VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Karkaso kūrimas standartizuotai roboto platformai

Framework development for standardized robot platform

Magistro baigiamasis darbas

Atliko: Darius Miškinis (parašas)

Darbo vadovas: lekt. Linas Būtėnas (parašas)

Recenzentas: prof. dr. Linas Laibinis (parašas)

Vilnius – 2020

Santrauka

Šiame darbe pateikiamas bepiločio orlaivio nusileidimo į pažymėtą vietą sprendimas, panaudojant ROS karkasą robotams ir bepiločio orlaivio platformą AR.Drone. Darbe aprašytas AR.Drone kvadkopteris, supažindinama su ROS karkasu, jo komponentais ir naudojama terminologija. Aprašomas markerių suradimo karkasas ar_pose, kuris bus naudojamas bepiločio orlaivio nusileidimo taškui rasti. Supažindinama su Gazebo simuliacine aplinka, kurioje bus vykdomi pirminiai tyrimai. Darbe išnagrinėjamas ir aprašomas, sukurtas atviro kodo programinės įrangos sprendimas iškeltai užduočiai atlikti. Pateikiamas drono nusileidimo agresyvumo parametrų tyrimas.

Raktažodžiai: bepilotis orlaivis, kvadkopteris, ROS, robotikos karkasas, AR.Drone, ar_pose, Gazebo.

Summary

This thesis investigates the task of unmanned aerial vehicle landing to the marked landing spot, using the ROS robotic framework and AR.Drone quadcopter platform. This document reviews the AR.Drone quadcopter and quadcopter flying principles. In particular, the ROS framework, its components and used terminology are overviewed. This paper also reviews the fiducial marker detection framework ar_pose, since this framework will be used for UAV landing spot detection. The Gazebo simulation environment is introduced because this tool will be used for initial task research. Moreover, this document presents the software that was created to solve the given task of landing a drone on a fiducial marker. Finally, this thesis presents research on how drone landing control module maneuvers aggressiveness levels impact the drone landing time and overall task success.

Keywords: UAV, quadcopter, ROS, robotic framework, AR.Drone, ar_pose, Gazebo.

TURINYS

Sutartinis terminų žodynas

Bepilotis orlaivis - unmanned aerial vehicle (UAV).

Inercinių matavimų blokas - inertial measurement unit (IMU).

Kvadkopteris – ketursraigtis orlaivis - quadcopter.

Lygianaris tinklas - peer-to-peer network.

Mažas bepilotis orlaivis - micro aerial vehicle (MAV).

Palyginamasis žymeklis - objektas skirtas nustatyti atstumą arba dydį - fiducial marker.

Pokrypis - orlaivio sūkimąsis aplink savo ašį į kairę arba dešinę - yaw.

Polinkis - orlaivio palinkimas į priekį arba atgal savo koordinačių sistemoje - pitch.

Posvyris - orlaivio pasvyrimas į kairę arba dešinę savo koordinačių sistemoje - roll.

Įvadas

Autonominiai dronai tapo populiariu įrankiu karo pramonėje, civilinėje srityje. Jie dažniausiai yra naudojami apžvalgos, duomenų surinkimo, mokslinių tyrimų ir pramogų tikslais. Dronai turi didelį praktinį panaudojimą. Jie buvo panaudoti įvykus nelaimei Fukušimos branduoliniame reaktoriuje [SOT+17], tiriant reaktorių padarytą žalą, nerizikuojant tyrėjų gyvybe ir sveikata. Yra ir daugiau dronų panaudojimo galimybių. Bepiločio roboto ir bepiločio orlaivio sistema, po stipraus žemės drebėjimo, leistų vykdyti išgyvenusių žmonių paiešką, nerizikuojant gelbėtojų gyvybėmis [KTL+12].

Dronai netolimoje ateityje gali pakeisti prekes pristatančius kurjerius. Didelės korporacijos, tokios kaip Amazon ir UPS [SNR18, SH16], jau dabar turi eksperimentines prekių tiekimo su dronais programas. Netgi sveikatos apsaugos srityje, tokie startuoliai kaip Zipline, bando panaudoti dronus medikamentams, kraujo komponentams pristatyti [SS18].

Viena populiariausių dronų formų yra kvadkopteris. Kvadkopteris savo skrydžio savybėmis yra panašus į sraigtasparnį. Jis gali pakilti ir nusileisti vertikaliai, kyboti ore, judėti bet kuria kryptimi, prieš tai nepasisukęs. Tai leidžia kvadkopteriams manevruoti nedidelėse erdvėse, ne tik vidiniuose namų kiemuose, bet koridoriuose ir kitose patalpose. Orlaivio su keturiais horizontaliai nukreiptais rotoriais konceptas buvo sukurtas dar 1922 metais, bet jis greitai buvo nurungtas dviejų rotorių sraigtasparnio. Nors kvadkopteris mechaniškai paprastesnis, jis yra žymiai nestabilesnis, todėl sunkiau valdomas. Nenaudojant elektroninių kontrolės ir stabilizavimo sistemų, rankinis kvadkopterio valdymas buvo sudėtingas. Tradiciniai sraigtasparniai sunaudoja mažiau energijos nei kvadkopteriai.

Išpopuliarėjus mažiems bepiločiams orlaiviams (angl. Micro Aerial Vehicles) kvadkopterių architektūra tapo vėl naudojama. Kvadkopteriai yra paprastesni mechaniškai, nes visi keturi rotoriai turi fiksuotą posvyrį ir polinkį. Dar vienas kvadkopterio pranašumas - jo rotoriai gali būti korpuso viduje. Taip apsaugomas rotorius nuo susidūrimų su kliūtimis. Kvadkopterio rotoriai yra mažesnio diametro, todėl skrydžio metu turi mažiau kinetinės energijos ir susidūrus su kliūtimi jie padaro mažiau žalos. Mažiems bepiločiams orlaiviams naviguoti erdvėje padeda juose sumontuoti sensoriai. Vienas pagrindinių sensorių yra inercinių matavimų blokas (inertial measurement unit), kuris matuoja drono akseleraciją ir orientaciją [ESD12b]. Kitas populiarus sensorius yra GPS, leidžiantis bepiločiam orlaiviui nustatyti savo globalią poziciją. Net jei dronas yra pilotuojamas žmogaus, šie matavimo prietaisai padeda valdyti droną ir naviguoti [Stu18]. GPS sensorius yra tinkamas pozicijai nustatyti tik nedideliu tikslumu. Civilinės paskirties imtuvai dažniausiai turi 3-5 metrų tikslumą. Bepiločiam orlaiviui reikia kitų būdų nustatyti savo lokaciją. Yra daugybė kitų sensorių, kurie padeda atlikti šią užduotį, nuo paprastų ir pigių ultragarsinių sensorių, kurie

matuoja atstumą viena kryptimi, iki aukštos rezoliucijos lazerinių sistemų, kurios sukuria detalius aplinkos matavimus. Deja, jos sąlyginai brangiai kainuoja. Vienas populiariausių būdų gauti informaciją apie bepiločio orlaivio aplinką yra skaitmeninė vaizdo kamera. Tai pigus ir lengvas sensorius, suteikiantis didelį informacijos kiekį apie orlaivio aplinką. Skaitmeninės kameros yra vienas populiariausių sensorių bepiločiuose orlaiviuose. Tiesa, skaitmeninės vaizdo kameros sukuria didelį duomenų kiekį ir jam apdoroti reikia pakankamai didelių duomenų apdorojimo pajėgumų.

Vienas pagrindinių autonominių sistemų komponentų yra lokalizacija. Orlaivis privalo turėti galimybę nustatyti savo poziciją aplinkos atžvilgiu. Norint išlaikyti stabilią poziciją ore, reikalingi pastovūs maži judesiai, kompensuojantys išorines jėgas. Šiai užduočiai atlikti nedideliu tikslumu yra naudojami IMU (angl. Inertial Measurement Unit), bet laikui bėgant matavimų paklaidos kaupiasi ir bepilotis orlaivis tolsta nuo savo tikslinių koordinačių. Šią problemą galima išspręsti žinant fiksuoto objekto koordinates aplinkoje.

Darant tyrimus su bepiločiais orlaiviais, nebūtina juos konstruoti patiems. Yra standartizuotų sistemų, kurios nebrangios ir pasižymi pakankamu sensorių kiekiu. Viena iš tokių sistemų yra AR.Drone. AR.Drone yra labai populiari platforma moksliniuose tyrimuose [VCM+16, NAC+18, BCV+11, KVF+11, SKF+12, DV11, WMZ12, LZ13, BHO14a, BHO14b, LMZ13, PBC14, BCS11a]. Dėl savo prieinamos kainos ir atviro API, ši autonominio orlaivio platforma naudojama labai plačiai, nuo orlaivio valdymo mintimis per elektroencefalogramas [NAC+18] iki dronų atsparumo DoS atakoms [VCM+16].

Apžvelgiant tyrimus, atliekamus su autonominiais orlaiviais, galime reziumuoti, kad dažniau yra tiriami praktiškesni dronų panaudojimo būdai: kito drono sekimo, norint skraidinti autonominius orlaivius rikiuotėje [WMZ12], arba dronų nuleidimo į tam tikrą tašką uždaviniai [LSP09, MYW+13].

Sėkmingo drono nuleidimo uždavinys nėra trivialus. Atliekant nusileidimą gali tekti atpažinti kitus dronus ir išvengti susidūrimo su jais [CVR+18] arba nustatyti nusileidimo tašką ir nusileisti ant jo. Sėkmingo ir efektyvaus drono nuleidimo problemą tiria ne viena mokslininkų grupė [YSZ13, BHO14a, BHO14a, LSP09, MYW+13]. Kai kurie eksperimentai koncentruojasi į palyginamųjų markerių panaudojimą, bandant kvadkopterį išlaikyti vienoje vietoje, kai GPS nustoja veikti arba jis tampa labai netikslus, dėl aplinkos veiksnių, tokių kaip aukšti pastatai [LSP09]. Kiti tyrimai koncentruojasi į drono kameros matomų raštų analizavimą, norint lokalizuotis aplinkoje [LZ13]. [MYW+13] drono nuleidimo vietai surasti nenaudojo palyginamųjų markerių. Tyrėjai sukūrė švyturį iš infraraudonos šviesos diodų ir eksperimentų metu nustatė, kad ši konfigūracija tinkama naudoti patalpose, bet prastai veikia dieną lauko sąlygomis.

Šio darbo tikslas - sukurti bepiločio orlaivio autonominio nusileidimo modelį ir atlikti jo efektyvumo tyrimą. Šiam tikslui pasiekti keliami šie uždaviniai:

1. Sukurti drono valdymo algoritmų testavimo platformą, įgalinančią išbandyti valdymo algoritmus simuliacinėje aplinkoje, prieš perkeliant juos į tikrus dronus.
2. Sukurti ROS karkaso modulį bepiločio orlaivio valdymui. Šis modulis leistų dronui, pasinaudojus savo vaizdo kamera, nusileisti ant iš anksto apibrėžto markerio. Sistema turi susitvarkyti su nusileidimo metu pamestu markeriu.
3. Ištirti sukurtą drono valdymo modulį, atliekantį bepiločio orlaivio nusileidimo ant palyginamojo žymeklio užduotį, su simuliacine aplinka ir su tikru dronu, parenkant optimalias valdymo parametrų reikšmes.

1. Susijusių darbų analizė

Darbo tikslui pasiekti buvo pasirinkta AR.Drone bepiločio orlaivio platforma. Straipsnyje tyrjai [KVF+11] pademonstravo, kad galima AR.Drone platformą naudoti kaip mokslinių eksperimentų įrankį tiriant objektų sekimo ir autonominės navigacijos uždavinius. Ši mokslininkų komanda nenaudojo simuliacinės aplinkos ir visus eksperimentus atliko iš karto su tikru dronu. Kūrėjai nenaudojo robotikai skirtų karkasų ir sukūrė savo programinės įrangos sprendimą. Autoriai [SKF+12] sukūrė programą kuri iš drono gauna vaizdo srautą jį analizuoja ir siunčia atgal drono judesio komandas. Straipsnyje teigiama, kad AR.Drone platforma yra puikus ir nebrangus įrankis atlikti pirminius tyrimus, taip patikrinant eksperimento konceptą, prieš naudojant brangią specializuotą įrangą.

AR.Drone platformą su ant drono sumontuotais papildomais skaičiavimo įrenginiais vaizdo apdorojimui naudojo [LZ13]. Jie pademonstravo, kad platformą galima pritaikyti žmonių sekimo ir drono pozicijos nustatymo uždaviniams tirti. Atliekant figūrinio skraidymo eksperimentus tyrėjai pastebėjo, kad ši platforma su jų padarytomis modifikacijomis, gali sekti ant žemės išbraižytas figūras pakankamai tiksliai, nors ir turi labai žemos raiškos kamerą. Drono pozicijos nustatymui jie naudojo specialų kambarį gebantį sekti jame esančius objektus dideliu tikslumu. Tyrėjai nenaudojo simuliacinės aplinkos ir tyrimus iš karto atliko su tikru dronu.

Tiriant bepiločio orlaivio nusileidino į pristatymo tašką problemą, dažnai naudojami nusileidimo vietos žymekliai. Straipsniuose [BHO14a, BHO14b] autoriai drono nusileidimo vietos žymekliui naudojo dviejų skirtingų splavų apskritimus. Tyrėjai nenaudojo simuliacinės aplinkos ir darė eksperimentus iš karto su tikru dronu. Drono valdymui naudojo ControlTower programinės įrangos sprendimą. Darbe jie parodė, kaip panaudojus papildomą valdymo modulį, galima pagerinti bepiločio orlaivio nusileidimo procedūrą naudojant net ir tokius paprastus nusileidimo taško žymeklius.

Viena iš palyginamųjų žymeklių schemų yra ARToolkit. Autoriai [MPT02] tyrė ARToolkit palyginamųjų žymeklių panaudojimą atstumo nustatymui. Jie nenaudojo bepiločių orlaivių, panaudodami prie kompiuterio pajungtą kamerą, jie nustatinėjo kaip tiksliai šis palyginamųjų žymeklių sprendimas leidžia nustatyti atstumą. Tyrėjai nustatė jog atstumo nustatymo tikslumui įtakos turi ne tik atstumas iki palyginamojo žymeklio, bet ir kampas iš kurio markeris yra filmuojamas.

Panaudodami palyginamuosius žymeklius tyrėjai [CWE15] pademonstravo sprendimą, kai dronas autonomiškai nusileidžia ant jūroje plaukiančio laivo denio. Tyrėjai pabrėžė, kad kariniams bepiločiams orlaiviams svarbu būti nepriklausomiems nuo GPS signalo ar komunikacijos su laivu. Jiems pavyko įgyvendinti drono nusileidimą, be prieš tai išvardintų infrastruktūros

priklausomybių. Šis eskperimentas buvo atliktas su AprilTag palyginamaisiai žymekliais kurie yra alternatyva ARToolkit žymekliams.

Panaudoję ROS karkasą, AR.Drone bepilotį orlaivį ir ALVAR palyginamuosius žymeklius [MJ15] pademonstravo, kad galima sukurti sistemą, kuri kybotų virš palyginamojo žymeklio ir sektų jį kai jis juda.

ArUco yra viena iš populiaresnių palyginamųjų žymeklių schemų. Straipsnyje [SK17] atliko tyrimus su AR.Drone bepiločio orlaivio platforma ir ArUco palyginamaisiais žymekliais. Pametus žymeklį dronas naudojo inercinių matavimų bloką, kad judėtų link nustatyto tikslo. Tyrėjai pastebėjo, kad nors ši bepiločio orlaivio platforma yra sąlyginai pigi, ji atliko visus jai keliamus reikalavimus ir susidorojo su užduotimis.

Nors ROS karkasas turi geras integracijas su Gazebo simuliacine aplinka, galima naudoti ir alternatyvias aplinkas. Tyrėjai [OKV15] panaudojo ROS karkasą ir V-REP simuliacinę aplinką drono autonominio nusileidimo ant ArUco palyginamojo žymeklio tyrimams. Drono judesių sprendimus valdė „fuzzy“ logikos moduliai. Šiame eksperimente pirminiai drono valdymo algoritmo tyrimai buvo atlikti simuliacinėje aplinkoje. Patikrinus jog algoritmas konceptualiai veikia, sekantys tyrimai buvo vykdomi su tikru dronu lauke. Tyrėjams pavyko su skrendančiu dronu sekti ant žemės judančią platformą ir galiausiai ant jos nusileisti.

Nusileidimo vieto žymėjimui galima naudoti ir alternatyvius būdus. [MYW+13] naudojo infraraudonųjų spindulių kamerą nusileidimo taško žymekliams aptikti. Žymekliams naudojo spalvotus teniso kamuoliukus, „H“ nusileidimo vietos žymeklį ir infraraudonųjų spindulių švyturius. Tyrėjai nustatė, kad jų valdymo algoritmas greičiausiai veikė su infraraudonųjų spindulių švyturiais, deja jie nėra tinkami naudoti lauke, nes tiesioginė saulės šviesa trukdo juos aptikti.

Tiriant nusileidimą ant judančio objekto [Fow16] naudoja AR.Drone bepiločio orlaivio paltformą ir Gazebo simuliacinę aplinką drono nusileidimui and judančio roboto su ALVAR žymekliu. Modulių apjungimui autorius naudoja ROS karkasą. Autorius nebando atlikti tyrimų realybėje su tikru dronu, eksperimentai atliekami tik simuliacinėje aplinkoje.

Atlikdami eksperimentus autoriai [RS15] panaudojo AR.Drone bepiločio orlaivio platformą ir ARToolkit nusileidimo vietos žymeklius. Nėra žinoma ar jie naudojo ROS karkasą komponentų apjungimui. Autoriai bandė panaudoti šį sprendimą drono nusileidimui ant jį pakrauti galinčios platformos, bet susidūrė su problema, kad dronas negali taip tiksliai nusileisti ant palyginamojo žymeklio kaip reikalauja ši užduotis. Norint droną pakrauti jis turi tiksliai nusileisti ant pakrovimo jungčių.

Išanalizavus mokslinius straipsnius pastebėta, kad AR.Drone platforma yra populiarus sprendimas norint atlikti tyrimus [KVF+11, Fow16, SKF+12, RS15, LZ13, MJ15] susijusius su bepiločiais orlaiviais, turint ribotą biudžetą. Šis dronas yra sąlyginai nebrangus ir turi programavimo sąsają, per kurią jam galima siųsti valdymo komandas. Vienas populiariausių karkasų robotikos užduotims spręsti yra ROS (Robot Operating System). Jis dažnai naudojamas ir moksliniuose tyrimuose [MJ15, RS15, OKV15]. Bepiločio orlaivio valdymo užduočiai spręsti buvo pasirinktas būtent šis karkasas, nes jis turi geras integracijas su AR.Drone bepiločiu orlaiviu ir yra aktyviai palaikomas atviro kodo bendruomenės. Bepiločio orlaivio nusileidimo užduočiai spresti dažniausiai naudojami palyginamieji žymekliai, kurie leidžia dronui iš aplinkos išskirti nusileidimo vietą ir apskaičiuoti atstumą iki jos. Išanalizuoti straipsniai nusileidimo taškams žymėti naudoja ArUco [OKV15, SK17], AprilTag [CWE15], ALVAR [Fow16], ARToolkit [RS15] palyginamuosius žymeklius. Kiekviena iš šių žymeklių technologija ne tik skiriasi vizualiai, bet jiems aptikti naudoja skirtingus algoritmus ir yra įgyvendinta skirtinguose programinės įrangos sprendimuose. Atlikdami bepiločio orlaivio nusileidimo ant pakrovimo stotelės tyrimą autoriai [RS15] naudojo ARToolkit palyginamojo žymeklio srendimą skirtą papildytos realybės uždaviniams spręsti. Tyrėjams nepavyko sėkmingai įgyvendinti savo tikslo, kadangi dronas turėjo nusileisti ant pakrovimo jungčių, poros centimetrų tikslumu. Naudojant šią bepiločio orlaivio AR.Drone ir ARToolkit palyginamojo žymeklio kombinaciją ir atliekant autonominį drono nusileidimą, jiems nepavyko pasiekti tokio didelio tikslumo, todėl dronas sėkmingai nenusileido ant pakrovimo stotelės jungčių. Tačiau autoriai savo išvadose teigia, kad toks sprendimas gali būti naudojamas bepiločiams orlaiviams kurie pristato prekes. Todėl šio darbo tikslui pasiekti ir buvo pasirinkta ARToolkit palyginamųjų žymeklių technologija.

