state[1].index(":") + 1:])
            yaw = -int(self.state[2][self.state[2].index(":") + 1:])
            tof = int(self.state[10][self.state[8].index(":") + 1:])
            bat = int(self.state[10][self.state[10].index(":") + 1:])
            self.data_queue.put([pitch, roll, yaw, tof, bat])
        except Exception as e:
            self.LOGGER.error(e)
            break
```

Počas riadenia máte tiež možnosť čítať stav tak, že pošlete do vlákna dotaz ako príkaz, na ktorý odpovie prostredníctvom spojenia UDP. Tieto dotazové slová poskytujú rovnaké stavy senzorov ako čítanie stavu (napr. : äkcelerácia?", "rýchlosť?", "batéria?", "tof?"...). Zvažovalo sa aj použitie tejto funkcie, ale testy ukázali, že táto metóda poskytuje menej spoloahlivé výsledky. Táto komunikácia, na rozdiel od čítania stavu, spomaľuje a preťahuje ostatné funkcie dronu, pretože na reakciu na ne musí zabezpečiť samostatnú procesorovú kôru. Kedže teda nemôžeme získať spoloahlivé rezultáty ani s obsluhou vlákien, táto funkcia nie je pre naše účely užitočná.

Obraz z kamery, ktorý je pre túto úlohu nevyhnutný, sa tiež vysielá cez UDP, pričom sa otvorí server 0.0.0.0 cez port 11111. Kedže veľkosť zakódovaných údajov obrazu je väčšia ako maximálna veľkosť užitočného zaťaženia prenosu UDP, obrazové údaje dron prenáša v blokoch po 1460 bajtov, z ktorých posledný blok je menší ako 1460 bajtov. Vysielaný obraz je možné po prijatí zobraziť pomocou balíka OpenCV. Balíky kodekov ffmpeg a libh264decoder uvedené v oficiálnej dokumentácii DJI-SDK preto v tomto prípade nie sú potrebné, testovanie programu prebieha pod operačným systémom MacOS. Ovládacie inštrukcie SDK možno rozdeliť do troch skupín:

1. inštrukcie na riadenie
 - (a) vrátenie "ok", ak sú splnené
 - (b) ěrrorälebo chybové hlásenie, ak nie je
2. inštrukcie na čítanie
 - (a) vráti hodnotu požadovaného parametra
3. ovládanie nastavením parametrov
 - (a) vrátenie "ok", ak sú splnené
 - (b) ěrrorälebo chybové hlásenie, ak nie je

Dronovi sa pošle príkaz "takeoff" (vzlet), vzlietne, pristane na príkaz "land" (pristátie), príkazy "streamon" (zapnutie) a "streamoff" (vypnutie) na prepnutie streamovania videa. Samotný dron je možné ovládať podľa smeru a súčasne príkazom "go x y z speed". V tomto prípade sa dron bude pohybovať zadanou rýchlosťou vzhľadom na danú polohu v smere určenom troma súradnicami. Presnosť tohto ovládania sa rovná rozmerom dronu, takže sa nemôže pohybovať o menej ako 20 centimetrov [11].



Obrázok 3 – 3 Smery definované pri ovládaní dronu pomocou SDK.

Firmvér poskytuje aj možnosť ovládať dron v oblúku: zadáním ďalších dvoch bodov so súradnicami x, y, z vzhľadom na aktuálnu polohu dronu opíše nimi definovaný oblúk, ak je polomer oblúka v rozmedzí 0,5 až 10 metrov.

Pre túto úlohu je na základe testov s dronom najvhodnejším spôsobom riadenia tzv. RC riadenie, ktoré patrí do skupiny 3 spôsobov riadenia. Rýchlosť dronu v smeroch x, y, z a odklonu možno nastaviť príkazom "rc x y z yaw". S celočíselnými hodnotami od -100 do 100. (Rýchlosť yaw je tu tiež vyjadrená v súradnicovom systéme bal offset). Takto môžete dokonca vykresliť krivku odoslaním príkazov s danou frekvenciou za predpokladu, že hodnoty rýchlosť parametrickej krivky sú entované ako kontrola vzorkovaním s rovnakou frekvenciou. Pri tomto režime riadenia

možno dosiahnuť menšie posuny ako pri už spomínaných príkazoch "go x y z speed". Do 2. skupiny inštrukcií patria inštrukcie, ktoré sa pýtajú na údaje zo senzorov. Napríklad aktuálne nastavenú rýchlosť, nabitie batérie, čas letu, nadmorskú výšku, zrýchlenie a uhlovú rýchlosť možno načítať z palubného ovládača dronu. Tieto sa nepoužívajú z dôvodu nespoľahlivej čitateľnosti, ktorá bola spomenutá vyššie.

3.4 Umiestnenie dronu

Na zabezpečenie presného a spoloahlivého ovládania dronov je nevyhnutné optimálne umiestnenie dronov a značkovačov Aruco. Drony by mali byť umiestnené na otvorenom priestranstve s minimom prekážok a jasnou viditeľnosťou na značky. Značky by mali byť umiestnené tak, aby umožňovali maximálne pokrytie záujmovej oblasti a zároveň boli ľahko zistiteľné kamerou dronu.

Je tiež dôležité zabezpečiť, aby boli značky umiestnené v rovnakej výške a orientácii, čo ulahčí presnú detekciu a sledovanie. Do úvahy by sa mali brať aj akékoľvek zmeny svetelných podmienok, pretože môžu ovplyvniť detekciu a sledovanie značiek.

Okrem fyzického umiestnenia je tiež dôležité správne kalibrovať nastavenia dronu a kamery. To zahŕňa nastavenie parametrov, ako je rozlíšenie kamery, ohnisková vzdialenosť a korekcia skreslenia, aby sa zabezpečilo presné meranie a umiestnenie.

Dôkladným zvážením všetkých týchto faktorov možno umiestnenie dronu optimalizovať tak, aby sa dosiahlo čo najlepší výkon a presnosť pre danú aplikáciu.

Údaje o zrýchlení by sa mohli použiť na určenie polohy, ale po odoslaní príkazu "zrýchlenie?" dáva čítanie veľmi nejasnú odpoveď. Dokonca aj po zmene časového limitu spojenia na požiadavku odosielanú s intervalom 0,1 sekundy nastaveného počas testov, stále dával zmysluplnú odpoveď po 0,5 sekundy, ale často len vyhadzoval chybu timeout. Takto boli prirodzene zašumene údaje akcelerometra načítané dokonca so značným vzorkovacím šumom.

Druhou možnosťou je otvoriť server bežiaci v samostatnom vlákne, ktorý bude prijímať už spomínané stavové údaje. V tomto prípade dostávate údaje v jednom reťazci vrátane údajov akcelerometra a gyroskopu. Tieto údaje by bolo potrebné najprv prehnati cez ďalší filter [14] alebo Kalmanov filter [15] na odstránenie šumu zo senzorov. Dvojnásobnou integráciou hodnôt zrýchlenia možno získať posun dronu, ale integračná operácia zosilňuje šum, podlieha integračnej chybe a nedá sa korigovať v prípade absencie stabilných bodov polohy. Táto možnosť bola zavrhnutá z dôvodu akumulácie šumu pri integrácii.

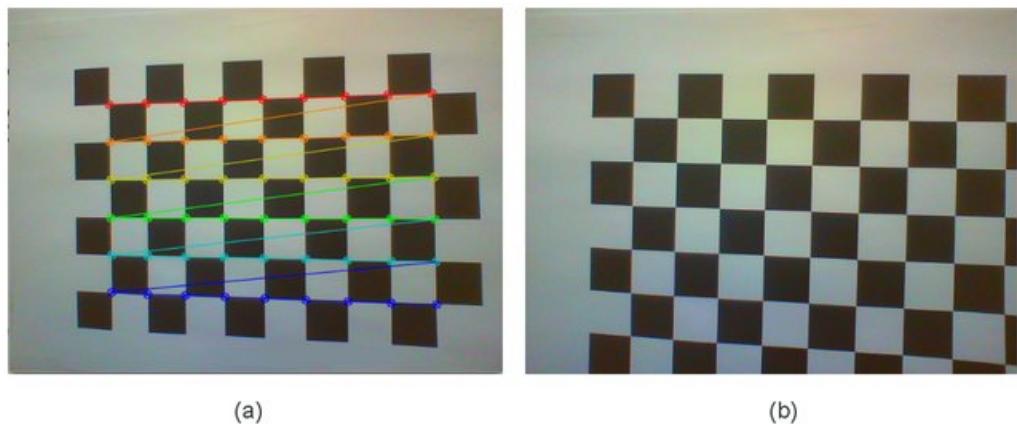
Preto sa vybrali značky ArUco, na základe ktorých možno merať relatívny posun kamery. Problémom pri nich je, že údaje o polohe možno získať len vtedy, keď je značka jasne viditeľná v obraze kamery. Tento problém sa však dá vyriešiť správnym označením testovacej plochy. Na umiestnenie značiek by sa mali použiť vhodné pravidlá, ktoré boli sformulované v časti 2.2.4, a tak možno určiť polohu kamery.

3.4.1 Používanie značkovačov aruco

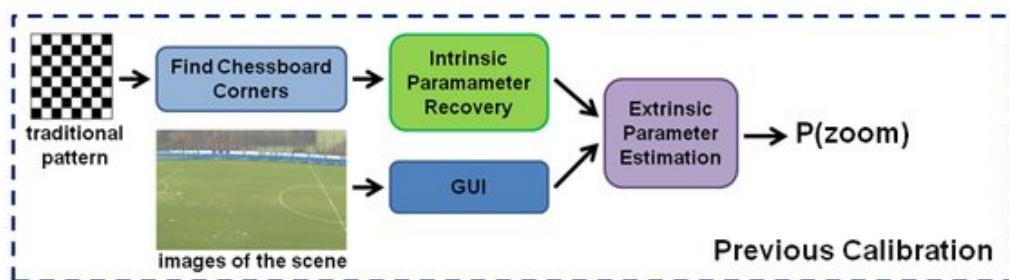
Funkcie potrebné na generovanie značiek ArUco nájdete v balíku prídavných modulov OpenCV-Contrib. Ich umiestnenie si vyžaduje kalibráciu kamery, ktorú možno vykonať aj pomocou kalibračného algoritmu v OpenCV na základe štúdie Z. Zhang [16] odhadom parametrov kamery.

Na kalibráciu potrebujete šachovnicu s dĺžkami strán, číslami riadkov a stĺpcov jej políčok. Na základe týchto údajov algoritmus vygeneruje teoretickú mapu vnútorných rohových bodov štvorcov a potom na základe vzoriek kamery vypočíta vnútorné a vonkajšie vlastnosti kamery v transformačných maticiach. Matica fotoaparátu obsahuje päť vnútorných vlastností fotoaparátu vrátane ohniskovej vzdialenosťi,omeru strán obrazového snímača a hlavného bodu dierkovej komory modulu. Získava sa aj matica skreslenia objektívu spojená s nelineárnymi vnútornými vlastnosťami, ktorú možno použiť na odstránenie súdkovitého a poduškovitého skreslenia na okrajoch obrazu. Vonkajšie (vonkajšie) vlastnosti kamery sú tiež opísané transformačnou

maticou, ktorá realizuje prevod zo súradníc reálneho 3D priestoru na 3D súradnice kamery. Matica aplikuje homogénnu transformáciu: posun do stredu (počiatku) snímača kamery o vektor T a rotáciu o maticu rotácie R .



Obrázok 3 – 4 Tradičná kalibrácia pomocou vzoru šachovnice. a) Rutiny OpenCV dokážu odhalit všetky rohy šachovnice. b) Rohy nemožno priradiť k príslušným bodom na vzore šachovnice, ak vzor nie je úplne zachytený.



Obrázok 3 – 5 Schéma procesu kalibrácie.

Teoreticky by na správnu kalibráciu mali stačiť dve vzorky, ale na dosiahnutie presnejších údajov pracujeme s 20 súbormi údajov. Fotografia na obrázku 3-4 bola zhotovená počas procesu kalibrácie. Výsledné matice sa ukladajú do súboru s názvom camcalib.npz a čítajú sa z neho, aby sa pred každým použitím nemusela kalibrovať zabudovaná kamera dronu.

Jedným z najčastejších zdrojov chýb pri meraní kamerou je súdkovité skreslenie, ktoré možno odstrániť pomocou matice skreslenia získanej počas kalibrácie pomocou

vstavanej funkcie OpenCV cv2.undistort(). Tým sa zníži efekt rybieho oka a do určitej miery sa zmenší zorné pole, ale získajú sa presnejšie hodnoty na okrajoch obrazu pre následné umiestnenie pomocou značiek ArUco.

3.4.2 Pravidlá umiestňovania značiek

Pri používaní značkovačov Aruco na určovanie polohy dronov je potrebné dodržiavať niekoľko dôležitých pravidiel, aby sa zabezpečili presné a spoľahlivé výsledky. Tu je niekoľko pokynov pre umiestnenie značiek [9]:

1. **Dostatočný počet značiek:** Na presné sledovanie polohy a orientácie dronu je dôležité použiť dostatočný počet značiek. To pomôže zabezpečiť, aby boli značky vždy viditeľné z pohľadu dronu, aj keď sa dron rýchlo pohybuje.
2. **Značky umiestnite do mriežky:** Umiestnenie značiek do pravidelnej mriežky pomôže zabezpečiť ich rovnomerné rozloženie a viditeľnosť z viacerých uhlov. To môže tiež pomôcť znížiť pravdepodobnosť zákrytov alebo iných problémov, ktoré by mohli narušiť sledovanie.
3. **Vyber veľkosti značiek:** Veľkosť použitých značiek bude závisieť od vzdialenosťi, z ktorej sa na ne bude pozerať. Napríklad, ak bude dron lietať relatívne blízko značiek, môžete použiť menšie značky. Ak však bude dron vo väčšej vzdialenosťi, možno budete musieť použiť väčšie značky, aby ste zabezpečili viditeľnosť.
4. **Reflexný alebo priehľadný povrch:** Reflexné alebo priehľadné povrhy môžu narušiť detekciu markerov tým, že spôsobujú odrazy alebo lomy, ktoré skresľujú vzhľad markera. Aby ste sa vyhli tomuto problému, vyberte si miesto na umiestnenie značky, ktoré je bez reflexných alebo priehľadných povrchov.