Taigi išanalizavus visus šiuos straipsnius galima teigti, kad yra reikalingas atviro kodo programinės įrangos sprendimas, naudojantis nusileidimo vietos palyginamųjų žymeklių technologiją ir panaudojantis ROS robotikos karkasą kitų sistemos komponentų apjungimui, įgalinantis sukurti autonominio bepiločio orlaivio valdymo sistemą. Ši sistema leistų kitiems tyrėjams ją panaudoti atliekant įvairius eksperimentus, bei pagalbėtų kuriant kitas bepiločių orlaivių valdymo sistemas.

2. Kvadkopteris

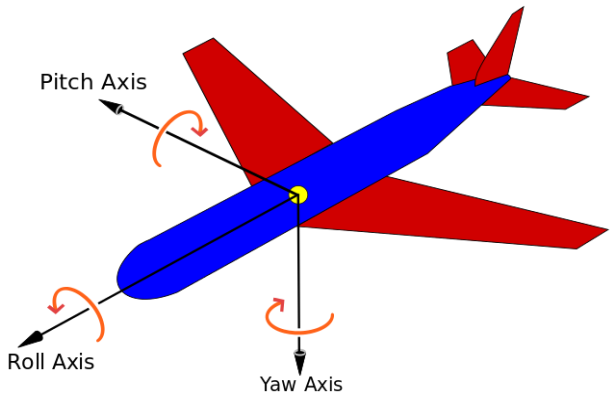
Iš pradžių apžvelkime norimo valdyti bepiločio orlaivio fizinę konfigūraciją - kvadkopterį. Kvadkopteris, taip pat vadinamas ketursraigčiu malūnsparniu arba ketursraigčiu, tai daugiarotorinis malūnsparnis, kurio keliamąją bei varomąją jėgą sukuria keturi varikliai su sraigtais. Kvadkopteriai priklauso sraigtasparnių klasei. Kvadkopteriai dažniausiai naudoja dvejas propelerių poras. Viena sukasi palei laikrodžio rodyklę, o kita - prieš laikrodžio rodyklę. Kvadkopteris yra valdomas, kiekvienam jo sraigtiniam varikliui keičiant sukimosi greitį. Kvadkopteriai skiriasi nuo malūnsparnių, kurie turi sraigtus, keičiančius menčių nuolydžio kampą. [Buc16]

Ankstyvosiomis savo gyvavimo dienomis kvadkopteriais buvo numatoma išspręsti problemas, kurias turėjo vertikalaus skrydžio orlaiviai. Kvadkopteriai neturi vieno sraigto sukimo momento valdymo problemų. Jiems nereikia galinio sraigto, kuris nekuria keliamosios galios, jų mentės trumpos, todėl yra paprastesnės konstrukcijos. Ankstyvieji kvadkopterių prototipai nebuvo efektyvūs ir reikalavo daug pastangų iš pilotų. Elektronikos pažangos dėka, vėlyvais du tūkstantaisiais metais, atsirado pigesni ir lengvi skrydžio valdikliai, akselerometrai, GPS sistemos ir kameros. Šis technologinis progresas leido naudoti kvadkopterio konfigūraciją bepiločiams orlaiviams. Jie yra maži ir manevringi, todėl gali būti naudojami tiek lauke, tiek pastatų viduje.

Kvadkopteriai yra pigesni ir labiau patvarūs nei malūnsparniai, nes yra mechaniškai paprastesni. Jų mentės yra mažesnės, todėl jos turi mažiau kinetinės energijos ir gali pridaryti mažiau žalos. Žalos potencialą galima dar labiau sumažinti naudojant apsaugas aplink kvadkopterių sraigtus. [Buc16]

2.1. Kvadkopterio veikimo ir skrydžio principas

Valdant kvadkopterį, svarbu suprasti, kaip šis multirotorinis orlaivis juda ir kokie fizikos dėsniai veikia jam skrendant. Kvadkopterio judėjimo esmė yra jo varikliukų sukimosi greityje.

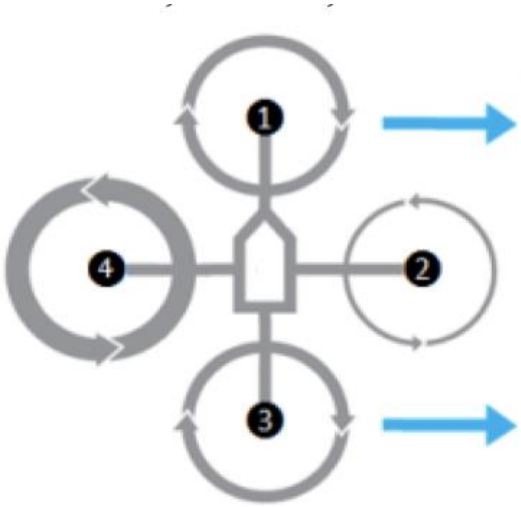


1 pav. Orlaivio polinkio (pitch), posvyrio (roll), pokrypio (yaw) ašys. [Wik18]

Keisdami drono varikliukų sukimosi greitį mes galime pakeisti kiekvieno sraigto keliamąją galią, taip priversdami kvadkopterį kilti arba leistis žemyn arba suktis apie vieną iš savo ašių: polinkio (pitch), posvyrio (roll), pokrypio (yaw). Šių ašių vizualinę reprezentaciją matome 1 paveikslėlyje. [Buc16]

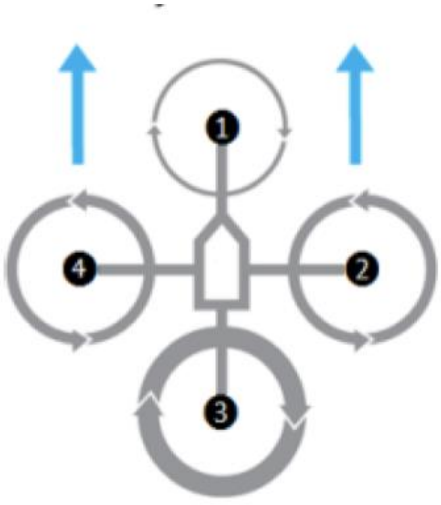
2.1.1. Posvyris ir polinkis

Drono posvyrio kampas nusako, kaip orlaivis yra pasvyręs į šonus. Sukimąsi pagal posvyrio ašį geriausiai galima būtų įsivaizduoti, kaip galvos palenkimą prie vieno iš savo pečių. Pokytis posvyryje priverčia droną judėti į šonus (2 pav.). [Buc16]



2 pav. Kvadkopterio posvyrio judesys. [RPR14]

Drono polinkio kampas nusako, kaip orlaivis yra palinkęs į priekį arba atgal. Sukimąsi pagal polinkio ašį geriausiai galima būtų įsivaizduoti kaip galvos pakreipimą bandant žūrėti į viršų arba žemyn. Pokytis polinkyje priverčia droną judėti į priekį arba atgal (3 pav.). [Buc16]

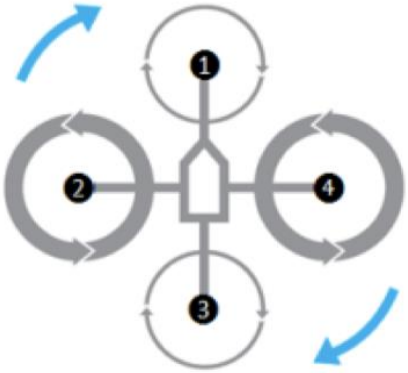


3 pav. Kvadkopterio polinkio judesys [RPR14]

Norint priversti kvadkopterį suktis apie posvyrio arba polinkio ašis reikia sraigtus viename jo krašte priversti suktis lėčiau nei kitame krašte. Kitaip tariant, vienas kvadkopterio kraštas turės daugiau keliamosios galios nei kitas ir kvadkopteris pasvirs arba palinks. Pavyzdžiui, norint orlaivį pasvirti į dešinę (pasukti apie posvyrio ašį pagal laikrodžio rodyklę) reikia, kad abu sraigtai kairėje kvadkopterio pusėje suktųsi greičiau už sraigtus dešinėje pusėje. Kairė pusė turės daugiau keliamosios galios už dešinę pusę ir orlaivis pasvirs. Panašiai, norint kvadkopterį palinkti žemyn (pasukti apie polinkio ašį pagal laikrodžio rodyklę), reikia, kad abu sraigtai galinėje orlaivio dalyje suktųsi greičiau nei sraigtai priekinėje kvadkopterio dalyje. Taip orlaivis pasvyra panašiai, kaip žmogaus galva žiūrint žemyn. [Buc16]

2.1.2. Pokrypis

Drono pokrypio kampas nusako jo azimutą, kitaip tariant orlaivio sukimąsį kai jis horizontalus žemės paviršiui. Sukimąsi pagal pokrypio ašį geriausiai galima būtų įsivaizduoti kaip galvos gestą reiškiantį "ne". Valdyti kvadkopterio sukimąsi apie pokrypio ašį yra sudėtingiau nei sukimąsi apie posvyrio ar polinkio ašis. Pirma reikėtų apžvelgti, kaip kvadkopteris stengiasi išvengti sukimosi apie pokrypio ašį. Kuriant ir konstruojant kvadkopterius, varikliukai yra konfigūruojami taip, kad jie suktųsi priešinga kryptimi nei jo kaimynai. Kitaip tariant, pradedant kvadkopterio priekiniu kairiu varikliuku ir iš eilės einant per kitus orlaivio varikliukus, jų sukimosi kryptys kaitaliotųsi: pagal laikrodžio rodyklę, prieš laikrodžio rodyklę, pagal laikrodžio rodyklę, prieš laikrodžio rodyklę. Tokia sukimosi konfigūracija naudojama norint neutralizuoti sraigtų tendenciją priversti kvadkopterį suktis. Pagal fizikos dėsnius, kai sraigtas pradeda suktis pagal laikrodžio rodyklę, galioja judesio kiekio momento tvermės dėsnis, todėl kvadkopteris suksis prieš laikrodžio rodyklę. Tai nusako Niutono trečiasis dėsnis, "jei vienas kūnas, kokio nors dydžio jėga, paveikia kitą kūną, tai tas kitas kūnas pirmąjį taip pat paveikia tokio pat dydžio priešingos krypties jėga". Kitaip tariant, kvadkopteris suksis į priešingą pusę nei sraigtų sukimosi kryptis. Todėl panaudojus konfigūraciją, kai kvadkopterio sraigtų poros sukasi į priešingą pusę, šis efektas yra panaikinamas ir orlaivis nesisuka pagal pokrypio ašį. Jei norime, kad kvadkopteris suktųsi apie pokrypio ašį, reikia sumažinti priešingų varikliukų sukimosi greitį santykinai kitai varikliukų porai. Taip dviejų sraigtų porų judesio momentas nebebus subalansuotas ir orlaivis suksis. Kvadkopterį taip galime priversti suktis į abi puses, tiesiog sulėtindami skirtingų varikliukų porų sukimąsi (4 pav.). [Buc16]



4 pav. Kvadkopterio pokrypio judesys. [RPR14]

2.1.3. Kybojimas ir aukščio kontrolė

Išsiaiškinome, kaip veikia kvadkopterio valdymas, o dabar galime išnagrinėti paprastenį manevrą - kybojimą. Kybojimas tai orlaivio buvimas pastoviame aukštyje nesisukant ir nesvyrant į kraštus. Norint priversti kvadkopterį kyboti reikia sulabansuoti visas atoveiksmio jėgas ir varikliukų kuriama keliamoji galia turi pasipriešinti sunkio jėgai. Sunkio jėga veikianti kvadkopterį yra lygi kvadkopterio masei padaugintai iš gravitacinio pagreičio, kuris yra pastovus ant Žemės paviršiaus. Ketursraigčio sukuriama keliamoji galia yra lygi visų jo sraigtų keliamosios galios sumai. Taigi, jei sunkio jėga bus lygi visų varikliukų keliamajai jėgai - kvadkopteris kabės pastoviame aukštyje. Norint pakilti arba nusileisti reikia pakeisti jėgų balansą. Jei kvadkopterio sukuriama keliamoji galia yra didesnė už sunkio jėgą, orliavis kils aukštyn. Jei orlaivio sukuriama keliamoji galia bus mažesnė už sunkio jėgą, orlaivis leisis žemyn. [Buc16]

2.1.4. Judėjimas

Mes apžvelgėme, kaip, keisdami varikliukų greičius, priverčiame kvadkopterį palinkti arba pasvirti. Kvadkopteris pradeda judėti tik palinkęs arba pasviręs. Pavyzdžiui, kvadkopteriui pasvirus į priekį (pasisukus apie posvyrio ašį pagal laikrodžio rodyklę) jis pradeda judėti į priekį. Jis pradeda judėti į priekį, nes dalis keliamosios galios yra nukreipiama horizontaliai, kitaip nei kybant, visa keliamoji galia yra nukreipta žemyn. Suprantama, kad, panaudojus dalį žemyn nukreiptos kvadkopterio traukos jėgos horizontaliam judėjimui, orlaivis judėdamas praras savo aukštį. Tam dronų skrydžio valdikliai turi "aukščio išlaikymo" funkcionalumą, kuris atitinkamai pritaiko varikliukų greičius, norint kompensuoti prarastą jėgą ir dronas galėtų skristi išlaikydamas pastovų aukštį. [Buc16]

2.2. AR.Drone platforma

Šiame skyriuje apžvelgsime platformą, kuriai kursime drono valdymo modulį. AR.Drone yra nuotoliniu būdu valdomas kvadkopteris, kurį gamina Prancūzų kompanija Parrot. Dronas yra valdomas iš iOS ir Android aplikacijų per WiFi. Jo gamintojai išleido atvirą API (application programming interface) drono valdymui, todėl jį galimą valdyti iš bet kurios aplikacijos, kuri implementuoja šią aplikacijų programavimo sąsają. Dėl šios atviros API sąlyginai žemos kainos ir didelio sensorių kiekio AR.Drone tapo populiariu tyrimų ir mokslo įrankiu. Ši platforma yra naudojama autonominės navigacijos, autonominio sekimo ir žmogaus-mašinos interakcijos eksperimentuose. Atliekant tyrimus šiose srityse buvo sukurtos aplikacijos, kai kurios iš jų atviro kodo. Šios aplikacijos praplėtė pirminį drono funkcionalumą. Prancūzijoje AR.Drone buvo išbandytas Specialiųjų Pajėgų, atliekant oro žvalgybą. [BCV+11, LZ13]

5 pav. AR.Drone kvadkopteriai.

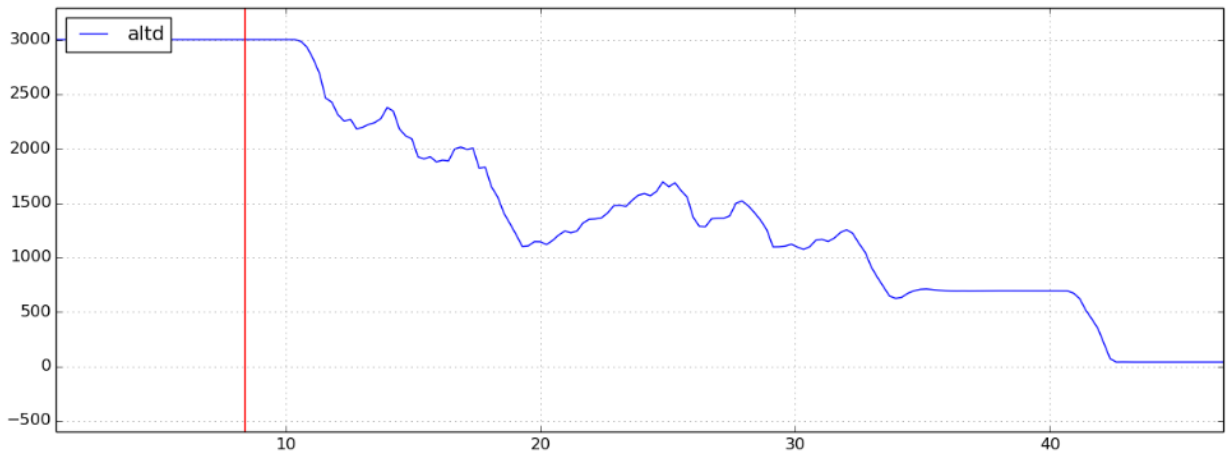
Drono karkasas pagamintas iš nailono ir anglies pluošto (5 pav.). Komplektacijoje yra du korpusai. Vienas korpusas pagamintas iš polipropileno ir skirtas skraidymui patalpose. Jis apgaubia menčių perimetrą, taip jas apsaugodamas. Kitas korpusas - pagamintas iš lengvo plastiko. Jis yra didesnio manevringumo ir yra skirtas skraidymui lauke. Dronas turi šešis laisvės laipsnius. Karkaso viduje sumontuoti jutikliai ir borto kompiuteris. Tarp jutiklių yra ultragarsinis aukštimatis, kuris veikia iki 6 metrų aukštyje ir padeda stabilizuoti droną vertikaliai. Drono stabilizacijai taip pat yra imontuotas IMU (Inertial Measurement Unit), kuris matuoja kvadkopterio polinkį, posvyrį ir pokrypį. Stabilizacijai dideliame aukštyje dronas naudoja slėgio

sensorių. Kvadkopterio borto kompiuteris naudoja Linux operacinę sistemą ir komunikuoja per Wi-Fi prieigos tašką su valdymo įrenginiu. Standartiškai valdymo įrenginiais gali būti iOS ir Android operacines sistemas naudojantys telefonai ir planšetės. Drono API (application programming interface) yra atviras, jį galima valdyti visais įrenginiais, kurie turi Wi-Fi ir kuriuose veikia aplikacijos, įgyvendinančios AR.Drone aplikacijų programavimo sąsają. [BCV+11, LZ13]

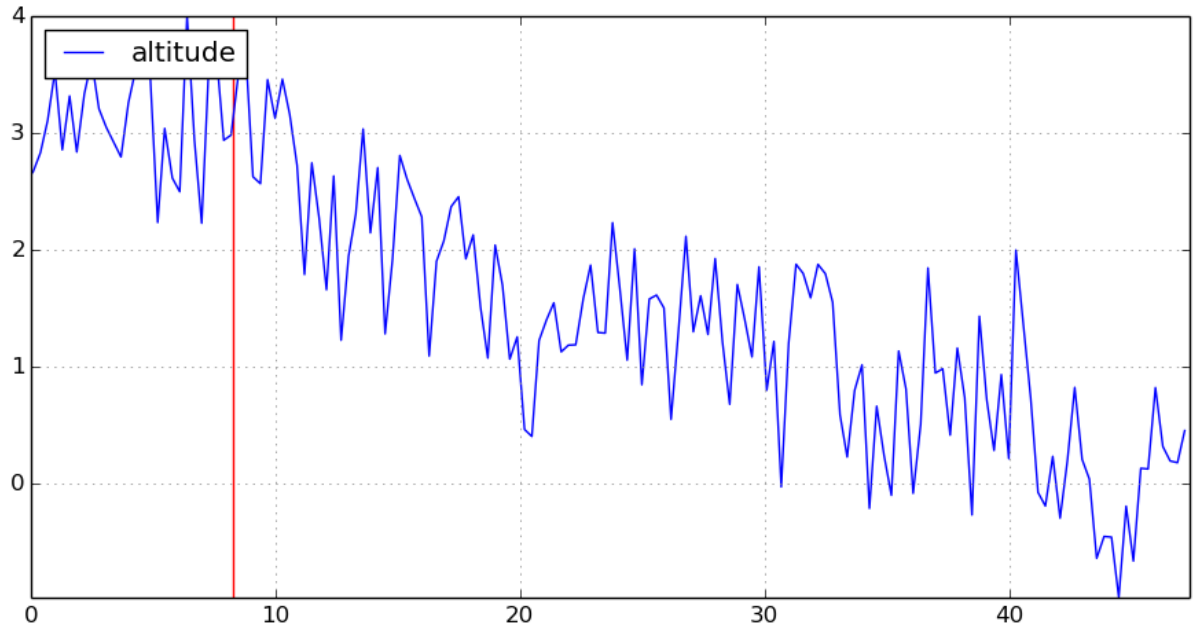
Sraigtams galią tiekia 15 vatų varikliukai be kolektorių, kuriuos maitina 11.1 voltų ir 1500 mAh ličio polimerų baterija. Su šia baterija dronas gali 12 minučių skristi 5 m/s greičiu. Dronas turi 720p rezoliucijos priekinę vaizdo kamerą, kuri gali filmuoti 30 kadrų per sekundę greičiu ir apatinę QVGA raiškos kamerą, kuri filmuoja 60 kadrų per sekundę greičiu. [BHO14a]

2.3. Skirtingi aukščio matavimo jutikliai

AR.Drone bepilotis orlaivis savo aukščiui nustatyti naudoja kelių skirtingų tipų jutiklius. Pirmasis jutiklis, tai sonaras, kuris naudoja ultragarso bangas atstumui nustatyti. Šis sensorius yra imontuotas drono apačioje ir matuoja drono atstumą iki žemės, iki 3 metrų aukščio (6 pav.). Didesniam aukščiui nustatyti dronas naudoja altimetrą. Šis įrenginys matuoja aukštį pagal barometrinį spaudimą. Kuo mažesnis atmosferos slėgis, tuo dronas yra pakilęs aukščiau. Nors gali pasirodyti, kad toks sensorius turėtų būti labai netikslus, sąlyginai nedideliems aukščiams, lyginant su lėktuvais. Tačiau iš grafiko (7 pav.) matome, kad jis yra pakankamai tikslus apytiksliam drono aukščiui nustatyti. Abu šiuos sensorius mes galime sėkmingai simuliuoti su sukurtu programinės įrangos sprendimu.



7 pav. Sonaro aukščio parodymai. X ašyje – laikas sekundėmis, Y ašyje - drono aukštis milimetrais.



6 pav. Altimetro aukščio parodymai. X ašyje – laikas sekundėmis, Y ašyje - drono aukštis metrais.

3. ROS karkasas

Šiame skyriuje apžvelgsime karkasą, kuriam bus kuriamas drono valdymo modulis. Robot Operating System (toliau – ROS) yra programinės įrangos karkasų rinkinys, skirtas palengvinti robotų programinės įrangos kūrimą. Nors ROS savo pavadinime turi žodžius "operacinė sistema", bet iš tikro tai nėra operacinė sistema. ROS tiesiog palengvina programinės įrangos kūrėjo darbą, tiekdama techninės įrangos abstrakcijos, įrenginių kontrolės, tarpprocesinio žinučių perdavimo ir programinės įrangos paketų valdymo servisus. ROS nėra realiojo laiko operacinė sistema, bet yra įmanoma ROS integruoti su realiojo laiko kodu. Antrojoje ROS versijoje realaus laiko kodas yra geriau palaikomas [Kou19, Len18]. ROS programinę įrangą būtų galima išskirti į tris dalis:

- įrankiai, skirti ROS pagrindu sukurtos programinės įrangos platinimui;
- ROS kliento implementacijos skirtingoms programavimo kalboms;
- ROS programinės įrangos paketai.

Privalumas yra tas, kad su tuo pačiu kodu ROS platformoje galima valdyti skirtingus robotus. Parašius modulį, išsprendžiantį tam tikrą problemą vieną kartą, jis gali būti dar kartą panaudojamas skirtingose robotų platformose. ROS turi daug vizualizacijos ir simuliacijos modulių. Jie tausoja roboto techninę įrangą ir pagreitina programinės įrangos vystymą. Pirmąsias ROS versijas sukūrė robotikos inkubatorius - Willow Garage. Kompanija Willow Garage yra viena iš garsiausių robototechnikos laboratorijų visame pasaulyje. Kasmet yra išleidžiamos atnaujintos ROS versijos. [Kou19]

Sekančiuose poskyriuose apžvelgsime pagrindinius ROS karkasą sudarančius komponentus.

3.1. ROS skaičiavimų grafas

ROS skaičiavimų grafas tai lygianaris (P2P) ROS procesų, kurie kartu apdoroja informaciją, tinklas [Kou19]. Šis tinklas susideda iš tokių dalių:

- Mazgai (Nodes)
- Šeimininkas (Master)
- Parametrų serveris (Parameter server)
- Žinutės (Messages)
- Temos (Topics)
- Servisai (Services)

Toliau apžvelgsime svarbiausius komponentus iš šio sąrašo.

3.2. ROS Mazgas

ROS mazgas (node) - tai procesas, kuris atlieka skaičiavimus. Kitaip tariant, ROS karkasui sukurta programa yra ROS mazgas. Robotą valdanti sistema dažniausiai susideda iš daugelio mazgų. Kiekvienas mazgas atsako už savo funkcinę sritį, pavyzdžiui, vienas mazgas valdo roboto varikliukus, kitas - atsakingas už lokalizaciją, o trečias už kelio planavimą. [Kou19, Len18]

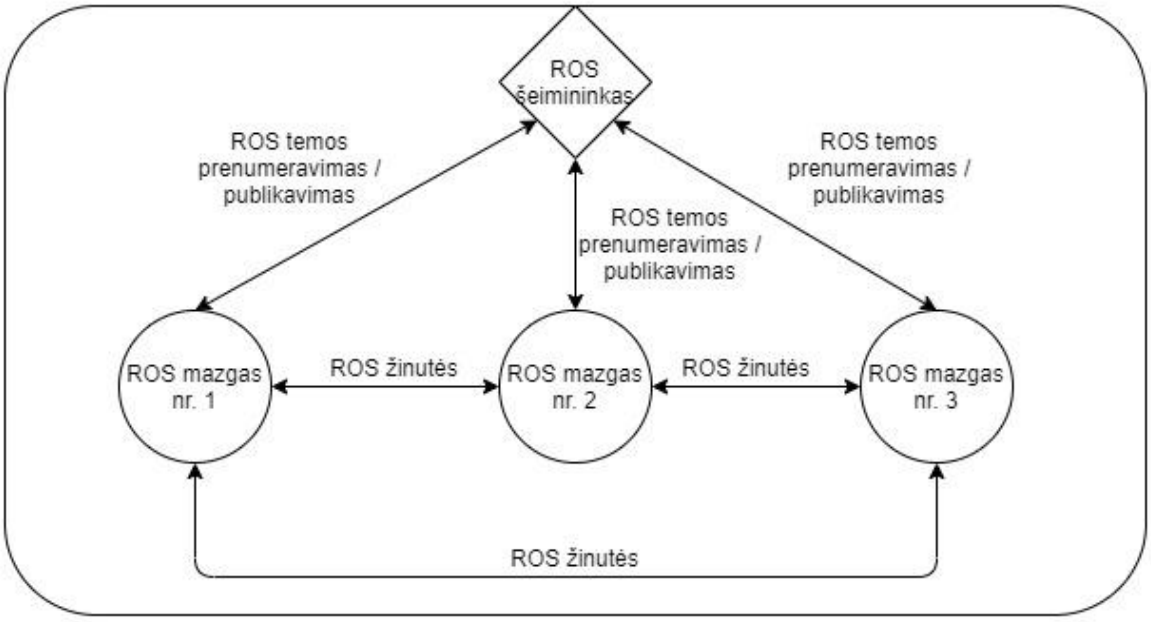
ROS mazgams rašyti yra sukurtos ROS kliento bibliotekos (client library). ROS kliento biblioteka leidžia programuotojui iš kodo pasiekti ROS skaičiavimų grafo sudedamąsias dalis. Šios bibliotekos įgalina kurti mazgus ir servisus, paskelbti ir prenumeruoti temas, naudotis parametrų serveriu. Dažniausiai yra palaikomos C++ ir Phyton programavimo kalbos, bet yra sukurta keliolika eksperimentinių kliento bibliotekų kitoms programavimo kalboms, tokioms kaip C#, Go, Haskell, Java, Javascript, Ruby. [Kou19, Len18]

3.3. ROS Šeimininkas

ROS Šeimininkas (Master) tai XMLRPC serveris. Jis tiekia vardų davimo, paieškos ir registracijos servisus ROS mazgams. Jis seka ROS temų leidėjus ir prenumeratorius, bei servisus. Pagrindinis ROS Šeimininko uždavinys - padėti mazgams rasti vienas kitą. Mazgams vienas kitą suradus jie toliau bendrauja lygianariame tinkle (peer-to-peer). Šeimininkas taip pat leidžia mazgams naudotis parametrų serveriu. Parametrų serveris - tai centralizuota vieta, kur mazgai gali saugoti duomenis. Iš esmės, tai per tinklą pasiekiamas žodynas. [Kou19, Len18]

3.4. ROS Žinutės

ROS sistemoje mazgai bendrauja tarpusavyje paskelbdami žinutes temose. Žinutė (message) tai paprasta duomenų struktūra, susidedanti iš tipizuotų laukų. ROS palaiko standartinius primityviuosius duomenų tipus: sveikojo skaičiaus (integer), slankiojo kablelio (float), Būlio (boolean) ir kitus. Yra palaikomi ir primityviųjų tipų duomenų masyvai. ROS žinutės palaiko įdėtines struktūras. Mazgai gali apsikeisti užklausomis ir atsakyti žinutėmis, iškviesdami ROS servisą. [Kou19, Len18]



8 pav. ROS šeimininko ir ROS mazgų bendravimo schema.

3.5. ROS Temos

ROS tema (topic) - tai griežtai tipizuota žinučių magistralė, kurioje mazgai apsikeičia žinutėmis. ROS žinutės yra nukreipiamos naudojant publikavimo/prenumeravimo semantiką. Mazgas išsiunčia žinutę, publikuodamas ją ROS temoje (8 pav.). Tema, tai ROS vardas, kuris naudojamas žinutės identifikavimui. Mazgas, kurį domina tam tikro tipo duomenys, turi prenumeruoti temą, kurioje to tipo duomenys yra publikuojami. Kiekviena tema gali turėti ne vieną leidėją ir prenumeratorių. Vienas mazgas gali publikuoti, bei prenumeruoti ne vieną temą. Publikatoriai ir prenumeratoriai nežino apie vienas kito egzistavimą. Tai padaryta, norint atsieti informacijos sukūrimą nuo vartojimo. [Kou19, Len18]

3.6. ROS Servisai

Publikavimo/prenumeravimo modelis yra labai lanksti komunikavimo priemonė. Šiame modelyje mazgai yra susiję abipus daugiareikšmiu sąryšiu ir duomenys yra siunčiami tik į vieną pusę. Toks modelis nėra tinkamas užklausos ir atsako interakcijoms, kurios yra dažnai reikalingos tokiose paskirstytose sistemose. [Kou19, Len18]

Užklausos ir atsako interakcijos yra įgyvendinamos per ROS servisus (services). ROS servisas privalo turėti aprašytas užklausos ir atsako žinučių struktūras. Vienas mazgas teikia servisą, kitas kliento mazgas gali pasinaudoti servisu, išsiųsdamas užklausos žinutę ir laukdamas atsako žinutės. Yra įmanomas ir nuolatinis ryšys tarp kliento ir serviso, tai leidžia didesnę greitaveiką paaukojant stabilumą esant serviso sutrikimams. [Kou19, Len18]

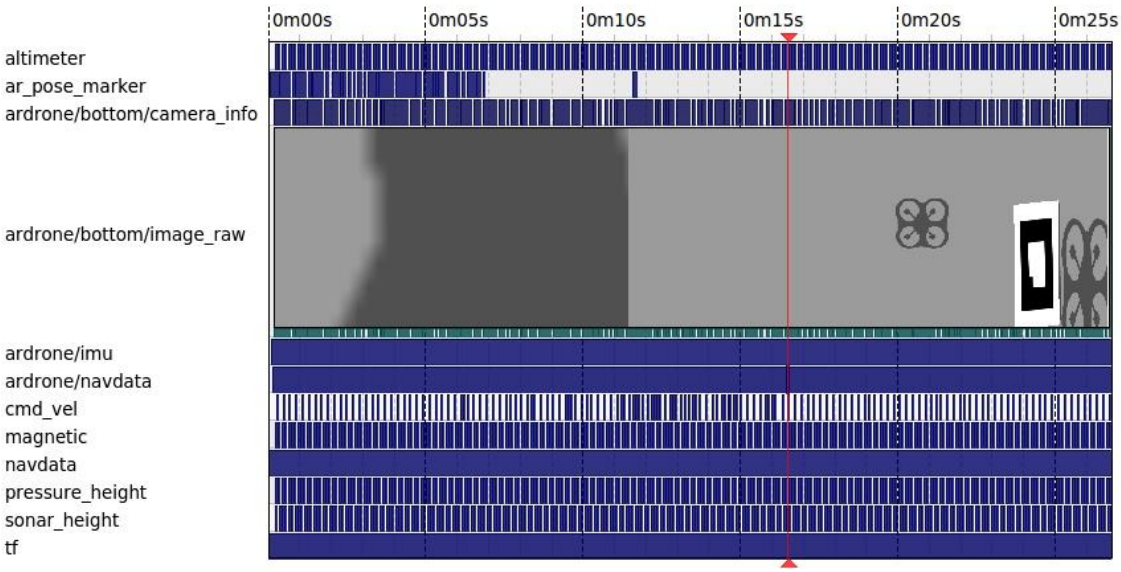
3.7. ROS žinučių saugojimo įrankis rosbag

“Bag”, tai failo formatas, skirtas ROS žinučių serializavimui ir išsaugojimui. „Rosbag”, tai įrankis leidžiantis dirbti su „bag“ tipo failais. „Rosbag“ įrankis leidžia įrašyti dominančias ROS temas. Kitaip tariant, paleidus ROS temų įrašymą su „rosbag“ įrankiu, galima įrašyti visus sistemos duomenis tuo metu publikuojamus į ROS temas ir juos išsaugoti „bag“ tipo faile. Su „rosbag“ įrankiu mes galime įrašinėti nevisas tuo metu sistemoje publikuojamas temas, bet galime selektyviai įrašinėti tik mus dominančias.

Kai kurios temos mums gali būti nesvarbios, dubliuoti duomenis arba tiesiog užimti per daug duomenų srauto. Puikus pavyzdys, kai kelios drono kameros ir temos, publikuojančios vaizdus ne tik iš abiejų kamerų vienu metu, bet ir publikuojančios vaizdą su papildomais filtrais. Mes dažnai norime įrašinėti tik ROS temas su vaizdais iš kameros, kurią tuo metu naudoja drono valdymo algoritmas.

„Bag“ tipo failuose įrašytas ROS žinutes galima atkurti ir paleisti tuo pačiu eiliškumu, kaip jos buvo įrašytos. Tiesa, žinutes atkuriant, jas galima priskirti kitoms temoms, nei jos buvo įrašytos originaliai. ROS žinutės, atkuriamos iš „bag“ failų į ROS skaičiavimo grafą, iš esmės nesiskiria nuo ROS žinučių, išsiųstų iš veikiančių ROS mazgų. Vienintelė vieta, kuri gali reikalauti papildomo apdorojimo, yra žinutės, kuriose yra išsaugoti kintamieji su laiko formato tipu. Dėl šios priežasties ROS sistema turi „/clock“ temą. Ji įgalina visiems ROS mazgams veikti pagal sisteminį laiką, kuris gali būti publikuojams tiesiogiai arba atkuriamas iš „bag“ tipo failų. Jei sistemoje veikaintys ROS moduliai darbui su laiku naudoja ROS API, jie turėtų veikti gerai, tiek paleidus žinutes iš „bag“ failo, tiek jas transliuojant iš kitų ROS mazgų.

Šis formatas yra našus žinutes išsaugant ir atkuriant, nes jos yra serializuojamos tokiu pačiu formatu, kaip ir būtų perduodamos tarp ROS modulių per tinklą.



9 pav. Įrankis rqt_bag, skirtas bag failams atkurti ir analizuoti.

„Bag“ tipo failai, pagrindinis būdas ROS sistemoje duomenų žurnalizavimui. Tyrėjai naudoja „bag“ failų ekosistemą duomenims įrašyti, juos vizualizuoti, kategorizuoti ir išsaugoti vėlesniam panaudojimui.

Norint įrašyti duomenis į „bag“ failą naudojama „rosbag record“ komanda. Ji gali įrašyti visas ROS skaičiavimo grafo temas, šiuo atveju „-a“ parametras įrašys visas temas:

```
rosbag record -a
```

Mes galime specifikuoti, kurias temas mes norime įrašinėti. Šiuo atveju mes įrašytumėme žinutes, publikuotas į „/ardrone/navdata“ ir „/altimeter“ temas:

```
rosbag record /ardrone/navdata /altimeter
```

Norint greitai suprasti, kokių ROS mazgų žinutės yra išsaugotos „bag“ faile, mes tai galime lengvai padaryti su „rosbag info“ komanda, kuri leis pamatyti greitąją „bag“ failo apžvalgą su publikuojamomis temomis, įrašo trukme ir žinučių kiekiu. Pavyzdys, kaip naudoti šią komandą:

```
rosbag info skrydis1.bag
```

Darbo Priede Nr. 1 yra pateikta šios komandos išeitis ją naudojant su vienu iš atliktu eksperimentu „bag“ failu.

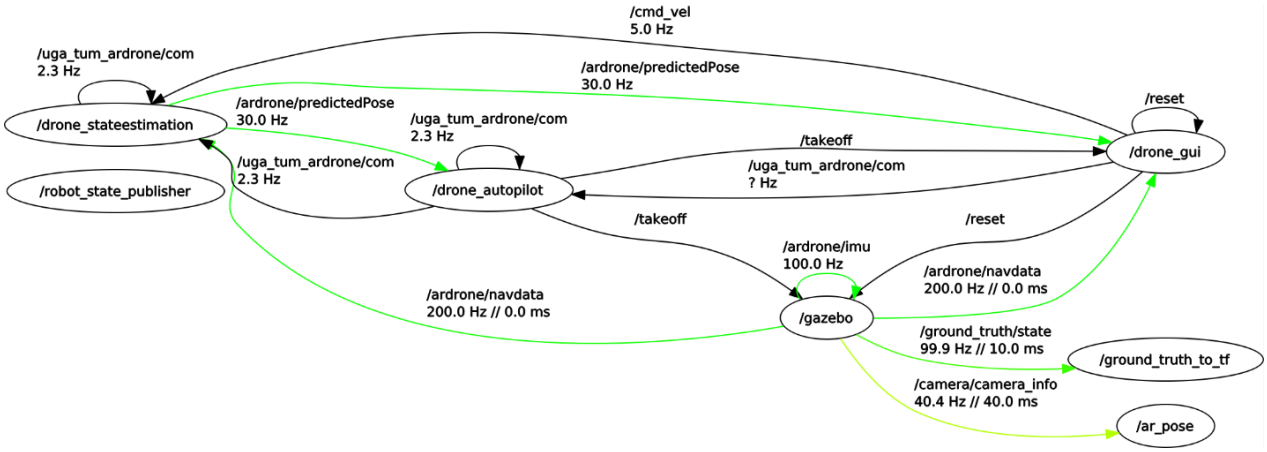
Norint atkurti ROS žinutes iš „bag“ failo, reikia panaudoti „rosbag play“ komandą. Su ja galime atkurti žinutes vienu metu net iš kelių failų:

```
rosbag play duomenys1.bag duomenys2.bag
```

Duomenis galime įrašyti ir naudoti ne tik per komandinę eilutę, bet ir per grafinį interfeisą. „Rosbag“ turi grafinio interfeiso programą, pavadintą „rqt_bag“. Kaip atrodo „bag“ failas, atidarytas su šia programa, galime matyti 9 paveikslėlyje. Su šiuo įrankiu galime iš karto atvaizduoti pasirinktų duomenų aibę grafike.

3.8. ROS skaičiaivmų grafo vizualizavimas

ROS skaičiavimų grafą (angl. computation graph) leidžia vizualizuoti rqt_graph įrankis. Kitaip tariant, jis realiu laiku atvaizduoja grafą, kuris parodo, kas šiuo metu vyksta ROS sistemoje. Šioje vizualinėje reprezentacijoje matoma, kokie ROS mazgai šiuo metu gyvuoja ir kokios ROS temos juos jungia. 10 pav. galime matyti, kaip atrodo ROS skaičiavimų grafas, paleidus Gazebo simuliaciją ir komunikacijos su dronu modulius.