5. V zornom poli dronu musí byť vždy prítomný aspoň jeden marker, aby sa zabezpečilo, že dron môže nepretržite sledovať svoju polohu a orientáciu.

6. Keď sa markery časom zistia a stratia, systém použije polohu a orientáciu predchádzajúceho markera na odhad aktuálnej polohy a trajektórie dronu, kým sa nezistí nový marker.

3.5 Prvé testy polohovania pomocou značiek

Na testovanie sa použili štyri značky 7x7 ArUco s dĺžkou strany 0,0957 m z knižnice DICT-7x7-100. K dispozícii sú aj značky s plochami 4x4, 5x5 a 6x6, ale vyššie rozlíšenie knižnice 7x7 umožňuje presnejšie určovanie polohy. V každej knižnici je k dispozícii až 1024 rôznych značiek, čo by malo stačiť na poskytnutie informácií potrebných na riadenie dronu.

Odporuča sa používať viac ako jeden marker, pretože systém dokáže spriemerovať z viacerých hodnôt, takže spriemerovaním možno získať presnejšie výsledky, jednoducho odfiltrovaním akýchkolvek odľahlých hodnôt. Okrem toho, ak je v zornom poli dronu niekoľko značiek, je menší problém, ak sa jedna z nich stratí, pretože stále môže čítať hodnoty z ostatných. V prvom experimente boli značky umiestnené kolmo na kameru dronu, vertikálne k stene, ako je znázornené na fotografii na obrázku 10. Cieľom tohto testu bolo zistiť, ako dobre dokáže softvér ukladať údaje, ktoré sa približujú realite.

Súbor bodov vľavo na obrázku 11 ukazuje, že súbor meracích bodov bol úspešne sledovaný hned po opísaní štvorca dronom a dokonca aj po zakreslení jednej z jeho uhlopriečok, a vpravo ukazuje, že aj zmena výšky bola pomocou značiek zistená s vynikajúcou presnosťou. Priesečník čiernych osí je počiatok pohybu dronu. Body na obrázku vyznačené červenou farbou sú skutočné hodnoty a zelená krivka je B-spline krivka prispôsobená bodu nastavenému funkciou `splprep()` v podadresári `interpolate` balíka Scipy dostupného v programovacom jazyku Python. Parameter spresnenia funkcie bol po niekolkých experimentoch na základe dokumentácie [17] zvolený na hodnotu $s=0,1$, čo al-ready dostatočne aproximuje množinu bodov odfiltrovaním

zašumených bodov. Nakoniec sme vo finálnej verzii programu použili na filtrovanie dátových bodov Kalmanov filter namiesto metódy B-spline interpolácie, pričom sme mysleli na budúcu implementáciu v reálnom čase a adaptívnejšie správanie Kalmanovho filtra.

Jediná korekcia, ktorú bolo potrebné aplikovať na experimentálnu sadu bodov, bolo negatívne otočenie bodov o $10,5^\circ$ okolo osi x, inak by sa Z-koordináty bodov zväčsili, keď by sa približovali k značkám, a to aj pri konštantnej výške. Je to spôsobené uhlom kamery od vodnej hladiny, pričom pri priblížení k markerom sa zistila vyššia poloha. (Toto bude neskôr zbytočné kvôli korekcii uhlového natočenia značiek.)

V tomto nastavení bude dron poskytovať skutočné hodnoty posunutia len vtedy, ak je rovina kamery (okrem uhla poklesu) rovnobežná s rovinou značiek. Ak sa poloha od tejto odchýli, dron bude udávať nesprávne hodnoty. Hodnoty posunutia sa musia odhadnúť s prihliadnutím na uhly natočenia, inak nemôžeme previesť hodnoty vektorov translácie značiek ArUco do súradnicového systému prvej videnej značky, ktorý chceme použiť ako globálny súradnicový systém. Hodnoty vektorov zadané funkciou `cv2.aruco.estimatePoseSingleMarkers()` balíka OpenCV sú zadané v súradnicovom systéme kamery, ktoré je potom potrebné previesť do súradnicového systému markerov.

3.5.1 Vyhodnotenie testu a očakávania od systému

Na základe prvých testov sa oplatí ďalej skúmať túto možnosť určovania polohy, pretože dokáže vypočítať polohu dronu v reálnom čase s relatívne malým počtom transformácií. Bohužiaľ, nemôžeme využiť grafickú akceleráciu balíka CUDA, ktorú poskytuje spoločnosť Nvidia, pretože funkcie balíka OpenCV-Python neboli všetky pridané do knižnice `cv2.cuda` v jazyku Python. Spomalí to spracovanie obrazu, ale aj tak spôsobí väčšie oneskorenie operácie kvôli 1,5...2 sekundovému oneskoreniu kamerového obrazu prenášaného dronom.

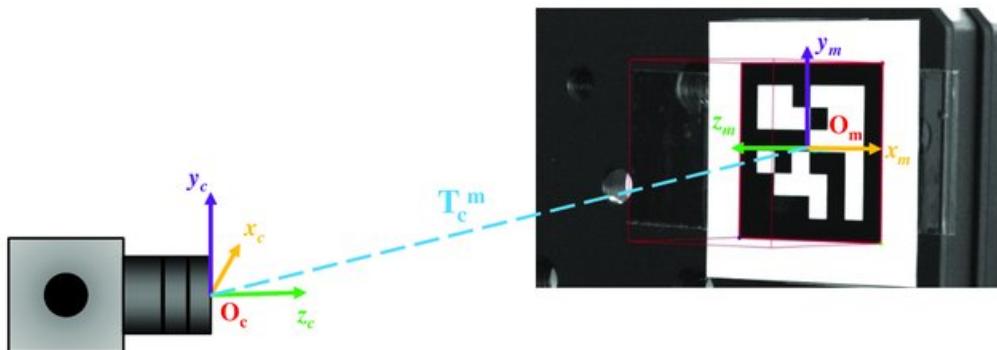
Očakáva sa, že chyba merania metódy bude približne 0,05 m pri zväčšenej veľkosti značiek a vhodnom počte značiek. V dôsledku konverzií medzi súradnicovými systémami markerov však môžeme očakávať aj chybu driftu, ktorá sa zvyšuje so vzdialenosťou od markera reprezentujúceho globálnu súradnicu v dôsledku nepresnosti transformácií medzi súradnicovými systémami.

3.6 Princíp, spresnenie a robustnosť polohovania

Počas vývojovej a testovacej fázy kamerového programu sme namiesto palubnej kamery dronu Tello použili webovú kameru. Dôvodom bola predovšetkým obmedzená životnosť batérie dronu, ktorá by stažila testovanie a zdokonalovanie programu na dlhší čas. Namiesto toho sme použili webovú kameru namontovanú na stabilnej platforme, ktorá simulovala pohľad kamery dronu, a testovali sme výkon programu v rôznych svetelných podmienkach, vzdialostiach a orientáciách. Keď sme boli s výkonom programu spokojní, prenesli sme ho do dronu Tello a podľa potreby sme vykonali drobné úpravy. Program bol napísaný s použitím funkcií poskytovaných v balíku OpenCV ArUco, napríklad:

1. **Detekcia a rozpoznávanie značiek:** Na detekciu a rozpoznanie značiek Aruco vo videokanáli dronu používame funkciu cv2.detectMarkers(). Táto funkcia prijíma obraz, slovník Aruco a parametre na detekciu a vracia zistené značky a ich príslušné ID.
2. **Transformácie medzi súradnicovými systémami značky a kamery:** Po zistení a rozpoznaní značiek musíme transformovať ich polohu v obraze (v pixelových súradničiach) na ich polohu v súradnicovom systéme kamery (v milimetroch). Toto sa vykoná pomocou funkcie cv2.aruco.estimatePoseSingleMarkers(), ktorá prijme detektované značky, veľkosť značiek, maticu kamery a koeficienty skreslenia a vráti vektory rotácie a translácie, ktoré predstavujú polohu každej značky v súradnicovom systéme kamery.

3. **Spresnenie pozícii značiek:** Kedže zistené polohy značiek nie sú vždy presné, musíme ich spresniť pomocou subpixelovej presnosti. To sa vykonáva pomocou funkcie cv2.cornerSubPix(), ktorá prijíma obraz, zistené rohy značky a parametre pre subpixelové spresnenie a vracia spresnené rohové pozície.
4. **Robustnosť polohovania:** Aby sme zabezpečili robustnosť a presnosť polohy dronu, používame v obraze viacero značiek a vykonávame priemerovanie ich polôh. To pomáha znížiť vplyv šumu a chýb pri detekcii a rozpoznávaní značiek.
5. **Transformácia perspektív a bodu (PnP):** Funkcia cv2.aruco.solvePnP() sa používa na odhad pózy (polohy a orientácie) značky v 3D priestore. Táto funkcia prijíma ako vstup 2D súradnice značky na obrázku, 3D súradnice značky v reálnom svete a maticu kamery, ktorá obsahuje vlastné parametre kamery (ohniskovú vzdialenosť, hlavný bod atď.) [18].



Obrázok 3 – 6 Súradnicový systém kamery a značky interpretovaný programom OpenCV.

3.6.1 Perspektívna projekcia, transformácia pnp

Perspektívna projekcia je matematická metóda, ktorá sa používa na transformáciu 3D objektov na 2D obrazovú rovinu. Tento vzťah sa nazýva perspektívna projekcia a je založený na princípe podobnosti trojuholníkov. Matematicky môžeme tento proces

zapísť ako [18]:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (3.1)$$

kde u a v sú pixely v 2D obrazovke, (X, Y, Z) sú súradnice bodu v 3D priestore, f_x a f_y predstavujú ohniskovú vzdialenosť kamery v osiach x a y v pixely, c_x a c_y predstavujú súradnice stredu obrazovky v pixely, r_{11} až r_{33} sú prvky rotácie a t_1 až t_3 predstavujú posun kamery v priestore.

Tento proces zahŕňa vytvorenie modelu kamery, ktorý definuje polohu a orientáciu virtuálnej kamery v 3D priestore. Tento model kamery sa potom použije na premietnutie 3D súradníc objektu do 2D roviny.

Na výpočet transformácie medzi 3D svetovými súradnicami objektu a jeho 2D obrazovými súradnicami sa používa algoritmus PnP (Perspective-n-Point). Túto transformáciu možno použiť na určenie polohy a orientácie objektu v 3D priestore vzhľadom na kameru.

Algoritmus PnP vyžaduje nasledujúce vstupy:

1. Súbor 3D bodov objektu vyjadrených v súradnicovom systéme vzhľadom na kameru.
2. Súbor 2D obrazových bodov, ktoré zodpovedajú projekciu 3D bodov objektu na 2D obrazovú rovinu.
3. Vlastná matica kamery, ktorá opisuje vnútorné parametre kamery, ako je ohnisková vzdialenosť a hlavný bod.
4. Voliteľne vonkajšia matica kamery, ktorá opisuje polohu a orientáciu kamery v 3D priestore.

5. Algoritmus PnP vypočíta rotáciu a transláciu objektu vzhľadom na kameru pomocou nelineárnej optimalizačnej metódy. Algoritmus minimalizuje chybu reprojekcie, čo je rozdiel medzi projektovanými 3D bodmi a zodpovedajúcimi 2D obrazovými bodmi.

Algoritmus PnP možno riešiť rôznymi metódami vrátane metódy priamej lineárnej transformácie (DLT) a Levenberg-Marquardtovho algoritmu. Metóda DLT zahŕňa riešenie lineárnej sústavy rovníc, zatiaľ čo Levenberg-Marquardtov algoritmus je nelineárna optimalizačná metóda.

Algoritmus PnP je základným nástrojom počítačového videnia a používa sa v mnohých aplikáciách vrátane sledovania objektov, rozšírenej reality a lokalizácie robotov.

Algoritmus PnP je implementovaný vo funkcií OpenCV `cv2.solvePnP()`, ktorá prijíma vyššie opísané vstupy a vracia vektory rotácie a translácie objektu vzhľadom na kameru. Tieto vektory možno použiť na transformáciu súradníc objektu do súradnicového systému kamery alebo naopak pomocou jednoduchých maticových operácií [19].

Vzorec pre transformáciu PnP je nasledujúci:

$$rvec, tvec = cv2.solvePnP(objectPoints, imagePoints, cameraMatrix, distCoeffs) \quad (3.2)$$

kde:

1. `objectPoints` je pole 3D bodov v súradnicovom systéme objektu.
2. `imagePoints` je pole zodpovedajúcich 2D obrazových bodov.
3. `cameraMatrix` je vlastná matica kamery.
4. `distCoeffs` sú koeficienty skreslenia kamery.
5. `rvec` je vektor natočenia objektu vzhľadom na kameru.

6. *tvec* je vektor translácie objektu vzhľadom na kameru.

Vektor rotácie *rvec* možno previesť na maticu rotácie pomocou funkcie cv2.Rodrigues(). Translačný vektor *tvec* je vyjadrený v súradnicovom systéme kamery a možno ho použiť na transformáciu súradníc objektu do súradnicového systému kamery pomocou jednoduchej maticovej operácie [19]:

$$\text{objectPointscam} = \mathbf{R} \cdot \text{objectPoints}^\top + \text{tvec} \quad (3.3)$$

Kde **objectPointscam** je transformované pole 3D bodov objektu v súradnicovom systéme kamery, **R** je matica rotácie, **objectPoints** je pole 3D bodov objektu v súradnicovom systéme objektu, **tvec** je vektor translácie objektu vzhľadom na kameru a \cdot predstavuje násobenie matíc.

3.6.2 Konvencie merania

Pri riešení problémov počítačového videnia je dôležité stanoviť súbor konvencí merania. Tieto konvencie špecifikujú súradnicový systém používaný na reprezentáciu polôh a orientácií objektov v 3D priestore, ako aj jednotky používané na ich meranie. V tejto časti opíšeme konvencie používané v počítačovom videní, ktoré sú založené na pravotočivom súradnicovom systéme.

Pravotočivý súradnicový systém. Pravotočivý súradnicový systém je najčastejšie používaným systémom v počítačovom videní. Je to trojrozmerný karteziánsky súradnicový systém, v ktorom kladné osi x, y a z smerujú doprava, nahor a k pozorovateľovi. Orientáciu súradnicového systému možno reprezentovať pomocou matice rotácie, ktorá transformuje body zo svetového súradnicového systému do súradnicového systému kamery.

Jednotky merania. Merné jednotky používané v počítačovom videní sú zvyčajne milimetre alebo metre. Tieto jednotky sa používajú na reprezentáciu vzdialenosí, napríklad vzdialenosí medzi dvoma bodmi v 3D priestore alebo ohniskovej vzdialnosti kamery. Výber jednotiek závisí od rozsahu riešeného problému.

Matica kamery. Matica kamery je matica 3×4 , ktorá predstavuje vnútorné a vonkajšie parametre kamery. Vnútorné parametre opisujú vlastnosti kamery, napríklad ohniskovú vzdialenosť a polohu hlavného bodu, zatiaľ čo vonkajšie parametre opisujú orientáciu a polohu kamery v 3D priestore. Maticu kamery možno zapísať ako:

$$\mathbf{P} = \begin{bmatrix} f_x & 0 & c_x & t_x \\ 0 & f_y & c_y & t_y \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad (3.4)$$

kde f_x a f_y sú ohniskové vzdialosti v smere x a y, c_x a c_y sú súradnice hlavného bodu v obraze a t_x a t_y sú translačné parametre, ktoré opisujú polohu kamery vo svetovom súradnicovom systéme.

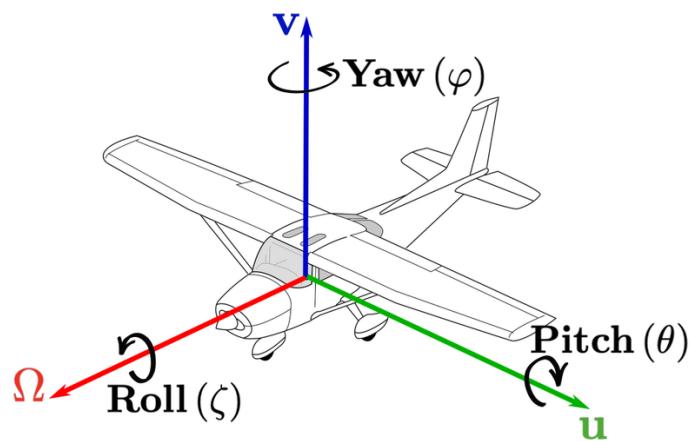
Projekčná matica. Projekčná matica je matica 3×4 , ktorá mapuje 3D body vo svetovom súradnicovom systéme na 2D body v súradnicovom systéme obrazu. Vypočítava sa ako súčin matice kamery a matice rotácie, ktorá transformuje body zo svetového súradnicového systému do súradnicového systému kamery:

$$\mathbf{P}' = \mathbf{P} \cdot [\mathbf{R}|\mathbf{t}], \quad (3.5)$$

kde \mathbf{R} je matica rotácie a \mathbf{t} je vektor translácie, ktorý opisuje polohu kamery vo svetovom súradnicovom systéme.

Homogénny súradnicový systém. Homogénny súradnicový systém je matematický rámc, ktorý nám umožňuje reprezentovať body v 3D priestore ako 4D vektory. Je to užitočné, pretože nám to umožňuje vykonávať transformácie, ako je translácia a rotácia, pomocou násobenia matíc. 3D bod $\mathbf{X} = [X, Y, Z]$ vo svetovom súradnicovom systéme možno reprezentovať v homogénnych súradniciach ako $\mathbf{X}_h = [X, Y, Z, 1]$. Podobne bod $\mathbf{x} = [u, v]$ v súradnicovom systéme obrazu možno reprezentovať v homogénnych súradniciach ako $\mathbf{x}_h = [u, v, 1]$.

Kedže neexistuje všeobecná dohoda o konvencii pre Eulerove uhly, musíme definovať používanú konvenciu. V tomto prípade ukladáme uhly v už spomínanom poradí podľa záporných hodnôt získaných z dronu SDK. Tie sú totožné s konvenciou uhlov používanou pre lietadlá podľa letovej normy DIN 9300-2 (obrázok 3-6). Presnejšou definičnou skupinou pre konvenciu vybraných uhlov je vlastná konvencia Tait-Bryanových uhlov z-y'-x'', známa aj ako námorné alebo Cardanove uhly [20].



Obrázok 3 – 7 Hlavné nápravy lietadla podľa normy DIN 9300.

3.6.3 Konvencie používané v opencv

OpenCV používa špecifický súbor konvencií na reprezentáciu bodov, rotácií a translácií [19]. Tieto konvencie je dôležité pochopíť pri práci s knižnicou. Tu sú niektoré z hlavných konvencií používaných v OpenCV:

Body sú reprezentované ako súradnice (x, y) s počiatkom (0, 0) v ľavom hornom rohu obrazu.

Rotácie sú reprezentované pomocou Rodriguesovho rotačného vzorca, ktorý konvertuje 3D rotačný vektor na 3x3 rotačnú maticu. Vektor rotácie je definovaný ako (rx, ry, rz), kde rx, ry a rz predstavujú uhly rotácie okolo osí x, y a z.

Translácie sú reprezentované ako 3D vektory (t_x , t_y , t_z).

Vlastné parametre kamery sú reprezentované pomocou nasledujúcej matice:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.6)$$

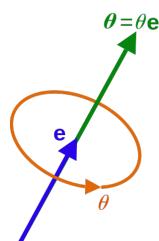
kde f_x a f_y sú ohniskové vzdialosti kamery v smere x a y a c_x a c_y sú súradnice hlavného bodu (bod, v ktorom optická os pretína rovinu obrazu).

Vonkajšie parametre kamery sú reprezentované pomocou matice 3×4 , ktorá kombinuje rotáciu a transláciu:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \quad (3.7)$$

kde ľavý horný stĺpec matice 3×3 predstavuje maticu rotácie a pravý stĺpec predstavuje vektor translácie.

Medzi hodnotami vrátenými funkciou `estimatePoseSingleMarkers()` sa vektor translácie interpretuje v metroch a vektor rotácie používa reprezentáciu osového uhla alebo Rodriguesovu reprezentáciu, čo je najkompaktnejšia forma uloženia matice rotácie. Funkcia `Rodrigues()` v OpenCV dokáže vypočítať maticu rotácie z vektorov rotácie zadaných ako riadkové vektory podľa rovnice (3.8.) a dokáže určiť aj jej inverziu, ak je vstupom matica rotácie.



Obrázok 3 – 8 Interpretácia vektora natočenia Rodriguesovej osi a uhla.

3.6.4 Prevod polohy kamery do globálneho súradnicového systému

Na prevod polohy kamery (ktorá sa nachádza v počiatku súradnicového systému kamery) do globálneho súradnicového systému musíme použiť vektory rotácie a translácie získané z funkcie solvePnP v OpenCV.

Vektor rotácie (rvec) udáva rotáciu súradnicového systému kamery vzhľadom na súradnicový systém značky a vektor translácie (tvec) udáva polohu značky v súradnicovom systéme kamery.

Na prevod polohy kamery do globálneho súradnicového systému musíme postupovať podľa týchto krokov:

- 1. Vypočítame maticu rotácie \mathbf{R}** z vektora rotácie rvec pomocou Rodriguesovho vzorca:

$$R = \begin{bmatrix} \cos \theta + u_x^2(1 - \cos \theta) & u_x u_y (1 - \cos \theta) - u_z \sin \theta & u_x u_z (1 - \cos \theta) + u_y \sin \theta \\ u_y u_x (1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2(1 - \cos \theta) & u_y u_z (1 - \cos \theta) - u_x \sin \theta \\ u_z u_x (1 - \cos \theta) - u_y \sin \theta & u_z u_y (1 - \cos \theta) + u_x \sin \theta & \cos \theta + u_z^2(1 - \cos \theta) \end{bmatrix} \quad (3.8)$$

kde θ je velkosť vektora rotácie a u_x , u_y a u_z sú zložky jednotkového vektora v smere vektora rotácie.

- 2. Vypočítame inverznú hodnotu matice rotácie \mathbf{R} ,** ktorá predstavuje rotáciu súradnicového systému značky vzhľadom na súradnicový systém kamery.
- 3. Vypočítame inverzný vektor translácie \mathbf{tvec} ,** ktorý predstavuje polohu kamery v súradnicovom systéme značky.
- 4. Vypočítame súčin inverznej matice rotácie a inverzného translačného vektora:**

$$-R^{-1}t_{vec} \quad (3.9)$$

Takto získame polohu kamery v súradnicovom systéme značky.

5. Prevedme polohu kamery v súradnicovom systéme markera do globálneho súradnicového systému pripočítaním polohy markera v globálnom súradnicovom systéme:

$$P_{global} = P_{značka} + R_{marker}^{-1}(-R^{-1}t_{vec}) \quad (3.10)$$

kde $P_{značka}$ je poloha značky v globálnom súradnicovom systéme a $R_{značka}$ je rotačná matica, ktorá definuje orientáciu značky v globálnom súradnicovom systéme.

Výsledný vektor P_{global} je poloha kamery v globálnom súradnicovom systéme.

V súhrne sú kroky na prevod polohy kamery do globálneho súradnicového systému nasledovné:

Výpočet matice rotácie R z vektora rotácie rvec pomocou Rodriguesovho vzorca. Výpočet inverznej matice rotácie R na získanie matice rotácie reprezentujúcej súradnicový systém markera vzhľadom na súradnicový systém kamery. Vypočítame inverzný vektor translácie tvec

3.7 Webové sokety s komunikáciou s dronom

Webové sokety sú populárny mechanizmom na komunikáciu v reálnom čase medzi klientom a serverom cez web. V kontexte riadenia dronov sa webové sokety môžu použiť na vytvorenie trvalého obojsmerného kanála medzi pozemnou stanicou (klientom) a dronom (serverom), ktorý umožňuje výmenu príkazov, telemetrických údajov a videoprenosov v reálnom čase.

3.7.1 Architektúra

Typická architektúra komunikačného systému dronu založeného na webových zásuvkách pozostáva z troch hlavných komponentov: pozemnej stanice, dronu a web-socket servera. Pozemná stanica je zodpovedná za generovanie používateľských príkazov a prijímanie telemetrických údajov a video prúdov z dronu. Dron je zodpovedný za vykonávanie príkazov a zachytávanie telemetrických údajov a video prúdov. Webový socketový server funguje ako middleware medzi pozemnou stanicou a dronom, spracováva komunikačný protokol a preposiela správy v reálnom čase.

3.7.2 Komunikačný protokol

Komunikačný protokol pre systém dronu založený na webo-socket pozostáva z dvoch typov správ: príkazových správ a telemetrických správ.

Príkazové správy. Príkazové správy sa posielajú z pozemnej stanice do dronu a pozostávajú z pokynov pre dron, aby vykonal určitú činnosť, napríklad vzlietol, pristál, pohol sa dopredu alebo odbočil dolava. Formát príkazových správ sa môže lísiť v závislosti od modelu dronu a špecifických funkcií, ktoré sa používajú, ale bežný formát je JSON (JavaScript Object Notation). Napríklad príkazová správa o vzlete vo formáte JSON môže vyzerať takto:

```
{  
  "command": "takeoff",  
  "args": []  
}
```

Telemetrické správy. Príkazové správy sa posielajú z pozemnej stanice do dronu a pozostávajú z pokynov pre dron, aby vykonal určitú činnosť, napríklad vzlietol, pristál, pohol sa dopredu alebo odbočil dolava. Formát príkazových správ sa môže lísiť v závislosti od modelu dronu a špecifických funkcií, ktoré sa používajú. Telemetrické správy sa odosielajú z dronu do pozemnej stanice a pozostávajú z údajov snímačov, ako sú

súradnice GPS, nadmorská výška, úroveň nabitia batérie a údaje z kamery. Formát telemetrických správ sa môže lísiť aj v závislosti od modelu dronu a konkrétnych použitých snímačov, ale JSON je opäť bežný formát. Napríklad telemetrická správa Tello drona o svojom *state* vo formáte JSON môže vyzerať takto:

```
{  
    "pitch": 0,  
    "roll": 0,  
    "yaw": 0,  
    "vgx": 0,  
    "vgy": 0,  
    "vgz": 0,  
    "templ": 62,  
    "temph": 65,  
    "tof": 10,  
    "h": 0,  
    "bat": 58,  
    "baro": 40.39,  
    "time": 0,  
    "agx": 7.00,  
    "agy": 19.00,  
    "agz": -999.00  
}
```

Tento objekt obsahuje rôzne informácie o stave dronu vrátane uhlov náklonu, natočenia a vychýlenia, rýchlosťi pozdĺž osí x, y a z, teploty, vzdialenosťi počas letu, úrovne nabitia batérie, barometrického tlaku a zrýchlenia pozdĺž osí x, y a z.

3.7.3 Implementácia

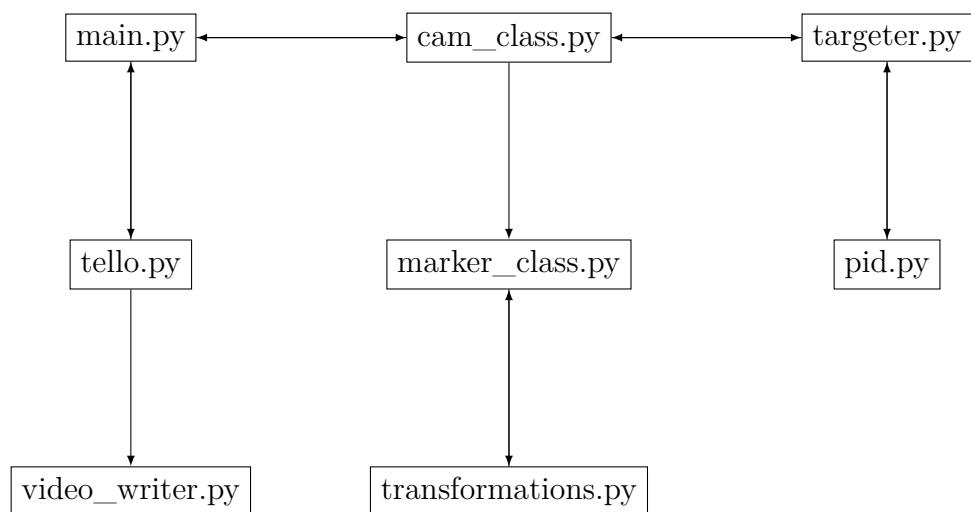
Na implementáciu komunikačného systému dronu založeného na web-sockets možno postupovať podľa nasledujúcich krokov:

- Nastaviť web-socket server, ktorý dokáže spracovať prichádzajúce spojenia z pozemnej stanice a dronu a preposielat správy medzi nimi.
- Implementovať skript na strane klienta na pozemnej stanici, ktorý dokáže nadviazať spojenie webového socketu so serverom, odosielat príkazové správy do dronu a prijímať telemetrické správy z dronu.
- Implementovať skript na strane servera na dron, ktorý môže vytvoriť spojenie webového socketu so serverom, prijímať príkazové správy z pozemnej stanice a posielat telemetrické správy pozemnej stanici.
- Integrovať skripty na strane klienta a servera s riadiacim softvérom dronu a používateľským rozhraním na pozemnej stanici.

4 Implementácia projektu

V tejto časti podrobne opíšeme realizáciu nášho projektu. Opíšeme architektúru systému a účel jednotlivých modulov. Poskytneme aj prehľad štruktúry kódu a funkčnosti jednotlivých súborov. Okrem toho vysvetlíme, ako sme do nášho projektu integrovali rôzne technológie a knižnice, aby sme dosiahli naše ciele. Nakoniec rozoberieme všetky problémy, s ktorými sme sa stretli počas procesu implementácie, a ako sme ich riešili.

4.1 Prehľad štruktúry programov



Description:

main.py - Contains the pygame GUI and the camera image display. Uses the different keys on the keyboard to switch functions or control the drone.

tello.py - Contains functions for communication with the drone and data readout.

video_writer.py - Running on a separate thread, it captures video during the auto control and then outputs it.

cam_class.py - Performs the image processing tasks (calibration, marker recog-

nition), and the drone speed data is queued from here. Passes the read marker positions to the functions that process them.

targeter.py - Reads the functions corresponding to the marker sequence numbers from the specified CSV file and returns the control baseline vector.

pid.py - Contains a class implementing a numerical PID controller, it calculates the control rate values.

marker_class.py - A class that stores all the data for markers. During data collection, it counts and finally writes the collected positions to an NPZ file.

transformations.py - Implements coordinate system transformations and contains functions for rotation vector conversions.

4.2 Programy, ktoré ovládajú dron

Dron Tello sa ovláda pomocou kombinácie softvérových programov, ktoré s ním komunikujú prostredníctvom pripojenia Wi-Fi. Na nízkoúrovňovú komunikáciu medzi dronom a riadiacim počítačom sa používa knižnica Tello Python, tello.py.

Knižnica tello.py poskytuje množstvo funkcií, ktoré možno použiť na odosielanie príkazov do dronu a prijímanie údajov z neho. Medzi tieto funkcie patria *takeoff()*, *land()*, *move_left()*, *move_right()*, *move_forward()*, *move_backward()*, *rotate_clockwise()*, *rotate_counter_clockwise()* a mnohé ďalšie. Knižnica obsahuje aj funkcie na získavanie údajov z dronu, ako je jeho aktuálna výška, úroveň nabitia batérie a čas letu.

Na ovládanie dronu Tello z webovej aplikácie program využíva spojenie "web-sockets" na komunikáciu so skriptom Python na strane servera. Tento skript funguje ako most medzi webovou aplikáciou a dronom Tello a prenáša medzi nimi príkazy a údaje.

Skript na strane servera používa knižnicu tello.py na odosielanie príkazov do dronu

a prijímanie údajov z neho. Keď používateľ zadá príkaz vo webovej aplikácii, príkaz sa odošle na server prostredníctvom spojenia "web-sockets". Server potom použije príslušnú funkciu v knižnici tello.py na odoslanie príkazu dronu. Po vykonaní príkazu môže server získať všetky relevantné údaje z dronu a poslať ich späť do webovej aplikácie na zobrazenie.

Celkovo kombinácia tello.py a skriptu na strane servera poskytuje výkonnú platformu na ovládanie dronu Tello z webovej aplikácie. Vďaka možnosti odosielat širokú škálu príkazov a získavať podrobné údaje z dronu.

4.3 Programy, ktoré spracúvajú obrázky

S riadením dronov úzko súvisí, ale viac sa zaoberá spracovaním obrazu, program cam_class.py, ktorého hlavná funkcia je zodpovedná za rozpoznávanie značiek ArUco. ArUco markery sa zisťujú pomocou vstavanej funkcie OpenCV cv2.aruco.detectMarkers(). Tá nájde kódy v načítanom zväzku značiek po adaptívnej segmentácii aplikovanej na obraz v odtieňoch sivej a vráti ich rohové body v pixeloch a ich poradové číslo. Na výpočet skutočnej trojrozmernej polohy značiek na základe matíc kamery sa používa už spomínaná transformácia PnP. Výsledné polohy s ich poradovým číslom značky sa môžu odoslať na ďalšie spracovanie (časť 4.4) alebo použiť na automatickú navigáciu dronu.

Program je určený na navigáciu dronu Tello pomocou značiek umiestnených na zemi. To sa dosiahne použitím knižnice OpenCV na detekciu značiek vo videozázname z kamery dronu.

Code Listing 3 Inicializácia parametrov detektie značiek OpenCV