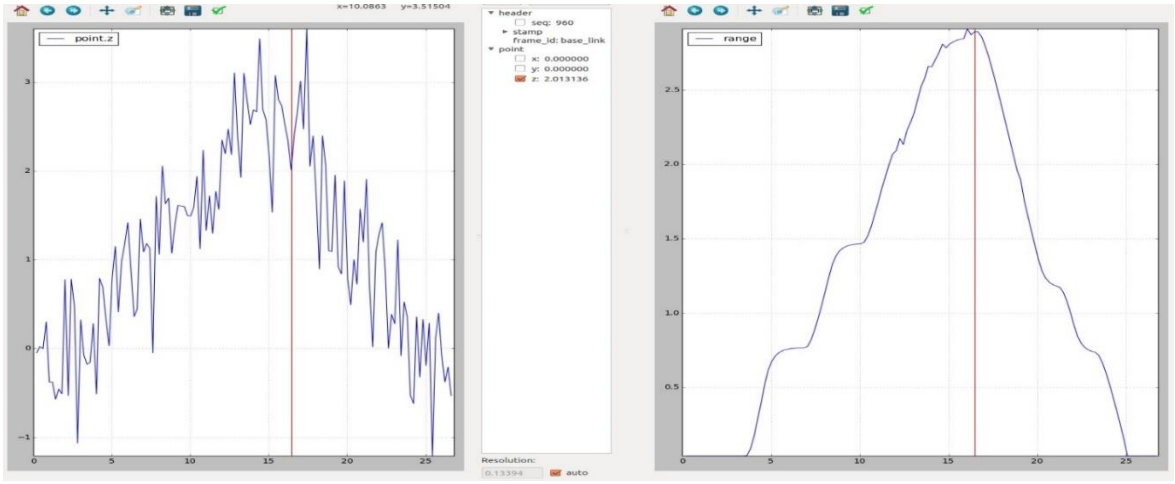
10 pav. ROS skaičiavimų grafo vizualizacija sugeneruota su rqt_graph įrankiu.

3.9. ROS karkaso įrankis grafikų generavimui

Pagalbinis ROS karkaso įrankis rqt_plot, skirtas atvaizduoti grafike bet kokias skaitines reikšmes, publikuojamas ROS temose. Tai puikus įrankis duomenų vizualizavimui iš vieno ar kelių ROS temų. Norint sugeneruoti grafiką iš sonaro aukščio duomenų, kurie yra skelbiami temoje "/sonar_height/range", reikėtų paleisti tokią komandą:

```
rqt_plot /sonar_height/range
```

Šiame mokslo tiriamojo darbo projekte galime palyginti drono aukščio duomenis, gautus iš dviejų skirtingų aukščio sensorių. Kairėje 11 paveikslėlio dalyje matome grafiką su aukščio duomenimis iš barometro, kuris nustato drono aukštį pagal atmosferos slėgį. Dešinėje 11 paveikslėlio dalyje matome grafiką su aukščio reikšmėmis iš sonaro, kuris skleidžia garso bangas nuo drono apačios link žemės ir pagal laiko tarpą, reikalingą garso bangai sugrįžti, nustato drono aukštį virš žemės.



11 pav. Įrankis rqt_plot, skirtas grafikų braižymui.

3.10. ROS paleidimo failai

"Roslaunch", tai įrankis skirtas paleisti ROS mazgus ir nustatyti parametrus parametrų serveryje. Šis įrankis gali sekti ir automatiškai iš naujo paleisti numirusius mazgus. Dažniausiai šis įrankis yra paleidžiamas paduodant konfigūracinius failus. Konfigūraciniai failai aprašomi XML formatu ir dažiausiai yra išsaugojami su ".launch" plėtiniu. Roslaunch įrankio paleidimo pavyzdys:

```
roslaunch ros_paketas konfiguracinis_failas.launch
```

Konfigūraciniai failai dažniausiai grupuoja mazgus, reikalingus tam pačiam funkcionalumui. Taip supaprastinamas funkcinių modulių paleidimas, iš karto paleidus visus mazgus, reikalingus kažkokiam ROS programinio sprendimo funkcionalumo veikimui. Konfigūraciniai failai yra patikrinami per vieną iteraciją, žymos yra patikrinamos serijiniu būdu, todėl jei yra keli kintamojo priskyrimai, bus naudojama paskutinė priskirta reikšmė. Apžvelgsime konfigūracinio failo pavyzdį. Visas konfigūracinis failas yra pridedamas, darbo priede Nr. 2.

Žemiau esanti žyma informuoja teksto redaktorius, kad šiame faile esančius simbolius reikėtų paryškinti pagal XML kalbos taisykles.

```
<?xml version="1.0"?>
```

Taip yra formatuojamas komentaras:

```
<!-- komentaras -->
```

Ši žyma indikuoja paleidimo konfigūracijos aprašo pradžią:

```
<launch>
```

Žemiau esantis pavyzdys parodo kintamųjų priskyrimą:

```
<arg name="drone_ip" default="127.0.0.1"/>
```

Su “node” žyma yra paleidžiamas ROS mazgas, šiuo konkrečiu atveju, tai yra duomenų vizualizacijos įrankis. Jis yra inicializuojamas su grafinės sąsajos konfigūracijos failu:

```
<node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find ar_pose)/launch/live_single.rviz"/>
```

Ši žyma paleidžia "static_transform_publisher" mazgą iš "tf" paketo, suteikdama šiam mazgui "world_to_cam" pavadinimą ir perduodama inicializacijos parametrus:

```
<node pkg="tf" type="static_transform_publisher" name="world_to_cam"
    args="0 0 0.5 -1.57 0 -1.57 world camera 10" />
```

"param" žyma yra skirta sukurti ir nustatyti kintamiesiems parametrų serveryje, kadangi ši žyma yra "node" žymos viduje, tai reiškia, kad kintamasis bus privatus ir bus matomas tik šiam mazgui:

```
<param name="DroneIP" value="$(arg drone_ip)"/>
```

"remap" žyma yra skirta ROS temų pervadinimui, nekeičiant mazgo išeities kodo ir neperkompiliuojant jo:

```
<remap from="/ardrone/takeoff" to="/takeoff"/>
```

Šiuo konkrečiu atveju, nors "drone_gui" mazgo išeities kode yra publikuojama į temą "/ardrone/land", mes, paleisdami šį mazgą su "remap" žyma, pakeičiame publikuojamą temą į "/land":

```
<remap from="/ardrone/land" to="/land"/>
```

“include” žyma leidžia importuoti kitą “launch” tipo konfigūracinį failą:

```
<include file="$(find cvg_sim_gazebo)/launch/empty_world.launch"/>
```

Konfigūracinis failas užbaigiamas dar viena “launch” tipo žyma su uždaromuoju simboliu priešakyje:

```
</launch>
```

Norint paleisti ROS mazgą, tuo metu sistemoje turi veikti „roscore“ procesas, kuris yra pagrindinis ROS procesas, prižiūrintis visos ROS ekosistemos darbą. Tiesa, paleidžiant mazgus su konfigūraciniais "*.launch" failais, "roslaunch" įrankis pats patikrina, ar tuo metu sistemoje yra paleistas "roscore" procesas ir, jei jis nėra aktyvus, automatiškai jį paleidžia.

Paleidimo sistema "roslaunch" yra architektūriškai pagrįsta funkcine kompozicija. Iš daugelio paprastų ROS mazgų yra sukuriamos sudėtingos sistemos.

3.11. ROS parametrų serveris

Konfigūraciniai kintamieji ROS sistemoje dažniausiai yra išsaugomi parametrų serveryje. Šis mokslo tiriamojo darbo projektas naudoja parametrų serverį globalių kintamųjų saugojimui. Parametrų serveris, tai bendras, per tinklo sąsają pasiekiamas, daugiamatis žodynas. Kintamieji parametrų serveryje gali būti pasiekiami per komandinę eilutę, ROS mazgus ir paleidimo failus. Kintamojo nustatymas ROS parametrų serveryje iš paleidimo failo atrodo taip:

```
<param name="kintamojo_vardas" value="kintamojo_reiksme" />
```

Kintamojo nustatymas ROS mazge, parašytame su C++ kalba atrodo taip:

```
void:ros::param::set(kintamojo_vardas, kintamojo_reiksme)
```

Kintamojo reikšmės nuskaitymas ROS mazge, parašytame su C++ kalba atrodo taip:

```
void:ros::param::get(kintamojo_vardas)
```

Kintamieji parametrų serveryje atitinka ROS vardijimo schemą ir turi hierarchines struktūras, kurios dažniausiai atitinka mazgų hierarchiją. Kintamųjų hierarchinės struktūros pavyzdys:

```
/kamera/kaire/pavadinimas: kaire_kamera
/kamera/kaire/ryskumas: 1.1
/kamera/desine/pavadinimas: desine_kamera
/kamera/desine/ryskumas: 0.9
```

Kintamasis “/kamera/kaire/pavadinimas” turi reikšmę “kaire_kamera”. O kintamasis “/kamera/kaire” turi reikšmę, kuri yra žodynas:

```
pavadinimas: kaire_kamera
ryskumas: 1.1
```

Kintamojo “/kamera” reikšmė šiuo atveju yra žodynų žodynas ir atrodo taip:

```
kaire: { pavadinimas: kaire_kamera, ryskumas: 1.1 }
desine: { pavadinimas: desine_kamera, ryskumas: 0.9 }
```

Kintamieji parametrų serveryje gali būti šių tipų:

- 32 bitų sveikieji skaičiai
- Būlio logikos
- Teksto
- Slankiojo kablelio skaičiai
- Datos
- Sąrašai
- Base64 formatu užkoduoti dvejetainiai duomenys

3.12. Dinaminių parametrų modulis

ROS modulis "dynamic_reconfigure" suteikia galimybę keisti mazgo parametrus bet kuriuo metu, nepaleidžiant mazgo iš naujo. Jis leidžia kai kuriuos mazgo parametrus padaryti pasiekiamus iš išorės, standartizuotu būdu. Kiti mazgai arba programos gali pateikti užklausas ir gauti dominančių parametrų pavadinimus, tipus ir galimus kintamojo nustatymo rėžius. Nustatant kintamuosius parametrų serveryje ir nepaleidžiant juos naudojančių mazgų iš naujo, nėra garantuojama jog naujos jų reikšmės bus nuskaitytos, nes parametrų serveris tėra kintamųjų repozitorija. Mazgai, naudojantys dinaminio perkonfigūravimo modulį, užtikrina, kad juose yra implementuotas perkonfigūravimo metodas. Taigi, pakeitus kintamojo reikšmę iš išorės per dinaminio perkonfigūravimo mechanizmą, mazgas privalės pakviesti savyje esantį metodą, kuris atnaujins tų kintamųjų reikšmes. Pradinės kintamųjų reikšmės yra nustatomos konfigūraciniame faile su "*.cfg" plėtiniu. Kintamojo pradinės reikšmės nustatymo pavyzdys konfigūraciniame faile:

```
param.add("agresyvumas", double_t, 0, "Drono agresyvumo daugiklis", .5, 0, 1)
```

Pirmasis parametras yra kintamojo pavadinimas sistemoje. Antrasis parametras yra kintamojo duomenų tipas. Trečiasis parametras indikuoja kintamojo grupę. Ketvirtasis parametras yra kintamojo aprašas, dažnai naudojamas grafiniuose interfeisuose. Penktasis parametras nustato kintamojo pradinę reikšmę. Šeštasis parametras nusako galimą kintamojo apatinio rėžio reikšmę, o septintasis nusako maksimalią viršutinio rėžio reikšmę.

3.13. Gazebo simuliacinė aplinka

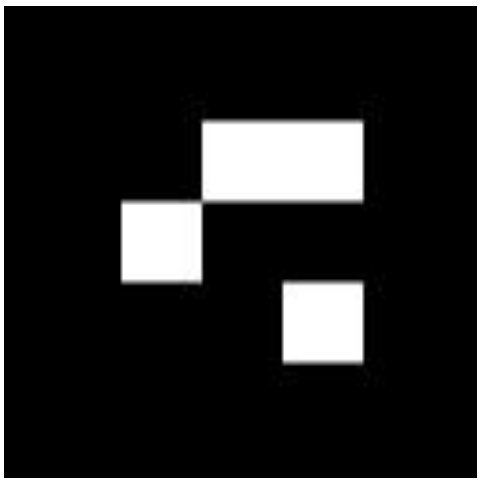
Norint testuoti algoritmus dronuose, reikia brangiai kainuojančios įrangos ir daug laiko. Klaidai įvykus drone, jis gali būti sugadintas. Norint išvengti dronų sugadinimo atvejų, galima naudoti simuliatorių. Simuliatoriai yra vienas esminių įrankių robotų kūrime. Jie padeda greitai, efektyviai ir pigiai išbandyti naujus konceptus, strategijas, konfigūracijas ir algoritmus. Gazebo yra atviro kodo projektas, kuris leidžia simuliuoti robotų veikimą ir jų aplinką. Šis projektas turi daug aktyvių projekto plėtotojų, todėl dažnai išleidžiamos vis naujos projekto versijos. Kadangi jis yra atviro kodo, žmonės jį pritaiko savo poreikiams ir tuo pačiu prisideda prie projekto plėtojimo. Šis simuliatorius gali būti naudojamas kaip tarpinė grandis tarp braižymo lentos ir realaus pasaulio. Gazebo simuliacijos įrankis matomas 12 paveikslėlyje. [KH04]

12 pav. Gazebo simuliacinės aplinkos langas [Sel19].

Gazebo simuliacinė aplinka, skirta tiksliai atkurti dinamiškas bei sudėtingas išorės ir vidaus aplinkas robotams. Visi simuliuojami objektai turi masę, pagreitį, trintį ir daugelį kitų parametrų. Tai leidžia objektams realistiškai elgtis. Robotai Gazebo simuliacijoje taip pat yra dinaminės struktūros, susideda iš standžių kūnų, sujungtų sąnariais. Gali būti simuliuojami kraštovaizdžiai, pastatai ir vartotojo sukurti objektai. Beveik visi simuliacijos parametrai yra kontroliuojami, nuo apšvietimo iki trinties koeficientų. Gazebo gali simuliuoti fizikinius procesus su keliais fizikos varikliukais. Dažniausiai naudojamas ODE - "Open Dynamics Engine", bet galima pasirinkti ir "Bullet". Gazebo įrankis turi labai didelį jau sumodeliuotų ir parengtų naudojimui robotų sąrašą. Simuliacijoje yra implementuota daugybė sensorių. Gazebo turi patogų grafinį interfeisą. Gazebo simuliatorius pritaikytas darbui su ROS karkasu. [KH04] [MSK+2012] [FBA+16]

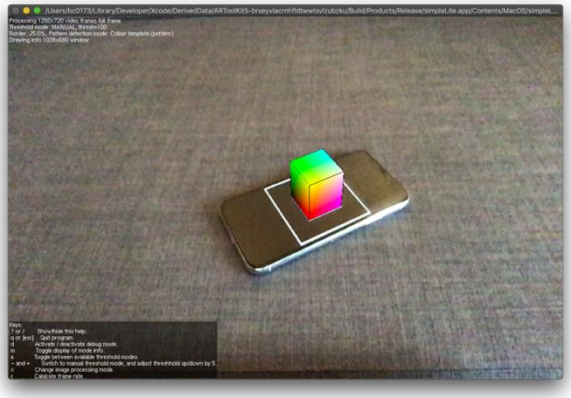
3.14. ARToolKit karkasas

ARToolKit buvo pirmoji atviro kodo platforma, skirta papildytai realybei ir objektų sekimui 3D erdvėje. Šis įrankis naudoja juodai baltus palyginamuosius žymeklius. Palyginamieji žymekliai yra panašūs į QR kodus. Juos lengva pagaminti, identifikuoti ir nuskaityti su skaitmeninėmis kameromis. Tiesa, palyginamieji žymekliai skiriasi tuo, kad, kitaip nei QR kodai, jie nėra skirti duomenų atvaizdavimui, bet yra naudojami pozicijos, orientacijos ir dydžio nustatymui jų atžvilgiu. ARToolKit palyginamojo markerio formatas matomas 13 paveikslėlyje. [KBP+00, Fia04, Was19a, Was19b]

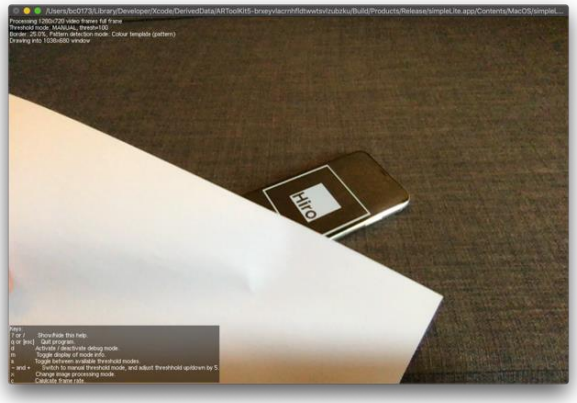


13 pav. ARToolKit lyginamojo markerio pavyzdys.

Kai kameros koordinatės yra nustatomos, mes, pasinaudoję šiuo karkasu, galime nupiešti 3D modelį ant palyginamojo markerio, sukurdami papildytą realybę. ARToolKit yra parašytas C ir C++ kalbomis. [MPT02, Lim18, Was19b]



14 pav. ARToolKit karkaso panaudojimo pavyzdys. Ant markerio yra nupiešiamas kubas.



15 pav. Dalinai uždengus lyginamąjį markerį, jo koordinatės nenustatomos.

Šiuose paveikslėliuose (14 pav., 15 pav.) demonstruojamas minimalus ArToolKit karkaso panaudojimas. Darbo autorius parašė macOS aplikaciją, kuri panaudoja ARToolKit karkasą, kad patikrintų, ar šis karkasas konceptualiai tinka norimai užduočiai atlikti. Ši minimali aplikacija aplinkoje suranda palyginamąjį markerį. Šiame pavyzdyje jis yra telefono ekrane. Aplikacija, gavusi markerio koordinates ir dydį, nupiešia trimatį kubą ant markerio koordinačių (13 pav.). Būtent markerio koordinačių radimo dalis bus panaudota drono nuleidimo ant markerio užduočiai spęsti. Uždengus dalį palyginamojo markerio, ArToolkit įrankis nesugeba jo atpažinti (14 pav.).

3.15. Karkaso apvalkalas ar_pose

Paketas ar_pose yra ARToolkit karkaso apvalkalas ROS ekosistemai. Pasinaudojant šiuo paketu galima su kamera sekti erdvėje vieną ar net kelis „ARToolkit“ markerius. Prieš naudojant šį įrankį, jį reikia sukalibruoti su kamera, kuria bus sekami markeriai. Šio mokslo tiriamojo darbo programinės įrangos sprendime yra panaudotas vieno markerio sekimas. Nustatoma markerio orientacija erdvėje ir atstumas nuo kameros. Identifikavęs markerį ir apskaičiavęs jo atstumą ir padėtį kameros atžvilgiu šis modulis publikuoja markerio pozą, bei užfiksavimo laiką į „/ar_pose_marker“ ROS temą.

Koordinatėms suskaičiuoti ir transformuoti tarp skirtingų koordinačių sistemų yra naudojamas „TF2“ paketas, kurį apžvelgsime kitame šio darbo skyriuje.

3.16. Transformacijų paketas TF

Robotai dažnai susideda iš daugelio judančių dalių. Kiekviena iš jų turi savo koordinačių sistemą. Eksperimentas gali turėti tokias koordinačių sistemas kaip: pasaulio koordinačių sistema, roboto kūno koordinačių sistema, roboto rankos koordinačių sistema ar roboto gnybtų koordinačių sistema. Šios koordinačių sistemos, laikui bėgant, kinta viena kitos atžvilgiu. Norint atlikti transformacijas tarp šių koordinačių sistemų mums pagelbsti „TF“ ROS paketas. Jis gali padėti lengviau suskaičiuoti, kokia buvo roboto galvos padėtis pasaulio koordinačių sistemos atžvilgiu, arba kokia šiuo metu yra roboto čiuptuvo padėtis roboto kūno atžvilgiu. [Foo13]

Atliekant užduotis su robotu yra labai svarbu žinoti, kur jis yra jį supančio pasaulio atžvilgiu. Robotinėms sistemoms tampant vis sudėtingesnėms, posistemių gebėjimas sekti visos sistemos būseną mažėja. Kuriant roboto komponentus reikia atsižvelgti, kokia yra minimali informacijos aibė, reikalinga užduočiai atlikti. Roboto komponentų valdymo moduliams esant viename kompiuteryje, didžiausias iššūkis yra sąsajų kūrimas, kurios pateiktų visą reikalingą informaciją kitiems moduliams. Kuriant robotų sistemas, kurių valdymo skaičiavimai yra paskirstyti keliuose kompiuteriuose, dėl informacijos srauto pralaidumo apribojimų, ne visada visa informacija apie sistemos konfigūraciją gali būti pasiekiama.

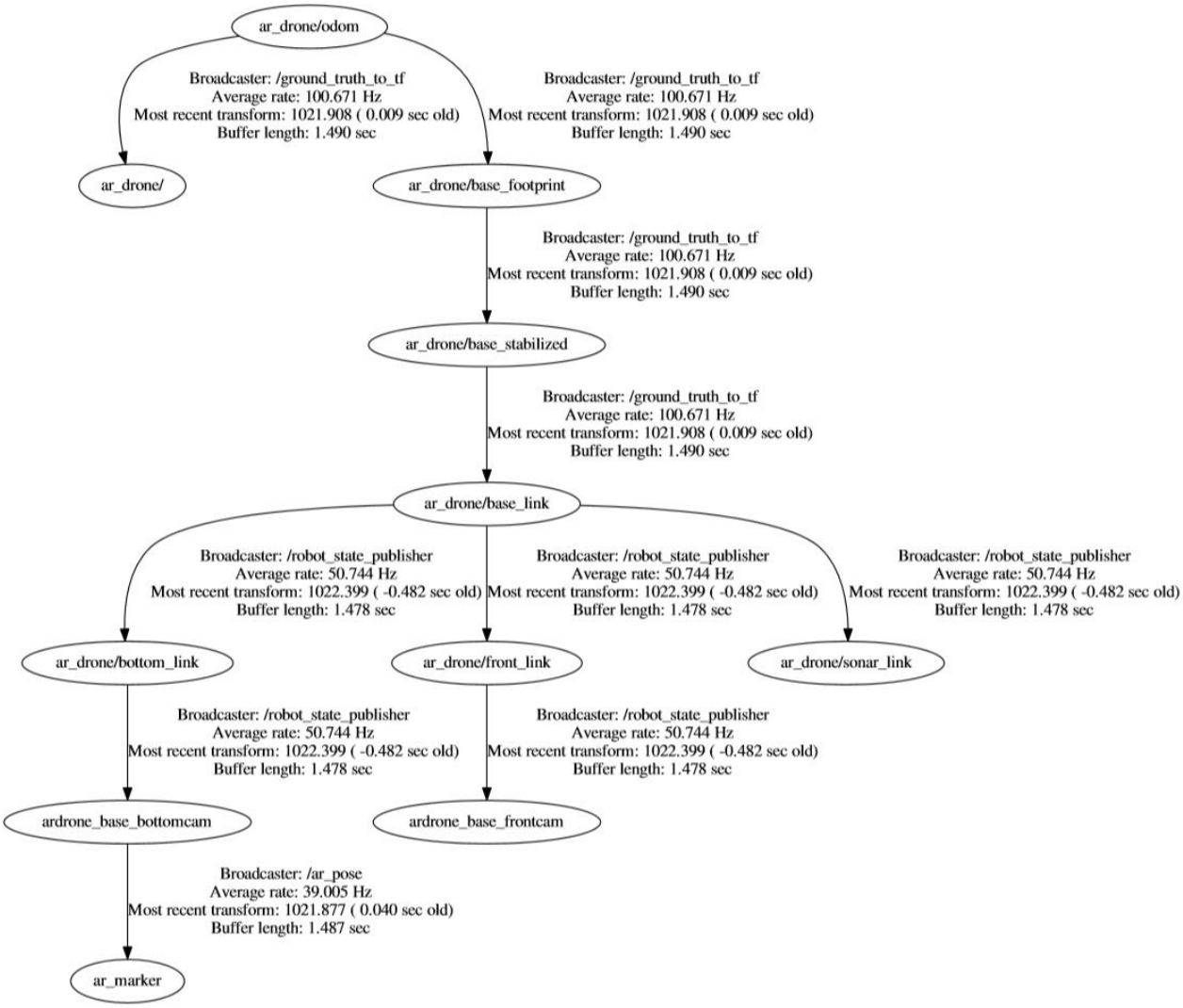
Įsivaizduokime paprastą pavyzdį – robotas, kuris su žnyplėmis turi paliesti kamuoliuką. Užduotis yra tiesiog pajudinti žnyples link kamuoliuko. Norint atlikti šią užduotį mes turime žinoti sąryšį tarp kamuoliuko ir roboto žnyplių. Jei eksperimento vietoje yra sensorius, kuris gali aptikti kamuoliuką erdvėje, tai mums reikia apskaičiuoti transformaciją tarp sensoriaus ir kambario, kambario ir roboto pagrindo, tarp roboto pagrindo ir roboto kūno, tarp roboto kūno ir roboto peties, tarp roboto peties ir roboto alkūnės, tarp roboto alkūnės ir roboto riešo, galiausiai tarp roboto riešo ir roboto žnyplių. Apskaičiuotas koordinates palyginti su kamuoliuko padėtimi erdvėje, nes mes

galime palyginti tik objektų koordinates, esančias toje pačioje koordinačių sistemoje. Tarkime, šio pavyzdžio transformacijų sprendinys – pajudinti roboto žnyples 5 centimetrus į dešinę sensoriaus koordinačių sistemoje. Norint apskaičiuoti, kokį judesį robotas turi padaryti, mes turime 5 centimetrų į dešinę judesį sensoriaus koordinačių sistemoje transformuoti į roboto kūno koordinačių sistemą. Tai padaryti galime suskaičiavę transformacijas tarp sensoriaus ir kambario, kambario ir roboto pagrindo, roboto pagrindo ir roboto kūno koordinačių sistemų. Suskaičiavus šias transformacijas gali paaiškėti, kad 5 centimetrų į dešinę judesys sensoriaus koordinačių sistemoje yra 5 centimetrų žemyn judesys roboto kūno koordinačių sistemoje. Taigi robotas turi pasilenkti 5 centimetrus žemyn.

Išnagrinėjus pateiktą pavyzdį, akivaizdu, kad programinės įrangos kūrėjui būtų žymiai paprasčiau užklausti bibliotekos, koks yra judesio vektorius, reikalingas roboto žnyplėms pajudinti roboto kūno atžvilgiu, kad žnyplės paliestų kamuoliuką. Tai yra esminis šio pavyzdžio problemos klausimas. Programuotojui neturėtų rūpėti tarpinių jungčių konfigūracijos ir kameros padėtis.

Šioms problemoms spręsti buvo sukurta „TF“ biblioteka. Ji suteikia standartinį būdą sekti koordinačių sistemas ir transformuoti duomenis tarp skirtingų roboto sistemos elementų. Šis modulis leidžia programuotojams, nežinant visų roboto sistemą sudarančių dalių koordinačių sistemų, gauti tikslius duomenis juos dominančioje koordinačių sistemoje. [Foo13]

Bibliotekoje naudojamos struktūros labai primena scenos grafą. Scenos grafas, tai duomenų struktūra, naudojama trimačių scenų vaizdams generuoti. Scenos grafas susideda iš objektų, kuriuos reikia generuoti trimatės scenos vaizdui, medžio. Kiekvienas iš objektų yra prijungtas prie tėvinio objekto (16 pav.) ir turi tam tikrą pozą jo atžvilgiu. [Foo13]



16 pav. TF scenos grafas.

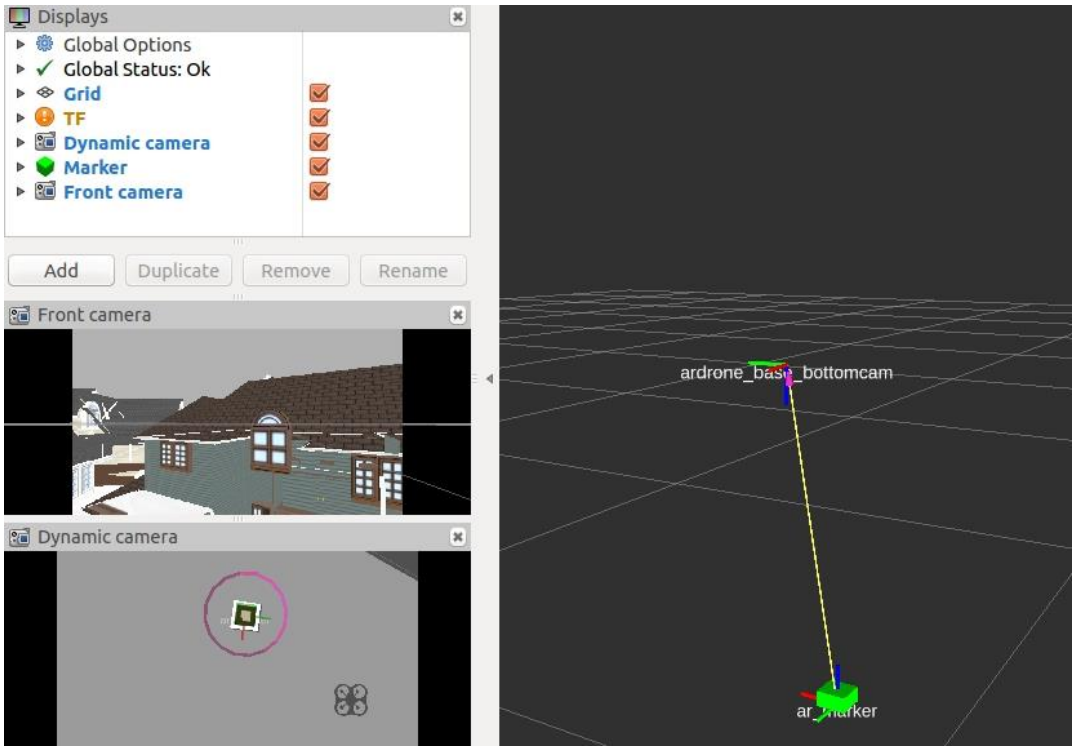
Kuriant „TF“ biblioteką, buvo atsižvelgta į tai, kad ji gali būti naudojama paskirstytose sistemose su nepatikimu ryšiu ir nemažu užlaikymu ir tai, kad sistema tarpusavyje turi bendrauti asinchroninių žinučių publikavimo/prenumeravimo mechanizmu. Biblioteka susideda iš dviejų pagrindinių dalių: transliuotojo (Broadcaster) ir klausytojo (Listener). Visi duomenys, kuriais operuoja „TF“ biblioteka, privalo turėti koordinačių sistemą, kurioje šie duomenys yra, ir laiką, kada jie yra teisingi. Transliuotojo modulis tam tikru dažniu transliuoja žinutes į ROS sistemą apie joje esančių koordinačių sistemų sąveiką, net jei sistema tuo metu nekinta. Klausytojas surašo transliuotojo atsiųstas žinutes į surikiuotą sąrašą. Klausytojas, gavęs užklausą dėl transformacijos, interpoliuoja surikiuotame sąraše esančius duomenis. [Foo13]

Norint suskaičiuoti transformacijas tarp dviejų medžio elementų, yra suskaičiuojamas persidengiantis rinkinys, o iš jo - pilna transformacija. Jei nepavyksta rasti transformacijų rinkinio tarp šių taškų, yra gražinama klaida. Skaičiuojant transformacijas tarp koordinačių sistemų a ir c, koordinačių sistemai b, esant tarp jų, transformacijų grandinėlė atrodo taip:

$$T_a^c = T_a^b * T_b^c$$

3.17. Duomenų atvaizdavimo įrankis Rviz

Rviz yra „ROS visualization“ trumpinys. Rviz tai trimačio vizualizavimo įrankis, skirtas sensorių duomenims ir roboto būsenos informacijai atvaizduoti (17 pav.). Jis leidžia atvaizduoti trimatį roboto modelį ir pavaizduoti roboto sensorių parodymus. Įrankis „rviz“ dažnai naudojamas atvaizduoti roboto mono arba stereo kamerų vaizdams, lazerinių ir ultragarsinių atstumo daviklių parodymams. Prie šio įrankio galima pajungti tikrą robotą, veikiantį realiu laiku tinkle ir atvaizduoti jo jungčių būsenas ir daviklių parodymus.



17 pav. Rviz duomenų atvaizdavimo įrankis.

Duomenų atvaizdavimas veikia ne tik su realiu laiku vykstančiomis simuliacijomis ar tikrais robotais, bet ir atkuriant duomenis iš anksčiau aprašytų „bag“ tipo failų, kuriuose gali būti išsaugoma visa tuo metu ROS sistemoje keliaujanti informacija. Vartotojas, matydamas

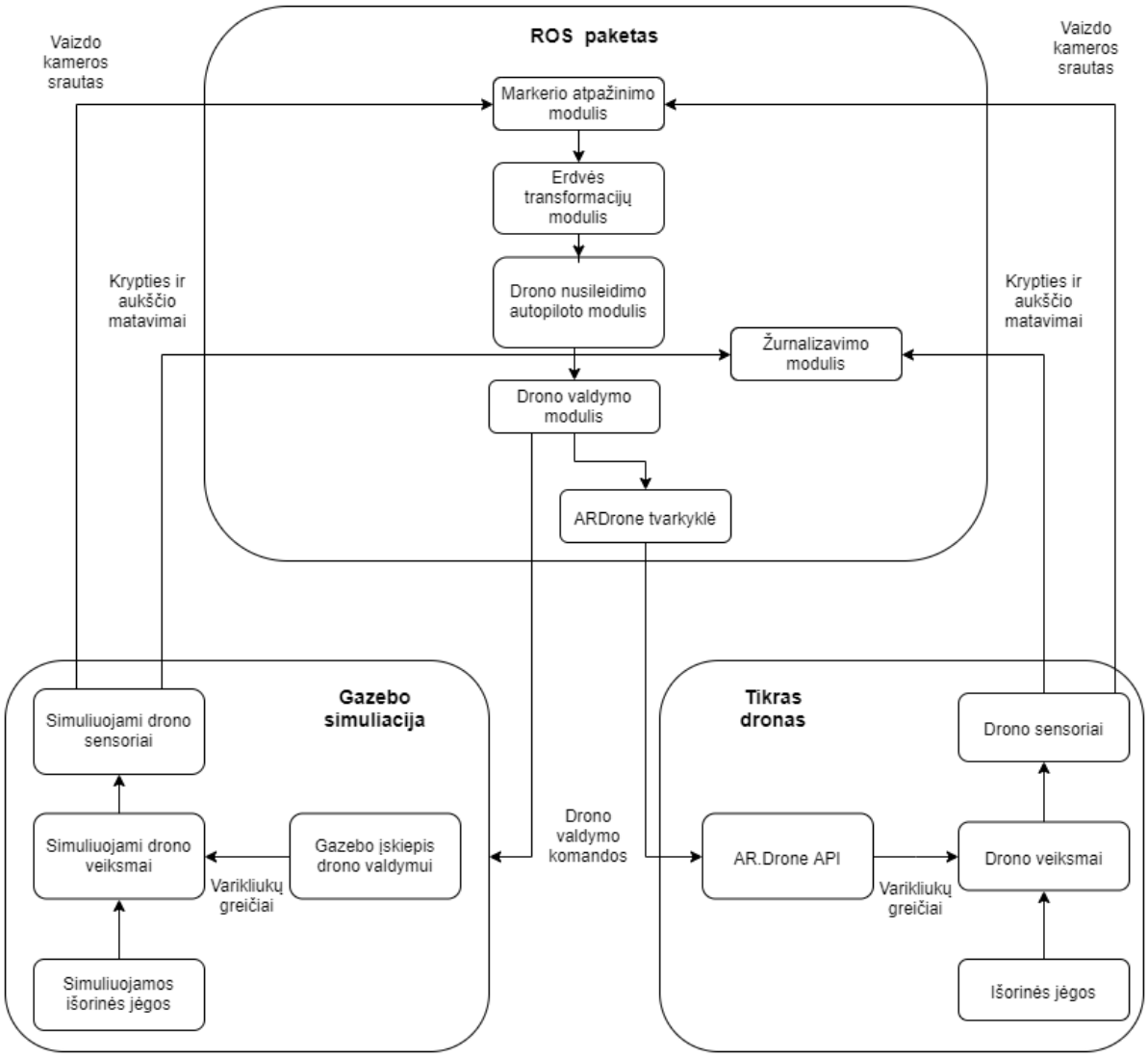
vizualizacijas, ką robotas daro ir ką mato jo sensoriai, gali lengviau taisyti klaidas ir derinti roboto valdymo kodą. Šis įrankis turi konfigūruojamą grafinę vartotojo sąsają, kurią vartotojai gali pritaikyti savo reikmėms ir susidėti sensorių atvaizdavimo įrankių aibę pagal savo norą.

4. Programinės įrangos sprendimo struktūra

Atliekant mokslo tiriamąjį darbą buvo sukurtas atviro kodo programinės įrangos sprendimas. Panaudojant šį programinės įrangos sprendimą buvo atlikti tyrimai, kurie bus apžvelgti kitose šio darbo skyriuose. Drono nuleidimo ant markerio užduoties sprendimas susideda iš trijų dalių (18 pav.).

Pirmoji dalis, tai Gazebo simuliacinis paketas ir jam sukurtas pasaulis, skirtas drono nusileidimo situacijoms modeliuoti. Ši simuliacinė aplinka yra integruota su ROS programinio paketo moduliu, kuris yra atsakingas už drono nuleidimą ant markerio. Gazebo simuliacija leidžia atlikti daugelį drono nusileidimo scenarijų, nerizikuojant tikro drono technine įranga. Simuliacijoje, kaip ir tikrame drone, yra simuliuojamos drono kameros, kurių vaizdą gauna drono valdymo modulis. Simuliacijoje esantis dronas taip pat turi daug sensorių, tokių kaip sonaras, oro slėgio sensorius, magnetometras. Šių sensorių duomenis iš simuliacijos gali panaudoti drono valdymo modulis.

Antroji dalis, tai tikras dronas ir jo valdymo tvarkyklė. Šis programinės įrangos sprendimas leidžia naudoti tiek simuliaciją, tiek tikrą droną užduočiai spręsti. Dronas, pradėjęs veikti, sukuria bevielio tinklo prieigos tašką. Prisijungus prie šio tinkle, per IP protokolą, galima pasiekti drono tvarkyklės valdymo komandas. Šio programinės įrangos sprendimo išeities failuose yra dvi pagrindinės sistemos paleidimo konfigūracijos. Pirmoji konfigūracija paleidžia Gazebo simuliaciją ir joje užkrauna simuliuojamą kelių namų kiemą su nusileidimo markeriu ir dronu. Antroji konfigūracija, vietoje simuliacijos, paleidžia drono tvarkyklę ir bando prisijungti prie tikro AR.Drone drono valdymo sistemų. Su paleistais programiniais įrankiais mes galime iš kompiuterio valdyti droną klaviatūra, žaidimų pulteliu arba siųsdami tekstines komandas ar jų eilę. Taip pat galime liepti dronui pakilti, nusileisti ir persijungti tarp priekinės ir apatinės kameros vaizdų transliavimo.



18 pav. Programinės įrangos sprendimo struktūra.

Trečioji dalis, tai ROS karkaso paketas, skirtas drono nuleidimo ant markerio valdymui. Iš tikro arba simuliuojamo drono gautas vaizdo srautas yra analizuojamas su „ar_pose“ paketu, skirtu atpažinti „ARToolkit“ tipo markerius. Suradus nusileidimo markerį su „TF“ transformacijų paketu, bandoma apskaičiuoti nusileidimo markerio koordinates drono atžvilgiu. Visą procesą, nuo markerio radimo iki nusileidimo, valdo „ard_autopilot“ ROS paketas. Šis atviro kodo paketas buvo parašytas šio mokslo tiriamojo darbo užduočiai atlikti. Jis savyje integruoja kitų ROS paketų funkcionalumus, atlieka skaičiavimus, reikalingus drono valdymo užduočiai, užtikrina loginę drono valdymo algoritmų seką, žurnalizuoja reikiamus drono nusileidimo proceso duomenis vėlesnei jų analizei. Detalesnė nusileidimo valdymo algoritmo apžvalga yra pateikta kitame šio darbo skyriuje. Priklausomai nuo paleidimo konfigūracijos, ROS modulis bendrauja arba su Gazebo simuliacine aplinka, arba su tikro AR.Drone drono tvarkykle. Detalesnę programinės įrangos sprendimo struktūrą galimą rasti priede Nr. 8.

4.1. Drono valdymo algoritmo aprašymas

Šioje dalyje apžvelgsime drono nuleidimo ant markerio algoritmą. Čia yra pateikiamas stipriai supaprastintas algoritmo pseudokodas su ji paaiškinančiu tekstu. Dar labiau supaprastintą algoritmo pseudokodą galima rasti priede Nr. 3.