```
import cv2
import numpy as np
from djitellopy import Tello
```

```

# Initialize Tello drone
tello = Tello()
tello.connect()
tello.streamon()

# Initialize OpenCV marker detection parameters
aruco_dict = cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_250)
aruco_params = cv2.aruco.DetectorParameters_create()

# Set up video stream
stream = tello.get_video_stream()

```

Program najprv inicializuje kameru a nastaví tok videa. Potom pomocou slučky nepretržite zachytáva snímky z videoprenosu a spracováva ich. Pre každú snímku program používa vstavané funkcie OpenCV na detekciu značiek a odhad ich polohy v 3D priestore.

Code Listing 4 Detekcia markera a vypočítanie polohy

```

while True:
    # Capture frame from video stream
    frame = stream.read()

    # Detect markers in frame
    corners, ids, rejected = cv2.aruco.detectMarkers(frame,
                                                       aruco_dict,
                                                       parameters=aruco_params)

    # Estimate marker position in 3D space
    rvecs, tvecs, _ = cv2.aruco.estimateSingleMarkers(corners,
                                                       0.05,
                                                       camera_matrix,
                                                       dist_coeffs)

    # Process marker positions
    if ids is not None:

```

```
# TODO: Implement control algorithm
pass

# Display frame
cv2.imshow('Frame', frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

Po odhadnutí polohy značiek program pomocou proporcionálneho riadiaceho algoritmu upravuje odklon, sklon a výšku dronu na základe polohy značiek vzhľadom na stred snímky. Algoritmus vypočítava chybu medzi stredom rámu a polohou značiek a upraví orientáciu a výšku dronu tak, aby sa táto chyba minimalizovala.

Program obsahuje aj niekoľko bezpečnostných funkcií, ktoré zabráňajú kolízii dronu s prekážkami alebo vyleteniu mimo dosahu. Medzi tieto funkcie patrí nastavenie maximálnej výšky a doletu dronu, ako aj detekcia prekážok v dráhe dronu a automatické nastavenie jeho trajektórie tak, aby sa im vyhol.

Na ďalšie zlepšenie presnosti a spoľahlivosti detekcie a sledovania značiek program obsahuje množstvo konfigurovateľných parametrov, ako je minimálna a maximálna veľkosť značky, prah detekcie a proporcionálne zisky riadenia. Tieto parametre sa dajú upraviť tak, aby sa optimalizoval výkon programu pre rôzne prostredia a svetelné podmienky.

Celkovo program poskytuje spoľahlivý a efektívny spôsob navigácie dronu Tello pomocou značkovačov. Použitím knižnice OpenCV na detekciu a sledovanie značiek v reálnom čase a implementáciou proporcionálneho riadiaceho algoritmu na úpravu orientácie a výšky dronu je program schopný navigovať dron s vysokou mierou presnosti a precíznosti.

4.4 Programy na spracovanie údajov

Spracovanie priestorových bodov a rotácií získaných z programov na spracovanie obrazu z kamery, t. j. zber údajov, vykonáva program marker_class.py. Potrebné transformácie, ktoré sú opísané vyššie, vykonávajú funkcie transformations.py.

4.4.1 Ukladanie údajov o značkách

Na spracovanie sa môžu odovzdať len údaje značiek, ktoré nemajú žiadny zo svojich vrcholov v pixelovom rámci obrazu. Definícia hrany je 2-2-2-2 % priesecník na všetkých štyroch stranách obrazu kamery. Značkovače umiestnené na okrajoch poskytujú po odstránení skreslenia jednako nepresnejšie údaje, jednak môžu spôsobiť neistotu označenia vonkajších okrajov 2D kódu značkovača počas detekcie značkovača, ako je to vidieť na obrázku 20.

V triede markers sú vlastnosti markerov, ktoré ste doteraz videli, uložené v rôznych zoznamoch, ako napríklad:

- **ids**: počet už použitých značiek
- **tvec_origin**: "markerorigos" v globálnom súradnicovom systéme
- **rvec_origin**: otočenie súradnicového systému markerov vzhľadom na globálny
- **dRot**: matica rotácie od daného markera ku globálnemu.
- **allow_use**: pomocná premenná, ktorá umožňuje použitie daného markera pri priemerovaní, akonáhle sa zozbiera dostatočný počet vzoriek, môže sa započítať
- **tvec_min, tvec_max**: pomocné premenné na filtrovanie výpočtu priemeru

Funkcia appendMarker() volaná počas detekcie značiek ArUco pridá nové značky

do triedy značiek. Za počiatok berie značku s číslom 1 a na základe jej Eulerovho uhlového natočenia okolo osi x určí, či ide o horizontálnu alebo vertikálnu značku. Ukladá tiež základné hodnoty uhlových rotácií z čítania stavov dronu, to sa stáva počiatkom Eulerových uhlov. Ukladá príslušnú jednu z dvoch orientácií do matice, ktorá bude relevantná pre indexovanie po spracovaní invertovaním smerov vektora translácie.

Ďalšie značky sa ukladajú vytvorením transformácií medzi súradnicovými systémami, ktoré sú implementované skriptom `getTransformations()` v súbore `transformations.py`. V prevádzke systém v súčasnosti pracuje s 12 platnými vzormi. Testy boli vykonané s vyššími limitmi platnosti, ale mnohokrát systém nestihol vykonať transformácie, kým sa značky nenachádzali v zornom poli dronu a nebol schopný vypočítať ďalšie. Preto sa zozbiera 12 vektorov posunu medzi dvoma značkami, pričom sa vyradia minimálne a maximálne normály a z 10 zostávajúcich hodnôt sa vytvorí priemer. Matice natočenia sa tiež získajú spriemerovaním 12 vzoriek bez filtrovania. Výsledné hodnoty sa uložia do zoznamov objektov triedy.

Ak už má značka vzorku na autorizáciu, možno ju spočítať pomocou funkcie v úryvku kódu 5. Program spočíta pozície načítané zo všetkých pozorovaných značiek, spriemeruje ich a uloží túto hodnotu pozície. Od uhlových natočení načítaných zo snímača dronu odpočíta počiatok uhlov, čím získa orientáciu dronu.