Inicializuojame ROS mazgą:

```
ros::init(argc, argv, "ard_autopilot");
```

Inicijuojame TF transformacijų prenumeratorių:

```
tf2_ros::Buffer transformBuffer;
tf2_ros::TransformListener transformListener(transformBuffer);
```

Inicijuojame drono valdymo komandų publikavimo objektą:

```
ros::Publisher commandPublisher = nodeHandle.advertise<std_msgs::String>("/uga_t
um_ardrone/com", 100);
```

Sukuriame markerio radimo su ar_pose temos prenumeratorių:

```
ros::Subscriber markerSubscriber = nodeHandle.subscribe("/ar_pose_marker", 1, Ma
rkerCallback);
```

Sukuriame temos, skirtos nusileidimo trukmei skaičiuoti, publikatorių:

```
ros::Publisher timelinePublisher = nodeHandle.advertise<std_msgs::Int32>("/landi
ng_timeline", 100);
```

Apsibrėžiame drono valdymo komandas, kurias naudosime. Nusileidimo komandos pavyzdys atrodytų taip:

```
std_msgs::String land;
land.data = "c land";
```

Nuskaitome x,y,z ašių ir pokrypio paklaidas leidžiantis ant markerio. Z ašies paklaidos nuskaitymas iš ROS parametrų serverio atrodytų taip:

```
nodeHandle.getParam("/ard_autopilot/target_Z", target_z);
```

Apsibrėžiame metodą, kuris bus kviečiamas, kai mes aptinkame markerį:

```
void MarkerCallback(const ar_pose::ARMarker::ConstPtr arPoseMessage)
{
```

Išsaugome laiką, kada paskutinį kartą sėkmingai radome markerį:

```
    lastArPoseMessage = arPoseMessage->header.stamp.sec;

    if (markeris_pamatytas_pirma_karta_sesijoje) {
```

Išsisaugome laiką, kada suradome pirmąjį markerį:

```
    firstMarkerTime = lastArPoseMessage;
    }
}
```

Sukuriame pagrindinį ROS mazgo ciklą:

```
while (ros_mazgo_gyvavimo_salygos_patenkinamos)
{
    try
    {
```

Bandome gauti tranformaciją tarp drono apatinės kameros ir “ar” markerio:

```
        transformStamped = transformBuffer.lookupTransform(
            "ar_marker",
            "ardrone_base_bottomcam",
            ros::Time(0)
        );
    }
    catch
    {
```

Nepavykus gauti tranformacijos palaukiame vieną sekundę:

```
        ros::Duration(1.0).sleep();
```

Tęsiame programos darbą sekančiame while cikle:

```
        continue;
    }

    if (pirma_karta_pamatytas_markeris_nera_issaugotas && radome_markeri) {
```

Į „/landing_timeline“ temą publikuojame pirmojo markerio aptikimo laiką

```
        timelinePublisher.publish(firstMarkerTime);
    }
```

Iš gautos TF trandformacijos tarp drono ir markerio sukuriame kvaternioną:

```
    tf::Quaternion qaternion(
      transformStamped.transform.rotation.x,
      transformStamped.transform.rotation.y,
      transformStamped.transform.rotation.z,
      transformStamped.transform.rotation.w
    );
```

Iš kvaterniono sukuriame tranformacijos matricą:

```
    tf::Matrix3x3 transformationMatrix(qaternion);
```

Iš tranformacijos matricos gauname posvyrį, polinkį ir pokrypį:

```
    transformationMatrix.getRPY(roll, pitch, yaw);
```

Nustatome drono judesio objeko reikšmes pagal x, y, z ašis ir pokrypį:

```
    droneMovement.SetRoll(-transformStamped.transform.translation.x);
    droneMovement.SetPitch(-transformStamped.transform.translation.y);
    droneMovement.SetThrust(-abs(transformStamped.transform.translation.z));
    droneMovement.SetYaw(((float)yaw * 180 / PI));
```

Atimame nusileidmo taško paklaidos dydžius:

```
    droneMovement.ReduceOffsetToZero(droneMovement, x_paklaida, y_paklaida, z_pa
klaida, pokrypio_paklaida);

    if (drono atstumas nuo markerio atėmus paklaidas lygus 0)
    {
```

Nuleidžiam droną ant markerio:

```
        commandPublisher.publish<>(land);
```

Publikuojame į „/landing_timeline“ temą drono nuleidimo ant markerio laiką:

```
        timelinePublisher.publish(timeline_msg)
```

Išjungiame drono autopiloto mazgą:

```
        ros::shutdown();
    }
    else
    {
        if (markerio pamatymo laikas == dabartinis laikas)
        {
```

Judame link markerio:

```
            commandPublisher.publish<>(droneMovementCommand);
        }
        else
        {
```

Pranešame konsolėje apie pamestą markerį:

```
            ROS_WARN("Ladning marker is lost");
```

Išvalome drono komandų eilę:

```
            commandPublisher.publish<>(clear);
```

Sukuriame komandą kilti 60 cm aukštyn:

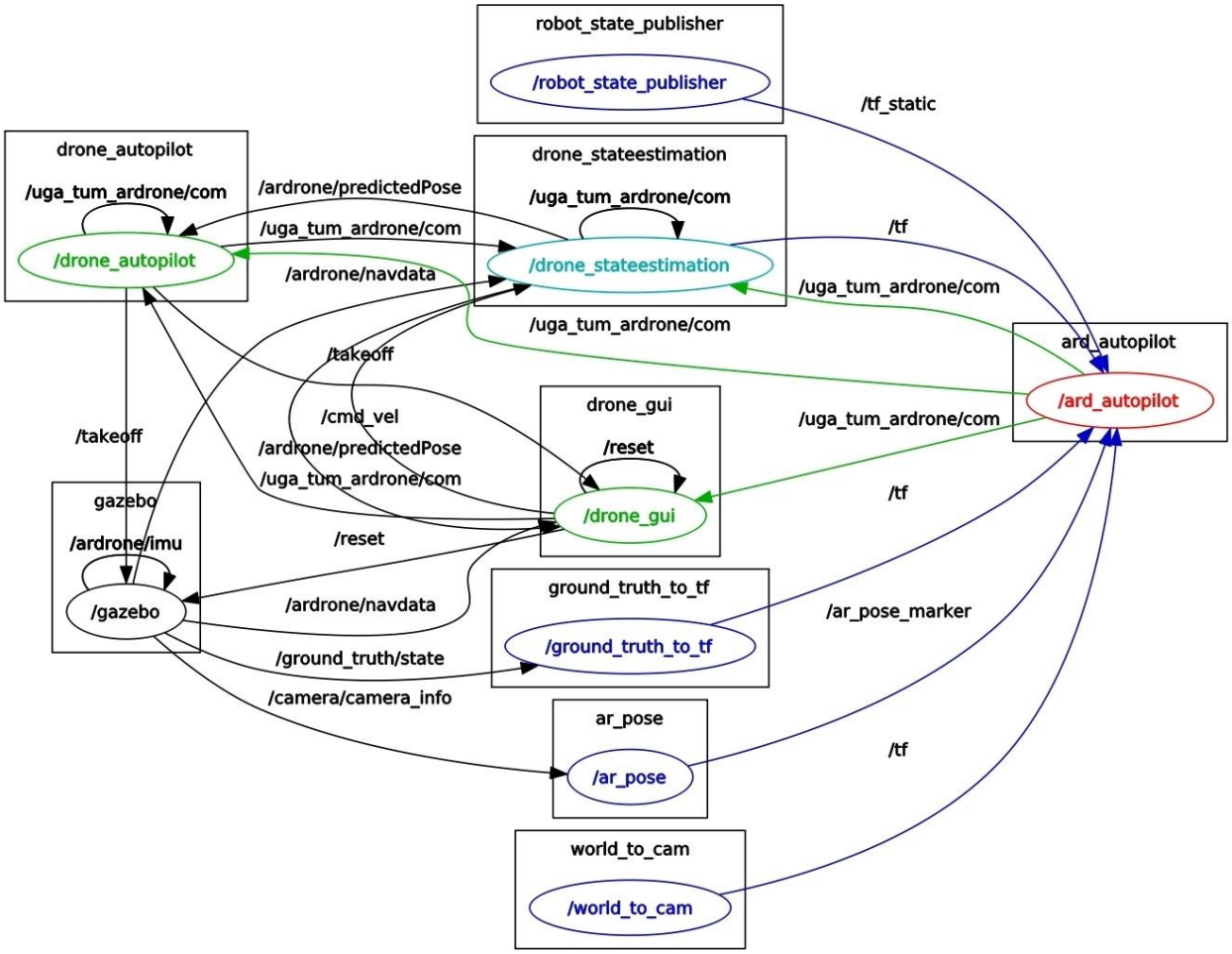
```
            droneMovementCommand.data = "c moveByRel 0 0 0.6 0";
```

Siunčiame dronui komandą kilti aukštyn:

```
            commandPublisher.publish<>(move_by_rel);
        }
    }
}
```

4.2. Sprendimo ROS mazgų grafo struktūra

Kaip jau minėta kituose skyriuose, ROS sistemoje ROS mazgai tarpusavyje dažniausiai bendrauja per ROS temas. ROS mazgai į ROS temas publikuoja žinutes jas išsiųsdami, o kiti ROS mazgai, kuriuos domina tam tikros temos, jas prenumeruoja ir taip gauna į tas temas publikuotas žinutes. Šiame skyriuje apžvelgsime pagrindius šio mokslo tiriamojo darbo programinio sprendimo ROS mazgus ir jų naudojamas ROS temas (19 pav.).



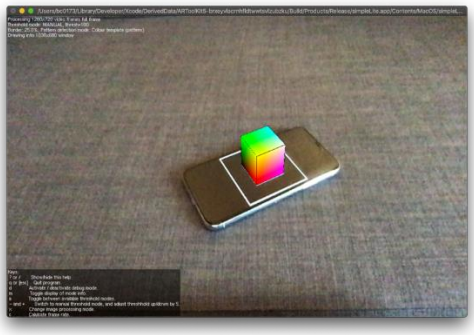
19 pav. Sistemos ROS temų grafas.

Pagrindinis šio ROS modulio mazgas yra „ard_autopilot“. Jis atsakingas už drono nuleidimo ant markerio proceso valdymą. Šis ROS mazgas prenumeruoja temą „/ar_pose_marker“, skirtą pranešti apie markerio aptikimą vaizdo sraute. ROS mazgas „ar_pose“ prenumeruoja temą „/camera/image_raw“, kurios duomenis analizuoja ieškodamas markerio. Į ROS temą „/camera/image_raw“ publikuoja dronas Gazebo simuliacijoje arba realus dronas kompiuterio tinkle. Aptikus markerį vaizdo sraute ROS mazgas „ar_pose“ apie tai praneša ROS mazgui „ard_autopilot“. Drono nuleidimo modulis „ard_autopilot“ savyje turi „TF“ bibliotekos prenumeratorių, kuris prenumeruoja ROS temą „/tf“. Į šia temą yra publikuojamos transformacijos

tarp markerio ir drono kameros. Valdymo mazgas „ard_autopilot“ iš temos „/ar_pose_marker“ gavęs žinutę, kad markeris yra aptiktas, kreipiasi į „TF“ prenumeratorių, kuris saugo visas „/tf“ temoje publikuotas žinutes. Transformacijų prenumeratorius patikrina, ar jo turimame surikiuotame sąraše yra šiam laiko momentui galiojančių transformacijų. Jei galiojanti tranformacija yra randama, ji yra grąžinama „ard_autopilot“ mazgui. Pastarasis mazgas, apdorojęs tranformaciją ir suskaičiavęs judėjimo vektorių, sukuria drono valdymo komandą ir ją publikuoja į temą „uga_tum_ardrone/com“. Kitas mazgas „drone_autopilot“, atsakingas už drono varikliukų valdymą, šią komandą paverčia atitinkamomis varikliukų greičių reikšmėmis ir perduoda jas Gazebo simuliacijai arba tikro drono tvarkyklei.

4.3. Sprendimo iteracijų aprašymas ir įžvalgos

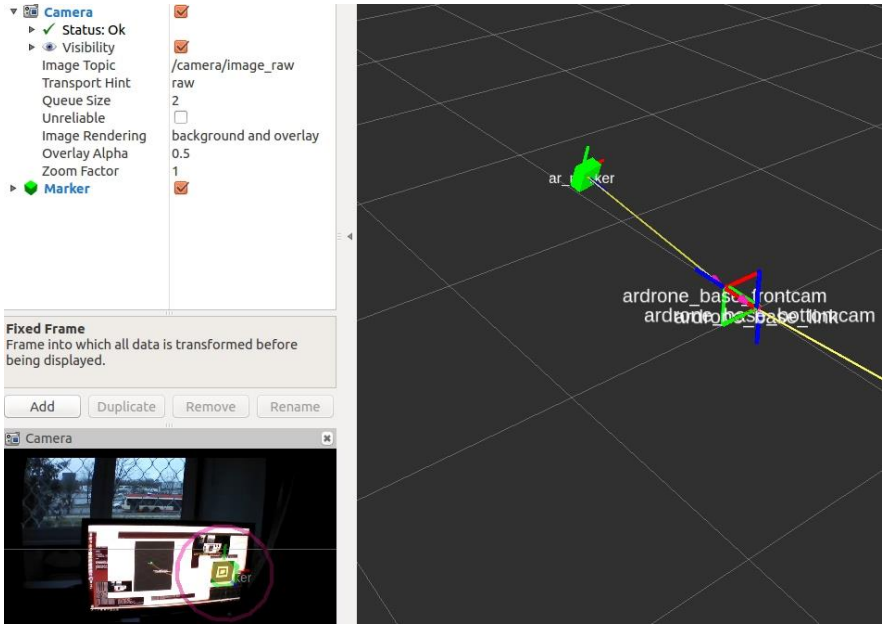
Pirmajai programinės įrangos sprendimo iteracijai buvo parašyta MacOS aplikacija, panaudojanti ARToolkit karkasą, sugebanti identifikuoti markerį iš kameros vaizdo srauto. Ši aplikacija, kuri panaudoja ARToolKit karkasą, buvo sukurta tam, kad patikrintų, ar šis karkasas konceptualiai tinka norimai užduočiai atlikti. Minimali aplikacija aplinkoje suranda palyginamąjį markerį. Šiame pavyzdyje, jis yra telefono ekrane. Aplikacija, gavusi markerio koordinates ir dydį, nupiešia trimatį kubą ant markerio koordinačių (20 pav.). Būtent markerio koordinačių radimo dalis bus panaudota drono nuleidimo užduočiai spręsti.



20 pav. ARToolkit programa.

Antroji užduoties sprendimo iteracija jau naudojo ROS karkasą ir „ar_pose“ paketą. Buvo sukurtas ROS mazgas, kuris iš kameros, prijungtos prie kompiuterio, vaizdo srauto, panaudodamas ARToolkit karkaso apvalkalą „ar_pose“, sugebėjo nustatyti markerio padėtį kameros atžvilgiu. Norint naudoti kamerą su „ar_pose“ paketu, būtina ją sukalibruoti per visą matymo lauką, vedžiojant specialią kalibracijos lentelę. Kameros kalibracijos programa suteikia vizualinius indikatorius ties kiekviena kameros ašimi, parodo, kiek procentaliai kamera yra sukalibruota. Kalibracijai baigti nebūtina pasiekti šimtaprocentines vertes kameros kalibravimo įrankyje. Pastebėta, kad šis žingsnis gali stipriai įtakoti markerio vietos nustatymo erdvėje tikslumą.

Rekomenduotina kameros kalibracijos ašis sukalibruoti kiek įmanoma tiksliau, nors tai ir užima daug laiko.

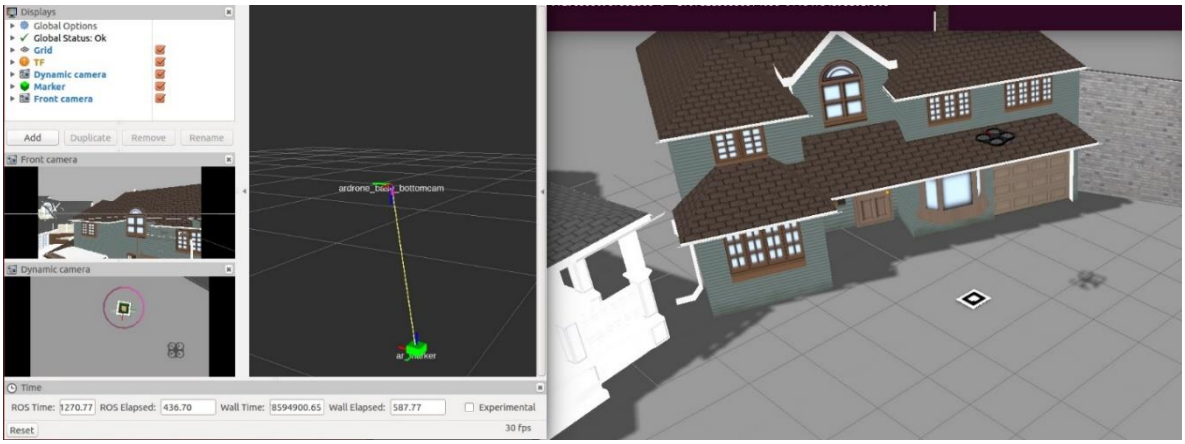


21 pav. Markerio aptikimas iš AR.Drone priekinės kameros.

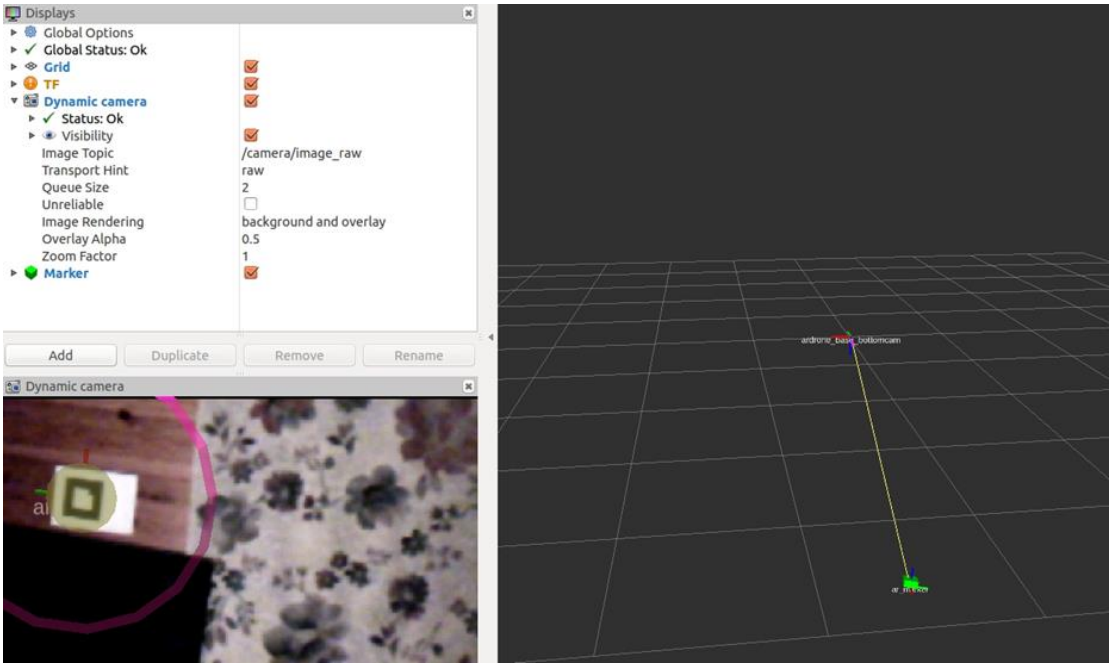
Sekantis žingsnis šioje iteracijoje, panaudoti AR.Drone drono priekinės kameros vaizdo srautą markerio aptikimui (21 pav.). Pradžioje reikėjo sukonfigūruoti ir paleisti drono tvarkyklės mazgą ir sukalibruoti markerio atpažinimo algoritmą su drono kamera. Kameros kalibracijos konfigūraciniai failai kiekvienai kamerai reikalingi skirtingi, todėl kad skirtingos kameros, dėl savo techninių specifikacijų, turi kitokias fizikines savybes ir skirtingai atvaizduoja tą patį vaizdą. Šiuo atveju drono priekinė kamera turi žymiai didesnį apžvalgos kampą nei paprasta kompiuterio kamera, todėl jai reikia naujos kalibracijos. Šioje iteracijoje pavyko sėkmingai aptikti markerį ir nustatyti jo padėtį erdvėje, kameros atžvilgiu, su paprasta kompiuterio kamera ir per tinklą pajungtu dronu.