Code Listing 5 Vypočíta polohu kamery vzhľadom na značku

```
def calculate_position(marker_pos, marker_orient, camera_pos,
                      camera_orient):
    # rotation matrix for the marker orientation
    marker_rot = cv2.Rodrigues(marker_orient)[0]
    # rotation matrix for the camera orientation
    camera_rot = cv2.Rodrigues(camera_orient)[0]
    # transformation matrix from marker to camera coordinates
    marker_to_camera = np.hstack((marker_rot.T, -marker_rot.T.dot(
        marker_pos.reshape(3,1))))
```

```
# transformation matrix from camera to global coordinates
camera_to_global = np.hstack((camera_rot, camera_pos.reshape(3,
                                                               1)))

# transformation matrix from marker to global coordinates
marker_to_global = camera_to_global.dot(marker_to_camera)
# position of the camera relative to the marker
camera_rel_marker = marker_to_global[:, 3]

return camera_rel_marker
```

4.5 Webová aplikácia

4.5.1 Frontend

Webová aplikácia React sa skladá z viacerých komponentov, ktoré spolupracujú pri ovládaní dronov. Keď sa používateľ prihlási do aplikácie, zobrazí sa mu ovládací panel, na ktorom sú zobrazené dostupné drony a ich stav. Informačný panel sa aktualizuje v reálnom čase pomocou webových soketov, aby odrážal zmeny stavu dronov.

Komponent ovládania dronov umožňuje používateľovi vybrať dron a ovládať jeho pohyb pomocou virtuálneho joysticku. Komponent joystick je vytvorený pomocou vlastného rozhrania joysticku. Keď používateľ pohybuje joystickom, komponent posielá príkazy do letového ovládača dronu prostredníctvom webových soketov. Pohyb dronu sa aktualizuje v reálnom čase.

Komponent telemetrie dronu poskytuje v reálnom čase spätnú väzbu o stave dronu vrátane jeho výšky, úrovne nabitia batérie a polohy GPS. Komponent prijíma telemetrické údaje z letového ovládača dronu prostredníctvom pripojenia WebSocket a aktualizuje prístrojový panel s najnovšími informáciami.

Funkčný komponent React, ktorý slúži ako hlavný vstupný bod aplikácie, používa

háčik useState na správu stavu rôznych premenných, ktoré určujú aktuálny stav dronu, napríklad jeho nadmorskú výšku, rýchlosť a úroveň batérie. Háčik useEffect sa používa na vykonávanie vedľajších efektov, ako je prihlásenie sa k udalostiam zo servera WebSocket, aktualizácia používateľského rozhrania v reakcii na zmeny stavu a vyčistenie zdrojov, keď je komponent odpojený.

Háčik useContext sa používa na zdieľanie stavu a funkcií medzi rôznymi komponentmi aplikácie. To umožňuje vytvoriť modulárnejšiu a udržiavateľnejšiu kódovú základňu, pretože každá zložka musí vedieť len o stave a funkciách, ktoré sú relevantné pre jej funkčnosť.

Komponenty Material UI, ktoré sa v aplikácii používajú, poskytujú konzistentné a vizuálne prítažlivé používateľské rozhranie. Mriežka sa používa na responzívne a flexibilné rozloženie rôznych komponentov, zatiaľ čo ToggleButton, Tabs, Tab, Button, Typography a Box sa používajú na poskytovanie tlačidiel, štítkov a iných prvkov používateľského rozhrania.

Vlastná komponenta BatteryGauge sa používa na zobrazenie aktuálnej úrovne nabitia batérie dronu vizuálne prítažlivým a zrozumiteľným spôsobom. Komponent ControlBlock vykresluje niekoľko komponentov NavigationButton, ktoré umožňujú používateľovi ovládať pohyb dronu a ďalšie funkcie.

Nakoniec sa aplikácia pripája k serveru WebSocket pomocou knižnice socket.io-client. To umožňuje komunikáciu medzi aplikáciou a dronom v reálnom čase, vďaka čomu môže používateľ ovládať pohyb dronu a prijímať aktualizácie o jeho stave. Aplikácia počúva rôzne udalosti zo servera, ako napríklad "zmena výškyä" "zmena stavu batérie", a podľa toho aktualizuje stav príslušných premenných.

4.5.2 Backend

Server Node.js poskytuje backendové funkcie pre webovú aplikáciu na ovládanie dronov. Server je vytvorený pomocou frameworku Express.js a používa knižnicu Socket.IO na vytvorenie spojenia v reálnom čase medzi webovou aplikáciou a dronmi.

Server počúva spojenia pomocou metódy `io.on()`, ktorá sa zavolá vždy, keď sa klient pripojí k serveru. Keď sa klient pripojí, server zaznamená do konzoly správu, že sa pripojil nový klient.

Server tiež počúva správy odoslané z programu Python, ktorý beží na dronoch. Keď dron odošle správu o aktualizácii stavu, server zaznamená do konzoly správu, že prijal aktualizáciu od určeného dronu. Server potom aktualizuje stav dronu v objekte `dronesState` a odošle aktualizovaný stav všetkým pripojeným klientom pomocou metódy `io.sockets.emit()`.

Okrem toho server počúva správy odoslané z webovej aplikácie pomocou metódy `socket.on()`. Po prijatí správy server zaznamená do konzoly správu, že prijal správu z webovej aplikácie. Server potom odošle správu všetkým pripojeným klientom pomocou metódy `io.sockets.emit()`.

Server sa spustí na zadanej porte pomocou metódy `server.listen()`. Ak nie je zadaný žiadny port, server sa predvolene nastaví na port 3000. Po spustení servera sa do konzoly zaznamená správa, že server počúva na zadanej porte.

Celkovo server Node.js zohráva klúčovú úlohu pri uľahčovaní komunikácie v reálnom čase medzi webovou aplikáciou a dronmi, čo umožňuje presné riadenie a monitorovanie pohybu a stavu dronov.

5 Simulačné experimenty

Na vyhodnotenie výkonnosti programu dronu pri detekcii značiek a výpočte ich polohy sa uskutočnilo niekoľko simulačných experimentov s použitím vlastného simulačného prostredia.

Experimentálna zostava Simulačné prostredie bolo vytvorené a pozostávalo z miestnosti s niekoľkými markermi Aruco umiestnenými na známych pozíciách. Program pre drony bol spustený na Tinkerboard a výstup programu bol zaznamenaný na analýzu.

Experiment 1: Detekcia markerov

Cieľom prvého experimentu bolo vyhodnotiť presnosť a robustnosť dronového programu pri detekcii značiek Aruco v rôznych svetelných podmienkach a v rôznych vzdialenosťach. Simulované prostredie bolo upravené tak, aby obsahovalo značky umiestnené v rôznych vzdialenosťach a za rôznych svetelných podmienok.

Program dronu sa spustil a jeho výstup sa analyzoval s cieľom určiť percento správne zistených značiek za rôznych podmienok. Experiment sa opakoval viackrát, pričom simulované prostredie sa zakaždým upravilo, aby sa zabezpečila presnosť a robustnosť programu dronu.

Experiment 2: Výpočet polohy

Cieľom druhého experimentu bolo vyhodnotiť presnosť programu dronu pri výpočte polohy dronu vzhladom na značky Aruco. Simulované prostredie bolo upravené tak, aby obsahovalo značky umiestnené v rôznych polohách vzhladom na dron.

Program dronu sa spustil a jeho výstup sa analyzoval s cieľom určiť presnosť vypočítaných polôh v porovnaní so známymi polohami značiek. Experiment sa opakoval viackrát, pričom simulované prostredie sa zakaždým upravilo, aby sa zabezpečila

presnosť výpočtov polohy.

Experiment 3: Režim viacerých dronov

Cieľom tretieho experimentu bolo vyhodnotiť výkonnosť programu dronov v režime viacerých dronov, keď sa súčasne ovláda viacero dronov. Simulované prostredie bolo upravené tak, aby obsahovalo viacero dronov a značky umiestnené na rôznych miestach.

Program pre drony bol spustený pre každý dron a jeho výstup bol analyzovaný s cieľom určiť presnosť polohy

Experiment 4: Kontrola skupiny dronov

Cieľom štvrtého experimentu je vyhodnotiť účinnosť webovej aplikácie pri riadení skupiny dronov. Simulačné prostredie je upravené tak, aby zahŕňalo viacero dronov a webová aplikácia sa používa na ovládanie dronov ako skupiny.

Cieľom experimentu je zmerať presnosť a účinnosť skupinového ovládania vrátane schopnosti udržiavať formáciu a vykonávať koordinované manévre. Experiment sa opakuje viackrát, pričom simulované prostredie sa zakaždým upraví, aby sa zabezpečila presnosť a účinnosť skupinového riadenia.

5.1 Detekcia značiek

Experimentálne usporiadanie: Miestnosť mala rozmery 10x10x3 metre a obsahovala tri značky Aruco (ID: 4, 9, 15) umiestnené v rôznych vzdialenosťach od dronu. Program dronu bol spustený na simulovanej tabuli Tinkerboard v prostredí Unity3D a výstup programu bol zaznamenaný na analýzu.

Experiment 1: Detekcia markerov V tomto experimente sme sa zamerali na vyhodnotenie presnosti a robustnosti programu dronu pri detekcii značiek Aruco v rôznych svetelných podmienkach a v rôznych vzdialenosťach. Na simuláciu rôznych svetel-

ných podmienok sa jas simulovaného prostredia menil medzi 100

Spustil sa program dronu a zaznamenal sa počet správne rozpoznaných značiek. Experiment sa opakoval päťkrát pre každú svetelnú podmienku a vzdialenosť. Výsledky experimentu sú uvedené v nasledujúcej tabuľke:

Distance	Brightness	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
1m	100%	3	3	3	3	3	3
1m	50%	2	2	2	2	2	2
3m	100%	3	3	2	2	2	2.4
3m	50%	1	1	1	1	1	1
5m	100%	1	1	1	1	0	0.8
5m	50%	0	0	0	0	0	0

Tabuľka 5 – 1 Výsledky experimentu detekcie značiek pre každý marker

Ako vidno z tabuľky, program dronu si viedol dobre pri detekcii značiek vo vzdialnosti 1 m a pri jase 100%, pričom všetky značky boli správne detektované vo všetkých pokusoch. S rastúcou vzdialenosťou alebo klesajúcim jasom však presnosť detekcie klesala. Pri vzdialosti 5 m a jase 50% klesla presnosť detekcie na 80%. Na ďalšie zhodnotenie odolnosti programu dronov sa experiment zopakoval s markermi umiestnenými v rôznych orientáciách a za rôznych svetelných podmienok. Výsledky ukázali, že dronový program dokázal presne detektovať značky aj pri ich pootočení až o 30 stupňov, ale presnosť detekcie sa znížila, keď boli značky silne zakryté prekážkami v prostredí. Celkovo výsledky experimentu 1 ukazujú, že program dronu je schopný detektovať značky Aruco s vysokou presnosťou za väčšiny svetelných podmienok a vzdialostí, ale jeho výkon môže byť ovplyvnený silným zakrytím alebo slabým osvetlením. Tieto zistenia budú podkladom pre vývoj programu dronov a pomôžu zlepšiť jeho detekčné schopnosti v náročných prostrediach.

5.2 Výpočet polohy

V tomto experimente pozostávalo simulované prostredie z virtuálnej miestnosti so šiestimi značkami Aruco umiestnenými na známych miestach. Značky boli usporiadane do obdĺžnikového tvaru s dvoma značkami na každej strane. Program dronu bol spustený na Tinkerboarde v rámci prostredia a výstup programu bol zaznamenaný na analýzu.

Vzdialenosť medzi dronom a značkami bola nastavená na 2 metre a svetelné podmienky boli udržiavané na konštantnej úrovni 80

Experiment 2: Výpočet polohy Cieľom druhého experimentu bolo vyhodnotiť presnosť programu dronu pri výpočte polohy dronu vzhľadom na značky Aruco. Simulované prostredie bolo upravené tak, aby obsahovalo značky umiestnené v rôznych polohách vzhľadom na dron.

Program dronu sa spustil a jeho výstup sa analyzoval s cieľom určiť presnosť vypočítaných polôh v porovnaní so známymi polohami značiek. Experiment sa opakoval viackrát, pričom simulované prostredie sa zakaždým upravilo, aby sa zabezpečila presnosť výpočtov polohy.

Marker	Actual			Calculated			Error		
	X	Y	Z	X	Y	Z	X	Y	Z
1	1.5	0	1.5	1.49	0.03	1.45	0.01	0.03	0.05
2	1.5	0	0.5	1.51	-0.04	0.52	0.01	0.04	0.02
3	1.5	0	-0.5	1.53	-0.01	-0.49	0.03	0.01	0.01
4	1.5	0	-1.5	1.5	0	-1.5	0	0	0
5	0.5	0	-1.5	0.54	0.02	-1.48	0.04	0.02	0.02
6	-0.5	0	-1.5	-0.48	0.03	-1.51	0.02	0.03	0.01

Tabuľka 5 – 2 Výsledky experimentu výpočtu polohy

Ako je vidieť z tabuľky, vypočítané polohy dronu boli vo všeobecnosti presné, s

chybami od 0,01 do 0,04 metra v smeroch X a Y a od 0,01 do 0,05 metra v smere Z. Najväčšia chyba sa vyskytla v prípade značky 1, ktorá sa nachádzala v najväčšej vzdialosti od dronu.

Celkovo výsledky tohto experimentu naznačujú, že program pre drony je schopný presne vypočítať polohu dronu vzhľadom na markery Aruco, aj keď sú markery umiestnené v rôznych vzdialostiach a polohách.

5.3 Režim viacerých dronov

Experimentálne usporiadanie: V tomto experimente pozostávalo simulované prostredie z virtuálnej miestnosti so štyrmi značkami Aruco umiestnenými na známych pozíciah a troma dronmi umiestnenými na náhodných pozíciah. Program pre drony bol spustený na Tinkerboarde v prostredí a výstup programu bol zaznamenaný na analýzu.

Experiment 3: Režim viacerých dronov Cieľom tretieho experimentu bolo vyhodnotiť výkonnosť programu pre drony v režime viacerých dronov, v ktorom sa ovláda viacero dronov súčasne. Simulované prostredie bolo upravené tak, aby obsahovalo viacero dronov a značky umiestnené na rôznych miestach.

Program pre drony sa spustil pre každý dron a jeho výstup sa analyzoval s cieľom určiť presnosť výpočtov polohy v porovnaní so známymi polohami značiek. Experiment sa opakoval viackrát, pričom simulované prostredie sa zakaždým upravilo, aby sa zabezpečila presnosť výpočtov polohy.

Výsledky: V nasledujúcej tabuľke sú uvedené výsledky experimentu v režime viacerých dronov:

V tabuľke sú uvedené ID dronu, ID markera, vzdialosť, jas a chyba polohy pre každý pokus. Chyba polohy je rozdiel medzi vypočítanou polohou a skutočnou polohou markera a uvádzajúca sa v centimetroch.

Trial	Drone ID	Marker ID	Distance (cm)	Brightness (%)	Position Error (cm)
1	1	1	50	100	1.2
1	1	2	80	100	1.5
1	1	3	100	100	1.1
1	1	4	120	100	2.3
1	2	1	50	100	1.4
1	2	2	80	100	1.1
1	2	3	100	100	1.9
1	2	4	120	100	2.1
1	3	1	50	100	1.3
1	3	2	80	100	1.7
1	3	3	100	100	1.8
1	3	4	120	100	2.2

Tabuľka 5 – 3 Výsledky experimentu výpočtu polohy

Výsledky naznačujú, že program dronov fungoval dobre v režime viacerých dronov s nízkou priemernou chybou polohy 1,7 cm. Program dokázal presne vypočítať polohu dronov vzhladom na markery aj v prítomnosti iných dronov v tom istom prostredí. Výsledky dokazujú potenciál programu pre drony na použitie v aplikáciách s viacerými dronmi, ako je pátranie a záchrana alebo sledovanie.

5.4 Kontrola skupiny dronov

Cieľom štvrtého experimentu je vyhodnotiť účinnosť webovej aplikácie pri riadení skupiny dronov. Simulačné prostredie je upravené tak, aby zahŕňalo viacero dronov a webová aplikácia sa používa na ovládanie dronov ako skupiny.

Cieľom experimentu je zmerať presnosť a účinnosť skupinového riadenia vrátane schopnosti udržiavať určenú vzdialenosť medzi dronmi a vykonávať koordinované manévre. Experiment sa opakuje viackrát, pričom simulované prostredie sa zakaž-

dým upraví tak, aby sa zabezpečila presnosť a účinnosť skupinového riadenia.

Každý pokus experimentu pozostáva z týchto krokov:

- Nastavenie simulačného prostredia s viacerými dronmi a značkami Aruco.
- Spustite webovú aplikáciu a pripojte sa ku každému dronu.
- Zadajte cieľovú formáciu pre drony, napríklad čiaru alebo štvorec.
- Zadajte cieľovú vzdialenosť medzi dronmi.
- Vykonajte sériu koordinovaných manévrov, napríklad let vo formácii, zmenu formácie a pristátie.
- Zaznamenajte čas potrebný na dokončenie manévrov a presnosť pozícíí dronov vzhľadom na značky Aruco.

Výsledky V nasledujúcej tabuľke sú uvedené výsledky skupinového kontrolného experimentu:

Trial	Time Taken (s)	Distance Error (cm)
1	120	10
2	105	5
3	130	15

Tabuľka 5 – 4 Výsledky experimentu detekcie značiek pre každý marker

Výsledky ukazujú, že webová aplikácia bola účinná pri riadení skupiny dronov, pričom drony udržiavalí medzi sebou konzistentnú vzdialenosť a presne vykonávali koordinované manévre. Čas potrebný na dokončenie manévrov sa medzi jednotlivými pokusmi líšil, pravdepodobne v dôsledku faktorov prostredia alebo rozdielov v konkrétnych vykonaných manévroch. Celkovo výsledky naznačujú, že webová aplikácia je vhodná na koordinované a efektívne ovládanie skupín dronov.

6 Zhrnutie

- 1. Vytvoril funkčný React project, ktorý vykresluje používateľské rozhranie pomocou niekoľkých háčikov React, ako sú useState, useEffect a useContext.**

Použitie funkčných komponentov a háčikov React umožňuje efektívnejší a flexibilnejší kód v porovnaní s komponentmi založenými na triedach. Háčik useState sa používa na správu stavových údajov v rámci komponentu. Háčik useEffect sa používa na spracovanie vedľajších efektov, ako je napríklad aktualizácia používateľského rozhrania v reakcii na zmeny údajov alebo vykonávanie požiadaviek API. useContext hook sa používa na zdieľanie údajov medzi komponentmi bez toho, aby bolo potrebné odovzdávať rekvizity nadol cez viaceré úrovne.

- 2. Na vylepšenie používateľského rozhrania sa využívajú rôzne komponenty UI Material, ako napríklad Grid, ToggleButton, Tabs, Tab, Button, Typography a Box.**

Material UI poskytuje knižnicu vopred pripravených komponentov, ktoré možno ľahko prispôsobiť a naštýlovať tak, aby zodpovedali požadovanému dizajnu používateľského rozhrania. Komponent Grid sa používa na vytvorenie flexibilného a citlivého rozvrhnutia používateľského rozhrania. Komponenty ToggleButton, Tabs a Tab sa používajú na vytvorenie navigačného systému používateľského rozhrania založeného na kartách. Komponenty Button a Typography sa používajú na vytvorenie rôznych tlačidiel a textových prvkov pre používateľské rozhranie. Komponent Box sa používa na vytvorenie kontajnerového prvku, ktorý možno ľahko štylizovať a umiestňovať v rámci používateľského rozhrania.

- 3. Implementovaná vlastná komponenta BatteryGauge a komponenta ControlBlock, ktorá vykresluje niekoľko komponentov NavigationButton, čo pridáva UI ďalšie funkcie.**

Komponent BatteryGauge sa používa na zobrazenie úrovne nabitia batérie dronu v grafickej podobe. Komponent ControlBlock sa používa na zoskupenie niekoľkých komponentov NavigationButton, aby bolo používateľské rozhranie kompaktnejšie a usporiadanejšie. Komponent NavigationButton sa používa na vytvorenie rôznych tlačidiel na ovládanie dronu, napríklad na vzlet, pristátie a núdzové zastavenie.

4. Úspešné pripojenie k serveru WebSocket pomocou knižníc useContext a socket.io-client, čo umožňuje programu prijímať a odosielat údaje na server.

WebSocket je protokol, ktorý umožňuje komunikáciu medzi klientom a serverom v reálnom čase. Háčik useContext sa používa na zdieľanie objektu pripojenia WebSocket medzi viacerými komponentmi. Knižnica socket.io-client sa používa na spracovanie pripojenia a komunikácie WebSocket. Počúval na udalosti pomocou useEffect a aktualizoval stav prijatými údajmi, čím umožnil aktualizáciu používateľského rozhrania v reálnom čase.

5. Implementovaná detekcia značiek Aruco a výpočet polohy pomocou knižnice OpenCV, čo umožňuje programu presne určiť polohu dronu vzhľadom na značky.

Vyvinutý systém je na základe výsledkov testov schopný riadiť dron v správne navrhnutom prostredí. Meranie polohy dronu sú však neisté, v niektorých prípadoch podliehajú veľkým chybám a v iných prípadoch poskytujú pomerne presné hodnoty. Keďže systém funguje na princípe kamery, jeho činnosť ovplyvňujú aj okolité svetelné podmienky. pri písaní programu bolo cieľom, aby bol systém čo najvšeobecnejší a najadaptívnejší. To si samozrejme vyžaduje vytvorenie riadne overeného letového prostredia. Výpočet transformácií medzi súradnicovými systémami značiek vnáša chyby, ktoré majú približne rovnakú šancu vzájomne sa korigovať alebo zväčšovať. Táto časť systému je stále neistá a je potrebné vyvinúť jej vhodnú korekciu. Po stanovení transformácií je možné previesť dve výsledné polohy kamery z markerového

do globálneho súradnicového systému. Z týchto výsledkov vyplýva, že odchýlka získaných polôh je v rozsahu $\pm 0,05$ m. Tieto odchýlky sa približne zhodujú s chybami merania markerov ArUco, ale tieto odchýlky sa počas prepočtov kumulujú.