Ketvirtojoje programinės įrangos sprendimo iteracijoje buvo gauti principiniai drono nusileidimo ant markerio užduoties sprendimai. Pradžioje buvo sukurtas virtualus pasaulis Gazebo simuliacijai, drono nuleidimui namo kieme simuliuoti. Jis susideda iš kelių namų, markerio ir drono (22 pav.). Šioje simuliacijoje buvo sukonfigūruotas virtualus dronas, kuris paklūsta drono valdymo modulio komandoms ir transliuoja vaizdo kamerų, bei drono sensorių informaciją, tokiu pačiu formatu ir į tas pačias ROS temas, kaip tikras dronas. Buvo sukurtas ROS karkaso modulis „ard_autopilot“, skirtas drono nuleidimo ant markerio procesui valdyti. Šis modulis, panaudodamas ar_pose ir TF programinės įrangos paketus, suranda ARToolkit tipo markerį, suskaičiuoja jo koordinates drono atžvilgiu ir valdo drono judėjimą, bandydamas jį nuleisti į

markerio lokaciją. Pametus markerį iš drono kameros vaizdo, šis modulis, manevruodamas, bando vėl aptikti markerį. Drono duomenims stebėti buvo sukonfigūruotas Rviz įrankis. Šiame sprendime jis atvaizduoja drono kamerų matomus vaizdus, ar_pose nuspėjamą markerio vietą ir TF paketo suskaičiuotą vektorių, tarp kameros ir markerio. Atliktas konceptualus bandymas, paleistas drono nuleidimo modulis su tikru AR.Drone dronu (23 pav.). Visiems naudojamiems ROS mazgams buvo sukonfigūruoti ROS „launch“ tipo failai, supaprastinantys keliolikos procesų paleidimą iki vienos komandinės eilutės komandos. Drono nuleidimo modulis buvo sukurtas kaip ROS paketas, tai leis kitiems ROS karkaso varotojams šį programinės įrangos sprendimą panaudoti savo eksperimentams.



22 pav. Pilnai veikianti simuliacinė aplinka.



23 pav. Markerio aptikimas su tikru AR.Drone.

5. Drono nusileidimo valdymo algoritmo tyrimas

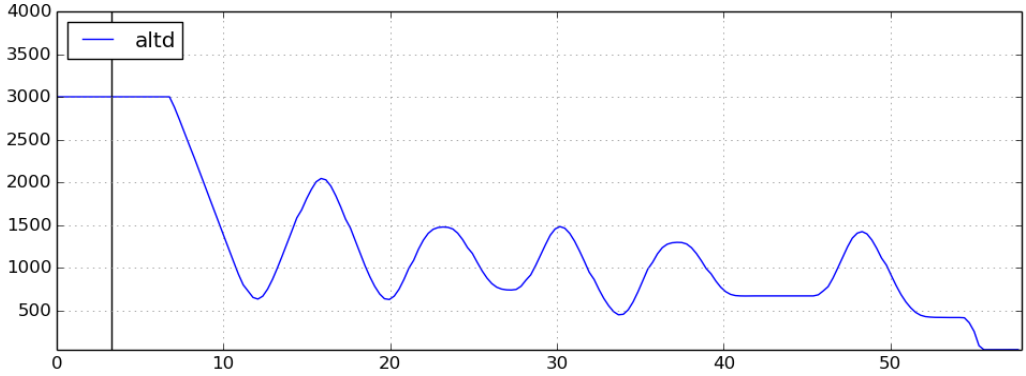
5.1. Valdymo algoritmo agresyvumo parametrų pirminis tyrimas

Drono valdymo sistemoje yra naudojamas suspaustos spyruoklės svyravimų slopinimo algoritmas. Jis skirtas stabilizuoti droną, dalinai suvaržant jo manevrus. Šio algoritmo slopinimo stiprumo valdymui yra naudojami du kientamieji. K_rp – kintamasis skirtas posvyrio (roll) ir polinkio (pitch) judesiams slopinti. K_direct – kintamasis skirtas pokrypio (yaw) ir akseleracijos Z ašimi judesiams. Toliau nagrinėjamuose tyrimuose apžvelgsime, kaip šie du valdymo parametrai įtakoja drono nusileidimo kokybę ir greitį.

Tyrimai buvo atlikti Gazebo simuliacinėje aplinkoje. Atliekant tyrimus, dronas buvo pakeliamas į trijų metrų aukštį ir paslenkamas nuo nusileidimo markerio per 1.5 metro X ašimi ir 1 metrą Y ašimi. Taip pat jis buvo pakreipiamas 225 laipsnius pokrypio ašimi, kad savo manevro pradžioje jis turėtų ne tik judėti į šonus, bet ir suktųsi apie savo ašį. Drono valdymo algoritmas, identifikavęs markerį aplinkoje, nustato dabartinį laiką ir publikuoja jį „/landing_timeline“ temoje. Visas simuliacinis procesas yra įrašomas su „rosbag“ įrankiu vėlesnei analizei. Eksperimento tyrimų grafikai buvo sugeneruoti iš „rosbag“ išsaugotų duomenų su „rqt_plot“ įrankiu.

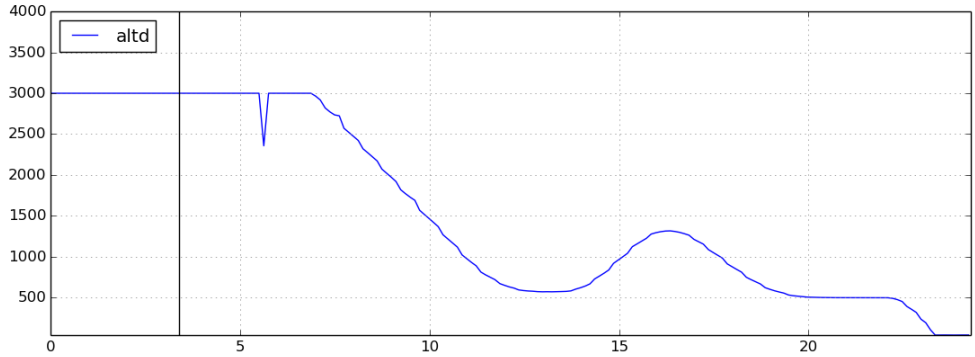
Atlikus keliolika eksperimentų su drono nusileidimu simuliacijoje ant markerio, išryškėjo kelios tendencijos. Kitaip nei manyta, mokslo tiriamojo darbo antrojoje dalyje, agresyvūs judesiai ties X ir Y ašimis nepadidino markerio pametimo dažnio. Per daug stabilizuoti judesiai ties X ir Y ašimis, nors ir stabilizavo drono judėjimą, bet prailgino jo nusileidimo laiką.

Pirma eksperimentų grupė buvo atlikta, pasirinkus mažą X ir Y ašių agresyvumą, kai K_rp yra 0.5 ir vidutinį Z ašies agresyvumą, kai K_direct yra 5. Dronas, kybantis 3 metrų aukštyje, nuo markerio aptikimo iki nusileidimo ant markerio, užtruko 48 sekundes (24 pav.). Pernelyg didelis drono judesių varžymas trukdė jam atlikti pakankamus pokrypio ir polinkio manevrus, kad galėtų nusileisti ant numatytų koordinačių iš pirmo karto.



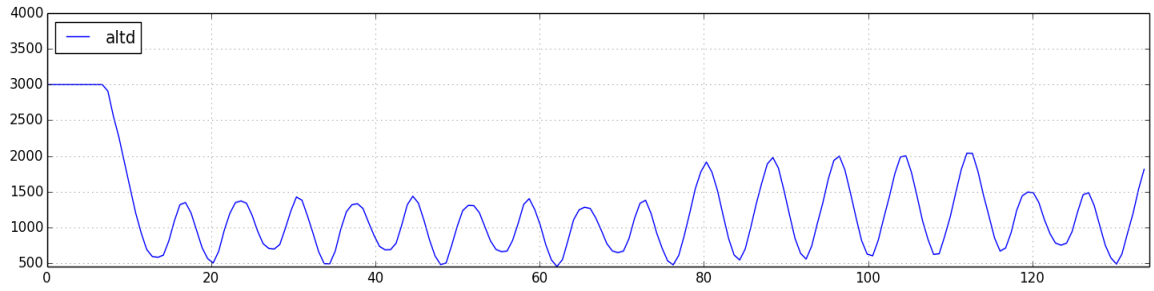
24 pav. Orlaivio nusileidimo aukščio grafikas. X ašyje – laikas sekundėmis, Y ašyje - drono aukštis milimetrais, K_rp = 0.5, K_direct = 5.

Antroji eksperimentų grupė buvo atlikta, pasirinkus labai didelį X ir Y ašių agresyvumą, kai K_rp yra 25 ir labai didelį Z ašies agresyvumą, kai K_direct yra 35. Dronas, kybantis 3 metrų aukštyje, nuo markerio aptikimo iki nusileidimo ant markerio, užtruko 16 sekundžių (25 pav.). Dronas, dėl pernelyg agresyvių manevrų, trumpam pametė nusileidimo markerį ir nusileido sėkmingai tik iš antro karto, po savo trajektorijos korekcijos.



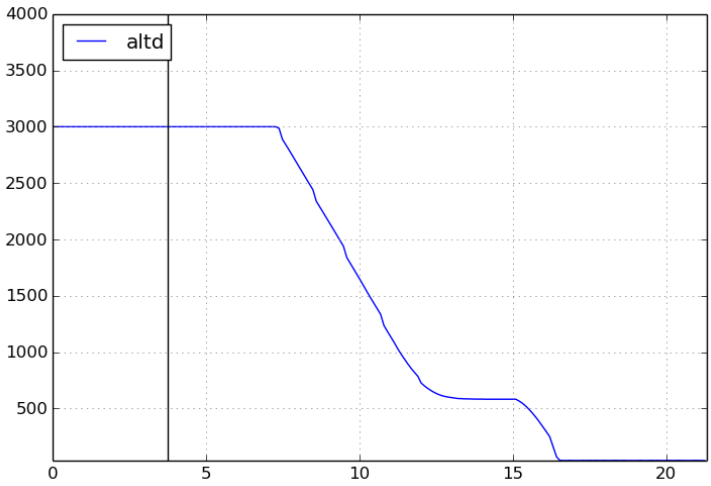
25 pav. Orlaivio nusileidimo aukščio grafikas. X ašyje – laikas sekundėmis, Y ašyje - drono aukštis milimetrais, K_rp = 25, K_direct = 35.

Trečioji eksperimentų grupė buvo atlikta, pasirinkus labai mažą X ir Y ašių agresyvumą, kai K_rp yra 0.01 ir labai didelį Z ašies agresyvumą, kai K_direct yra 35. Dronas, kybantis 3 metrų aukštyje, po markerio aptikimo, nesugebėjo nusileisti net per 125 sekundes (26 pav.).



26 pav. Orlaivio nusileidimo aukščio grafikas. X ašyje – laikas sekundėmis, Y ašyje - drono aukštis milimetrais, K_rp = 0.01, K_direct = 35.

Ketvirtoji eksperimentų grupė buvo atlikta, pasirinkus didelį X ir Y ašių agresyvumą, kai K_rp yra 10 ir vidutinį Z ašies agresyvumą, kai K_direct yra 5. Dronas, kybantis 3 metrų aukštyje, nuo markerio aptikimo iki nusileidimo ant markerio, užtruko 8 sekundes (27 pav.). Tai buvo optimaliausia drono nusileidimo konfigūracija, pademonstravusi trumpiausią laiką.



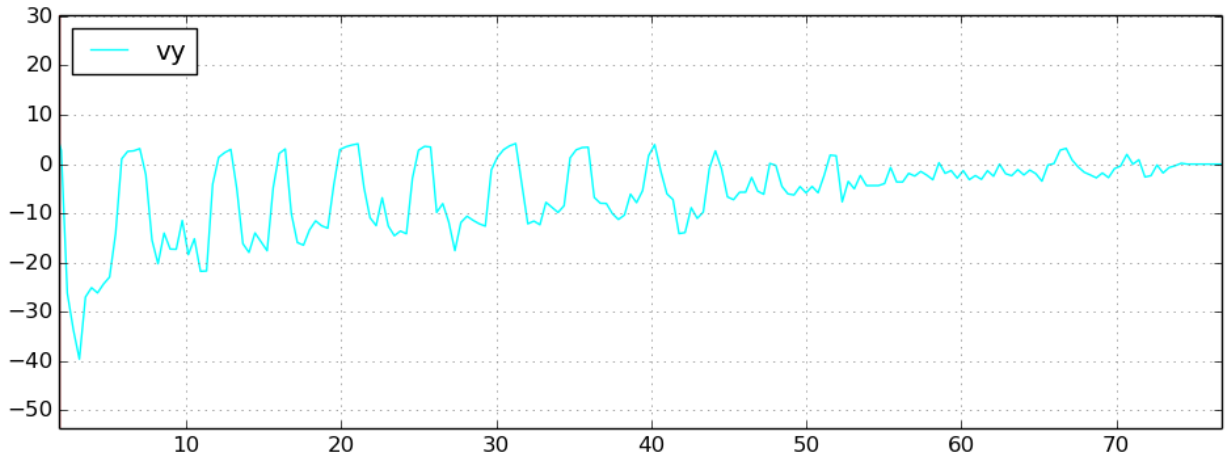
27 pav. Orlaivio nusileidimo aukščio grafikas. X ašyje – laikas sekundėmis, Y ašyje - drono aukštis milimetrais, K_rp = 10, K_direct = 5.

Taigi, tyrimo eigoje paneigtos išankstinės prielaidos, kad dronas besileisdamas neturėtų daryti labai staigių ir didelės amplitudės judesių X ir Y ašimis. Pastebėta, kad pernelyg didelis drono judesių amplitudės ribojimas šiomis ašimis pablogina nusileidimo ant markerio laiką. Prie labai mažų X ir Y ašių agresyvumo verčių, netgi užblokuoja galimybę dronui sėkmingai įvykdyti nusileidimo užduotį.

5.2. Skirtumai tarp agresyvumo konfigūracijų

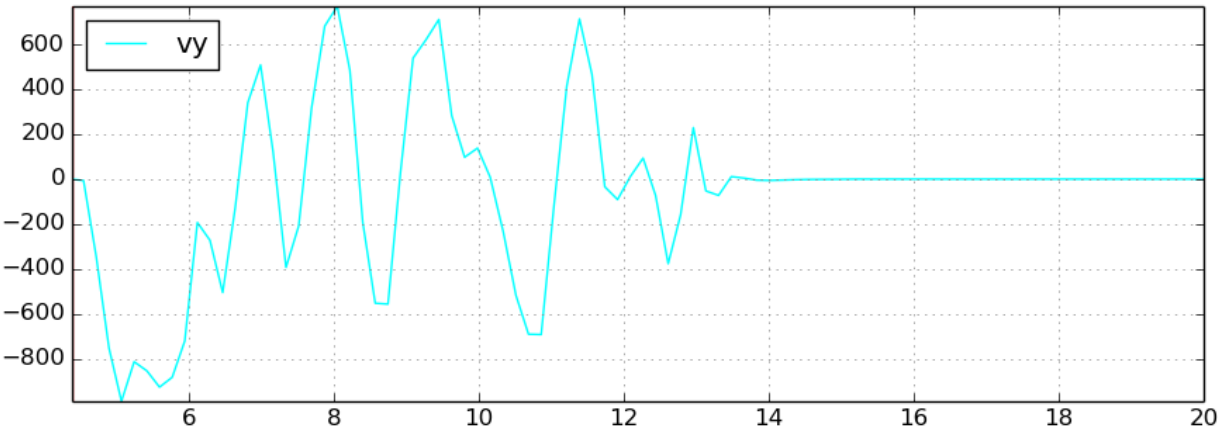
Bepiločio orlaivio tyrimams atlikti bus naudojamos kelios skirtingos drono judesių agresyvumo konfigūracijos. Kad būtų lengviau suprasti, kaip skiriasi mažai agresyvūs drono manevrai ir agresyvūs drono manevrai, palyginkime dviejų skirtingų agresyvumo konfigūracijų grafikus. Šiuose grafikuose pavaizduoti bepiločio orlaivio judesių greičiai Y ašimi, atliekant nusileidimą ant palyginamojo žymeklio, indikuojančio nusileidimo tašką.

Naudojant “suvaržytą” drono judesių agresyvumo konfigūraciją (28 pav.) dronas Y ašimi juda nuo 0 mm/s iki 40 mm/s. Kadangi beveik visos reikšmės tik viename (neigiamame) reikšmių diapazone, tai reiškia, kad dronas nekerta X ašies atlikdamas pernelyg stiprius manevrus. Jis artėja link X ašies ir, ją pasiekęs, stabilizuoja judesius Y ašimi.



28 pav. Orlaivio judesių greitis Y ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę. Taip atrodo drono greitis Y ašyje (vY) prie "suvaržyto" agresyvumo ir aukšto atnaujinimo dažnio.

Atliekant nusileidimą ant palyginamojo žymeklio ir naudojant "aukštą" drono judesių agresyvumo konfigūraciją (29 pav.) dronas Y ašimi juda nuo -1000 mm/s iki 800 mm/s greičiu. Kadangi greičiai yra ir teigiamoje ir neigiamoje skaičių aibėse, tai reiškia, kad dronas kerta X ašį. Šių greičių reikšmės pakankamai didelės, tiek teigiamoje aibėje, tiek neigiamoje aibėje, o tai reiškia, kad dronas stipriai persistengia judėdamas link X ašies, ją kerta per daug dideliu kampu ir todėl turi stipriai kompensuoti savo manevrą judėdamas link jos iš kitos puses, taip pat dideliu kampu. Iš grafiko galima matyti, kad dronas, darydamas nusileidimo manevrą agresyviais judesiais, kerta X ašį daug kartų.



29 pav. Orlaivio judesių greitis Y ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę. Taip atrodo drono greitis Y ašyje (vY) prie "aukšto" agresyvumo ir aukšto atnaujinimo dažnio.

5.3. Nusileidimo algoritmo valdymo dažnio pirminis tyrimas

Patobulinus drono valdymo modulį buvo atliktas detalesnis valdymo algoritmo agresyvumo parametrų tyrimas. Buvo ištirta algoritmo efektuvumo priklausomybė nuo algoritmo darbo dažnio.

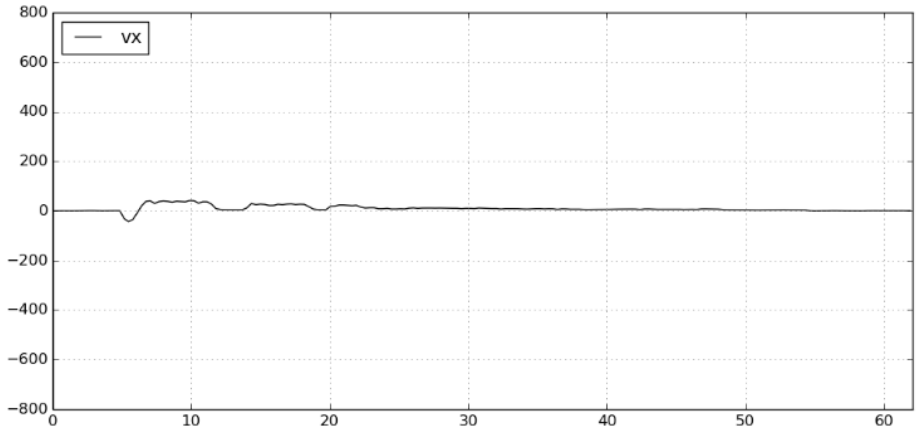
Dronas, kaip ir ankstesniuose testuose, buvo pakeliamas į 3 metrų aukštį. Šiame tyrime jis buvo toliau nuo markerio X ir Y ašyse ir buvo paslenkamas nuo nusileidimo markerio per 2 metrus X ir Y ašimis.

Atliekant pirminius tyrimus, algoritmo darbo dažnio didinimas neigiamai koreliavo su nusileidimo laiku. Atlikus daugiau drono valdymo algoritmo darbo dažnio tyrimų paaiškėjo, kad pirminiuose tyrimuose gauti rezultatai buvo lokalūs minimumai. 1 lentelėje apibrėžiamos trys drono agresyvumo konfigūracijos. „Pirminis“ agresyvumas, tai konfigūracija su nustatymais pagal nutylėjimą. „Optimalus“ agresyvumas, tai konfigūracija geriausiai pasirodžiusi per pirminius drono nuleidimo tyrimus. „Aukštas“ agresyvumas, tai konfigūracija su maksimaliomis agresyvumo parametrų reikšmėmis.