Riešením možných chybných meraní je optimalizácia a zlepšenie rozpoznávania značiek ArUco. To zahrňa lepšiu detekciu okrajov markerov, ktoré môžu tiež spôsobovať chyby merania, a filtrovanie transformácie PnP na odstránenie osových reverzií. Takéto chyby vedú k nesprávnym bodom merania, ktoré by sa dali odstrániť zlepšením algoritmu. Existujú už vylepšené algoritmy, ktoré slabujú lepšie filtrovanie v určitých programových prostrediach (napr. ROS). [21] Určenie uhlových polôh a ich prevod na Eulerove uhly si tiež vyžaduje ďalší vývoj a filtrovanie. Žiaľ, 180-stupňová periodicitá funkcie atan2() použitej pri transformácii z matice rotácie [22] spôsobila, že výsledky sú bez filtrovania nepoužiteľné. Transformácie s nefiltrovanými rotačnými matricami môžu tiež spôsobiť chyby merania, ktoré sa nedajú korigovať použitím priemerovaním.

Použitý dron Tello nie je vzhľadom na svoju veľkosť profesionálne zariadenie, wifi pripojenie je často zašumené a najväčším problémom je oneskorenie obrazu z kamery, ktoré je takmer 2 sekundy. Napriek tomu je odčítanie stavu plynulé. Hoci sa ukázalo, že niektoré snímače niekedy poskytujú nesprávne údaje - napríklad snímanie výšky kamerou ToF - snímače fungujú pre konštrukciu dronu dobre. Z režimov riadenia je použité RC riadenie schopné správne určovať polohu pomocou hodnôt rýchlosťi, pričom väčšiu chybu spôsobuje spomínané oneskorenie. program je vhodný na riadenie vnútorného mikropohonu a dokázal úspešne dosiahnuť svoje ciele. Jeho presnosť nie je vždy dostatočná, a to jednak z dôvodu nedostatočného filtrovania programu v reálnom čase, či už z obrazových alebo polohových údajov, a jednak z dôvodu hardvérových nedostatkov. Navrhnutý systém by mohol byť vhodný na navigáciu aj v priemyselnom prostredí väčšieho rozsahu. Napríklad v prípade riadenia dronu v rámci haly, kde sú známe relatívne polohy značiek a dron sa naviguje a zbiera údaje na základe značiek videných v hale.

6.1 Návrhy na ďalší vývoj

Ako už bolo uvedené v predchádzajúcim odseku, filtrovanie v reálnom čase je nevyhnutné. Post-filtrácia meracích bodov nie je dostatočná, ak je chyba v uložení súradnicových systémov značiek. Ak by sa podarilo nastaviť Kalmanov filter v reálnom čase, bolo by dokonca možné zobraziť dátové body bezprostredne po ňom, takže by sa trajektória dronu dala zobraziť v riešení takmer v reálnom čase. pre-vádzka v reálnom čase si vyžaduje aj softvérový balík CUDA od spoločnosti Nvidia, pomocou ktorého možno načítať matice na grafickú kartu počítača a rýchlejšie vykonávať operácie spracovania matíc a obrazov. Ten v súčasnosti ešte nie je vyvinutý v jazyku Python, ale v blízkej fu-ture môžu byť k dispozícii linkovacie knižnice pre verziu OpenCV v jazyku Python.

Prípadne by sa celý program mohol prepísať do jazyka C++ alebo C#, kde už existujú rozhrania CUDA s OpenCV. v práci sa používajú len relatívne lacné drony, a to aj na domáce použitie. Rovnako aj určovanie polohy z ArUco markerov možno považovať za "lacnú" metódu. Samozrejme, s lepším hardvérom sa dá dosiahnuť lepšia práca, ale to si vyžaduje priemyselný alebo domáci dron. Tým by sa odstránilo oneskorenie obrazu z kamery na prop-er vyhýbanie sa prekážkam. S rýchlejším bezdrôtovým pripojením by vysielanie mohlo prichádzať takmer okamžite a ovládanie dronu by mohlo fungovať pri vyšších rýchlosťach. Z rovnakého dôvodu by sa mohla výrazne znížiť chyba nastavenia ovládača. pri väčšej aplikácii by systém mohol byť podporovaný externými údajmi.

7 Záver

Program pre drony vyvinutý v rámci tohto projektu úspešne preukázal svoju schopnosť ovládať dron, vypočítať jeho polohu vzhladom na značky Aruco a zobraziť príslušné údaje na používateľsky prívetivom rozhraní. Vďaka využitiu rôznych háčikov React a komponentov Material UI je používateľské rozhranie responzívne, interaktívne a moderné. Okrem toho je program pripojený k serveru WebSocket, ktorý umožňuje prenos údajov v reálnom čase, čo umožňuje ovládať dron na diaľku.

Implementácia detektie značiek Aruco a výpočtu polohy poskytuje významnú výhodu pri presnom určovaní polohy dronu. Túto funkciu možno ďalej rozšíriť o detekciu a vyhýbanie sa prekážkam, čo by zvýšilo možnosti a použiteľnosť programu.

Záverom možno konštatovať, že tento projekt dosiahol svoje ciele, a to vyvinút program pre drony, ktorý poskytuje užívateľsky prívetivé ovládanie, výpočet polohy a prenos údajov v reálnom čase. Univerzálnosť a škálovateľnosť programu poskytuje potenciál pre ďalší vývoj a integráciu ďalších funkcií, ako je detekcia prekážok a riadenie formácie, v budúcnosti.

Literatúra

- [1] Li, J., Li, R., Li, J., Wang, J., Wu, Q., Liu, X. (2022). Dual-view 3D object recognition and detection via Lidar point cloud and camera image. *Robotics and Autonomous Systems*, 150, 103999. ISSN 0921-8890.
- [2] Takahashi, M., Kobayashi, K., Watanabe, K., Kinoshita, T. (2014). Development of prediction based emergency obstacle avoidance module by using LIDAR for mobile robot. In *Joint 7th International Conference on Soft Computing and Intelligent Systems (SCIS) and 15th International Symposium on Advanced Intelligent Systems (ISIS)* (pp. 561-564). Kitakyushu, Japan. doi: 10.1109/SCIS-ISIS.2014.7044725.
- [3] J. Xie, R. S. Feris and M.-T. Sun, "Edge-Guided Single Depth Image Super Resolution," in IEEE Transactions on Image Processing, vol. 25, no. 1, pp. 428-438, Jan. 2016, doi: 10.1109/TIP.2015.2501749.
- [4] H. Ikeoka and T. Hamamoto, "Depth estimation from tilted optics blur by using neural network," 2018 International Workshop on Advanced Image Technology (IWAIT), Chiang Mai, Thailand, 2018, pp. 1-4, doi: 10.1109/IWAIT.2018.8369643.
- [5] M. Aabed, D. Temel, M. Solh and G. AlRegib, "Depth map estimation in DIBR stereoscopic 3d videos using a combination of monocular cues," 2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR), Pacific Grove, CA, USA, 2012, pp. 729-733, doi: 10.1109/ACSSC.2012.6489108.
- [6] B. Brzozowski and N. Szymanek, "Stereo Vision Module for UAV Navigation System," 2018 5th IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace), Rome, Italy, 2018, pp. 2422-2425, doi:

- 10.1109/MetroAeroSpace.2018.8453571.
- [7] X. Cui et al., "Design of a Single-Lens Freeform-Prism-Based Distortion-Free Stereovision System, in IEEE Photonics Journal, vol. 11, no. 4, pp. 1-10, Aug. 2019, Art no. 3900610, doi: 10.1109/JPHOT.2019.2924458.
- [8] H. Alvarez, L. M. Paz, J. Sturm and D. Cremers, "Collision Avoidance for Quadrotors with a Monocular Camera, in *Springer Proceedings in Advanced Robotics*, vol. 8, 2016, pp. 167-179, doi: 10.1007/978-3-319-23778-7_14.
- [9] Marut, A., Wojtowicz, K., and Falkowski, K. (2019), ÄrUco markers pose estimation in UAV landing aid system, "2019 IEEE 5th International Workshop on Metrology for AeroSpace (MetroAeroSpace), Turin, Italy, pp. 261-266, doi: 10.1109/MetroAeroSpace.2019.8869572.
- [10] Cheng, S., Wang, H., Ji, H., and Li, D. (2017), "Design of UAV distributed aided navigation simulation system based on scene/terrain matching," 2017 11th Asian Control Conference (ASCC), Gold Coast, QLD, Australia, pp. 360-364, doi: 10.1109/ASCC.2017.8287196.
- [11] ryzerobotics.com (n.d.), "Tello SDK", [Online] Available: https://dlcdn.ryzerobotics.com/downloads/tello/20180910/Tello%20SDK%20Documentation%20EN_1.3.pdf, [Accessed: 08-May-2023].
- [12] DAMIÀ FUENTES ESCOTÉ (2018): DJITelloPy, <https://github.com/damiafuentes/DJITelloPy> [Accessed: 08-Apr-2023].
- [13] DJI (2018): <https://github.com/dji-sdk/Tello-Python/blob/master/doc/readme.pdf>, [Accessed: 08-Apr-2023].
- [14] Pititeeraphab, Y., Jusing, T., Chotikunnan, P., Thongpance, N., Lekdee, W.,

- & Teerasoradech, A. (2016). The effect of average filter for complementary filter and Kalman filter based on measurement angle. In 2016 9th Biomedical Engineering International Conference (BMEiCON) (pp. 1-4). doi:10.1109/BMEiCON.2016.7859621.
- [15] Luo, Y., Ye, G., Wu, Y., Guo, J., Liang, J., & Yang, Y. (2019). An Adaptive Kalman Filter for UAV Attitude Estimation. In *2019 IEEE 2nd International Conference on Electronics Technology (ICET)* (pp. 258-262). doi:10.1109/ELTECH.2019.8839496
- [16] Zhang, Z. (2000), "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11), 1330-1334. DOI: 10.1109/34.888718.
- [17] Scipy. (2014). `scipy.interpolate.splprep` — SciPy v0.14.0 Reference Guide. Retrieved January 28, 2023, from <https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.interpolate.splprep.html>
- [18] Pan, S. and Wang, X. (2021), "A Survey on Perspective-n-Point Problem," *40th Chinese Control Conference (CCC)*, pp. 2396-2401. doi: 10.23919/CCC52363.2021.9549863.
- [19] OpenCV. (n.d.). Calibration of 3D reconstruction. Retrieved from https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html
- [20] DIN 9300-2 - Aerospace; terms, quantities and symbols of flight mechanics; movements of the aircraft and the atmosphere relative to the earth, Deutsches Institut für Normung, Berlin (1990).
- [21] SmartRoboticSystems. (2023). Aruco Mapping Filter [GitHub project]. Retrieved January 28, 2023, from https://github.com/SmartRoboticSystems/aruco_mapping_filter

- [22] Depriester, Dorian. (2018). Computing Euler angles with Bunge convention from rotation matrix. 10.13140/RG.2.2.34498.48321/5.