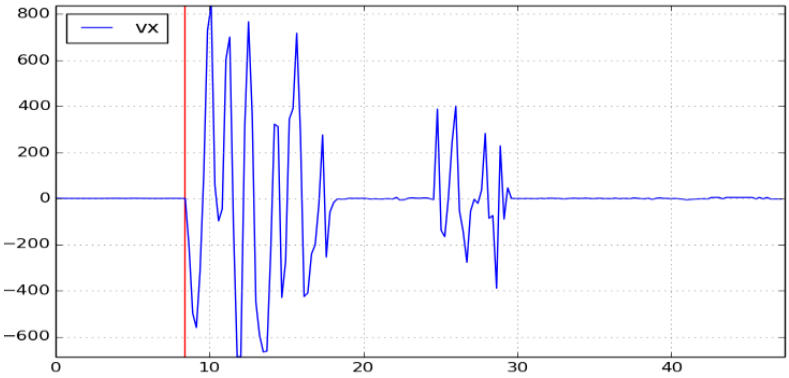
1 lentelė. Agresyvumo parametrų konfiguracijos.

Agresyvumai	*Pirminis*	*Optimalus*	*Aukštas*
K_direct	5	5	50
K_rp	0.5	10	50

Agresyvumo konfigūracijų palyginimui pateikiami du grafikai, parodantys drono judėjimo X ašimi greičius. 30 pav. drono judėjimo greičiai su Pirmine agresyvumo konfigūracija. 31 pav. drono judėjimo greičiai su Aukšto agresyvumo konfigūracija. Iš grafikų galima spręsti, kiek kartų agresyviau manevrus atlieka dronas su Aukšto agresyvumo nustatymais.



30 pav. Orlaivio judesių greitis X ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę.



31 pav. Orlaivio judesių greitis Y ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę.

Šiam tyrimui atlikti buvo pasirinktos trys dažnio konfigūracijos (2 lentelė). „Žemas“ dažnis, tai konfigūracija, kai drono valdymo algoritmas nuskaito sensorius 1Hz dažniu ir, nustatęs markerio poziciją, juda link jo 2 sekundes, nepertraukdamas manevro. Pametęs nusileidimo markerį dronas 2 sekundes kyla aukštyn, nepertraukdamas manevro. “Vidutiniu” dažniu dirbantis dronas sensorių informaciją nuskaito 10 kartų per sekundę, o judėjimo link nusileidimo taško arba kilimo aukštyn jo beieškant, manevrus atlieka bent po 1 sekundę nepertraukdamas darbo. “Didelio” dažnio konfigūracija reiškia, kad dronas nuskaito sensorius 30 kartų per sekundę, o jo judėjimo manevrai yra atnaujinami 10Hz dažniu.

2 lentelė. Greitaveikos konfigūracijų dažnių parametrai.

Dažnio pavadinimas	*Žemas dažnis*	*Vidutinis dažnis*	*Didelis dažnis*
Sensorių nuskaitymo dažnis	1 Hz	10 Hz	30 Hz
Algoritmo uždelsimas sekundėmis vykdant judėjimo link tikslo komandą	2 s	1 s	0.1 s
Algoritmo uždelsimas sekundėmis vykdant žymeklio paieškos komandą	2 s	1 s	01. s

Žemiau esančioje 3 lentelėje pateikiama drono nusileidimo laiko priklausomybė nuo jo valdymo algoritmo judesių agresyvumo ir valdymo algoritmo darbo dažnio. Detalesnį tyrimų aprašą galima rasti priede Nr. 4.

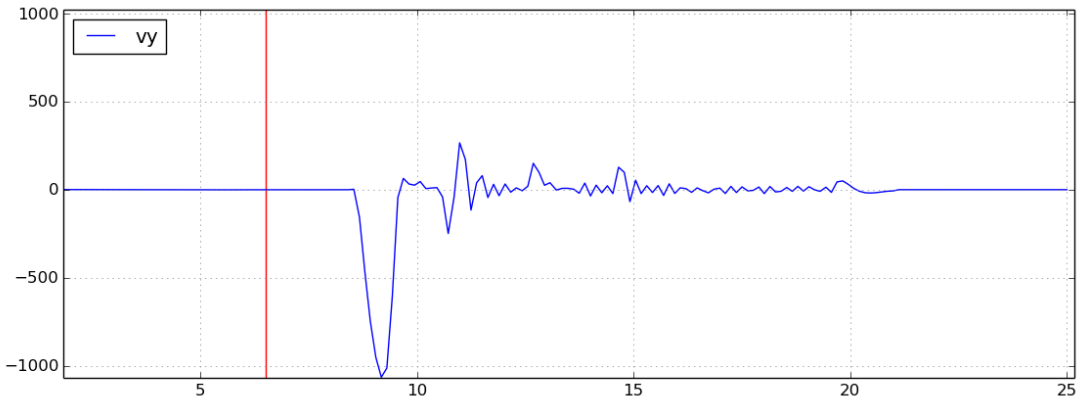
3 lentelė. Agresyvumo parametrų ir greitaveikos dažnių įtaka rezultatams.

Agresyvumas \ Dažnis	Žemas dažnis	Vidutinis dažnis	Didelis dažnis
Pirminis agresyvumas	45 s	15 s	44 s
Optimalus agresyvumas	24 s	7 s	7 s
Aukštas agresyvumas	19 s	7 s	26 s

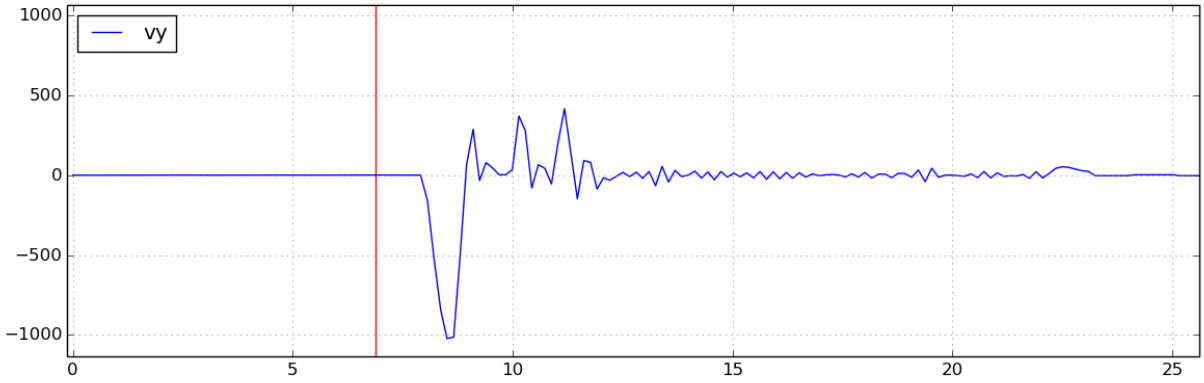
Atlikus tyrimus paaiškėjo, kad drono valdymo algoritmui veikiant „žemu“ dažniu, didinant jo valdymo konfigūracijos agresyvumą, trumpėja nusileidimo laikas. Tai reiškia, kad gerėja drono nusileidimo valdymo algoritmo veikimo efektyvumas.

Didinant valdymo algoritmo greitį – drono nusileidimo laikas taip pat trumpėjo. Dronui veikiant „vidutiniu“ dažniu, palyginti su „žemo“ dažnio algoritmo greičiu, pagerėjo drono nusileidimo laikas su visomis agresyvumo konfigūracijomis. Tiesa, perėjus prie didelio dažnio tik su viena iš agresyvumo konfigūracijų, nusileidimo laikas išliko optimalus. Su „pirmine“ ir „aukšta“ agresyvumo parametrų konfigūracijomis, nusileidimo ant žymeklio laikas išaugo, esant „dideliam“ algoritmo darbo dažniui.

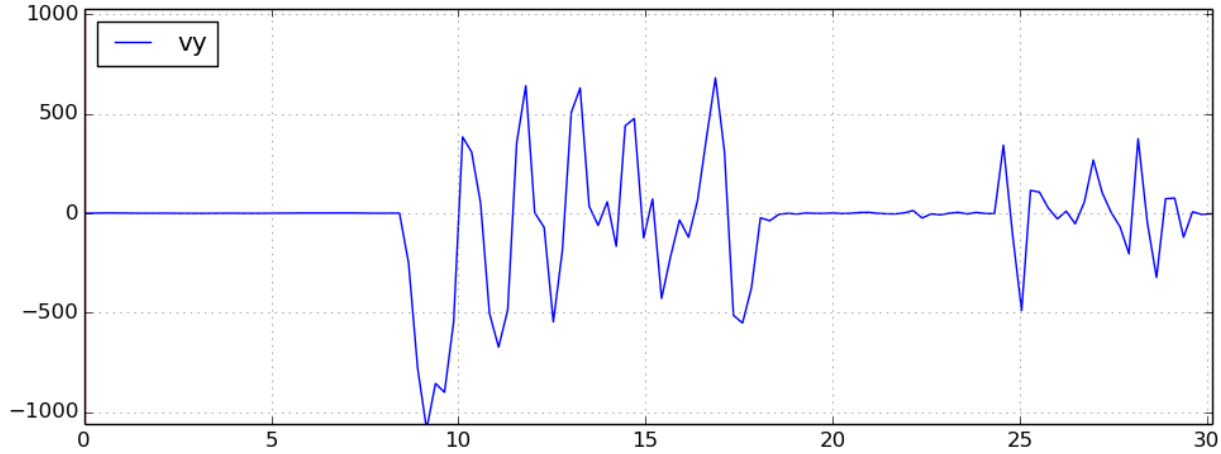
Toliau esančiuose 32, 33 ir 34 paveikslėliuose galime matyti, kaip skiriasi drono judėjimo Y ašimi greičiai, naudojant „aukšto“ agresyvumo konfigūraciją ir skirtingus algoritmo veikimo dažnius.



32 pav. Orlaivio judesių greitis Y ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę. “Aukštas” agresyvumas ir “žemas” dažnis.



33 pav. Orlaivio judesių greitis Y ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę. "Aukštas" agresyvumas ir "vidutinis" dažnis.



34 pav. Orlaivio judesių greitis Y ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę. "Aukštas" agresyvumas ir "didelis" dažnis.

Išanalizavus grafikus – matome, kad prie „didelio" algoritmo veikimo dažnio, su „aukšto" agresyvumo konfigūracija, drono judesiai kelis kartus agresyvesni nei prie lėtesnių veikimo dažnių.

5.4. Detalus nusileidimo algoritmo tyrimas

Patobulinus bepiločio orlaivio valdymo algoritmą ir pasirinkus sudėtingesnes pradines koordinates, buvo atlikti detalūs bepiločio orlaivio valdymo algoritmo priklausomybės nuo algoritmo veikos dažnio ir agresyvumo konfigūracijos tyrimai. Detalesnis tyrimų aprašymas matomas priede Nr. 6.

4 lentelė. Agresyvumo konfigūracijos.

Agresyvumas	*Suvaržytas*	*Standartinis*	*Realus*	*Optimalus*	*Z agresyvus*	*Tarpinis*	*Vidutinis*	*Aukštas*
K_direct	15	5	3	5	45	15	35	50
K_rp	0.25	0.5	5	10	15	15	35	50

Šiam tyrimui buvo sukurta daugiau skirtingų bepiločio orlaivio valdymo algoritmo agresyvumo konfigūracijų. 4 lentelėje išvardintos konfigūracijos yra parametrų rinkiniai. Šie parametrai valdo drono judesių agresyvumą, judant link nusileidimmo taško. Pagrindiniai parametrai, kurie įtakoja bepiločio orlaivio judėjimą, yra „K_direct“ ir „K_rp“. „K_direct“ kintamasis nusako, kaip stipriai yra suvaržyti judesiai pokrypio (yaw) ir akseleracijos Z ašimi. „K_rp“ kintamasis indikuoja drono posvyrio (roll) ir polinkio (pitch) judesių suvaržymus. Sekančioje pastraipoje apžvelgsime agresyvumo konfigūracijas, naudotas tyrimams atlikti. Kad būtų paprasčiau jas įvardinti ir suprasti jų paskirtį, joms suteikti pavadinimai.

5 lentelė. Agresyvumo parametrų ir greitaveikos dažnių įtaka rezultatams. Reikšmės sekundėmis.

Dažnis\ Agresyvumas	*Suvaržytas*	*Standartinis*	*Realus*	*Optimalus*	*Z agresyvus*	*Tarpinis*	*Vidutinis*	*Aukštas*
Žemas	276 s	47 s	48 s	14 s	22 s	12 s	12 s	15 s
Aukštas	71 s	40 s	6 s	6 s	6 s	5 s	5 s	9 s
Labai aukštas	86 s	38 s	6 s	5 s	5 s	5 s	5 s	12 s

„Standartinis“ agresyvumas, tai konfigūracija su numatytais parametrais, su kuriais buvo atlikti pirmieji skrydžiai. „Suvaržytas“ agresyvumas, tai drono konfigūracija, kurioje posvyrio ir polinkio agresyvumo parametras dar labiau sumažintas nei „Standartinėje“ konfigūracijoje, o pokrypio ir Z ašies agresyvumas yra didelis. Su šia konfigūracija dronas turėtų lėtai judėti X ir Y ašimis, bet pakankamai laisvai Z ašimi. „Realus“ agresyvumas, tai konfigūracija sukurta retrospektyviai po bandymų su tikru dronu. Su šia konfigūracija pavyko lauke įvykdyti nusileidimą ant markerio su tikru dronu. Tam, kad palygintumėme drono elgesį realiame pasaulyje ir simuliacijoje buvo sukurta ši agresyvumo konfigūracija simuliacijos tyrimams. „Optimalus“ agresyvumas, tai pirminiuose tyrimuose geriausiai pasirodžiusi konfigūracija. „Z agresyvus“ –

kaip galime spręsti iš pavadinimo, tai konfigūracija su dideliu agresyvumu Z ašiai ir normaliu agresyvumu X ir Y ašims. „Tarpinis“ ir „Vidutinis“ agresyvumai, tai didelio agresyvumo konfigūracijos. „Aukštas“ agresyvumas tai maksimalių agresyvumo verčių konfigūracija.

6 lentelė. Agresyvumo parametrų ir greitaveikos dažnių įtaka rezultatams. Reikšmės procentine išraiška.

Agresyvumas	*Suvaržytas*	*Standartinis*	*Realus*	*Optimalus*	*Z agresyvus*	*Tarpinis*	*Vidutinis*	*Aukštas*
Žemas -> Aukštas	-74.3%	-14.9%	-87.5%	-57.1%	-72.7%	-58.3%	-58.3%	-40%
Aukštas -> Labai aukštas	21.1%	-5%	0%	-16.7%	-16.7%	0%	0%	33.3%

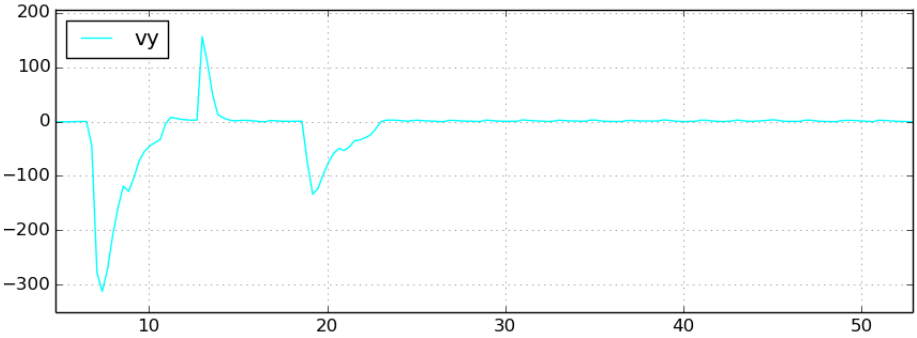
Šis tyrimas užtruko ilgiausiai. Atlikus detalius agresyvumo konfigūracijų ir algoritmo veikimo dažnių tyrimus, išryškėjo kelios tendencijos, matomos 5 ir 6 lentelėse. Naudojant “suvaržytą” agresyvumo konfigūraciją ir padidinus algoritmo greitaveikos dažnį, iš “žemo” į “aukštą”, drono nusileidimo laikas sutrumpėjo 74%. Tiesa, toliau didinant algoritmo veikimo greitį iš „aukšto“ į „labai aukštą“, nusileidimo laikas pablogėjo 21%. Su „standartiniu“ agresyvumu, padidinus algoritmo veikos dažnį iš „žemo“ į „aukštą“, nusileidimo greitis pagerėjo 15%, o toliau didinant dažnį iki „labai aukšto“, išliko panašus, atsižvelgiant į matavimų paklaidas. Nusileidimo laiko skirtumas tarp algoritmo veikimo su „aukštu“ dažniu ir „labai aukštu“ dažniu, naudojant „realų“, „optimalų“, „Z agresyvų“, „tarpinį“ ir „vidutinį“ agresyvumus, yra 1 sekundė, kaip ir paklaidos rėžis, atliekant šiuos tyrimus, todėl šiose konfigūracijose nusileidimo efektuvumas laikomas vienodu. Naudojant „realaus“ agresyvumo konfigūraciją, dažnio didinimas pagerino nusileidimo laiką beveik 88% su „aukštu “ ir „labai aukštu“ dažniais. „Optimalią“ agresyvumo konfigūraciją aukštesnis dažnis pagerino 57%. Su „Z agresyvia“ konfigūracija, didinant veikos dažnį, nusileidimo ant markerio efektuvumas padidėjo 72%. Naudojant „tarpinį“ ir „vidutinį“ agresyvumus, didinant algoritmo greitaveiką, drono nusileidimo laikas sumažėjo 58%. Naudojant „aukštą“ agresyvumą ir padidinus greitaveikos dažnį, nuo „žemo“ iki „aukšto“, bepiločio orlaivio nusileidimo laikas ant lyginamojo žymeklio sumažėjo 40%, bet toliau didinant dažnį iki „labai aukšto“ dažnio, drono nusileidimas sulėtėjo 33%.

Atliekant eksperimentus pastebėta, kad, didinant drono valdymo algoritmo greitaveikos dažnį, nusileidimo efektyvumas gerėja ir trumpėja drono nusileidimo laikas ant lyginamojo žymeklio. Tik ribinės agresyvumo konfigūracijos elgiasi kitaip. Su „suvaržyta“ ir „aukšta“ agresyvumo konfigūracijomis, didinant algoritmo greitaveikos dažnį nuo „žemo“ iki „aukšto“, pastebimas bepiločio orlaivio nusileidimo efektyvumo padidėjimas, bet toliau didinant greitaveikos dažnį iki „labai aukšto“, efektyvumas vėl krenta.

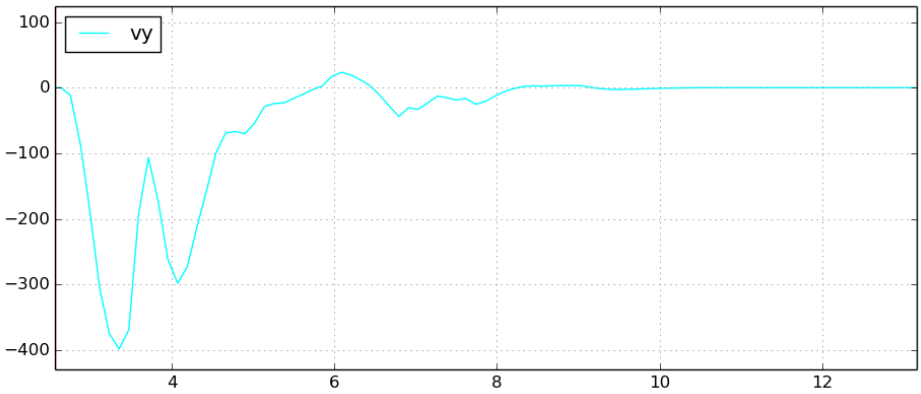
Dauguma agresyvumo konfigūracijų parodė panašų nusileidimo laiką tiek su „aukštu“, tiek su „labai aukštu“ algoritmo greitaveikos dažniais. Nėra poreikio naudoti „labai aukšto“ algoritmo greitaveikos dažnio, taip sutaupome skaičiavimo resursus ir drono baterijos įkrovą signalams siųsti. Todėl tolimesni tyrimai su realiu dronu buvo atliekami, būtent, su šia greitaveikos konfigūracija.

Atlikus detalesnius bepiločio orlaivio valdymo algoritmo agresyvumo parametrų testus, išryškėjo kelios agresyvumo konfigūracijos, kurios pademonstravo didesnį efektyvumą: „optimali“, „tarpinė“ ir „vidutinė“. Šios konfigūracijos buvo pasirinktos tolimesniems tyrimams su tikru dronu. Tyrimai su realia įranga parodė, kad šios konfigūracijos buvo per daug agresyvios realaus drono valdymui. Todėl, toliau mažinant agresyvumo parametrus, galiausiai buvo įvykdytas sėkmingas realaus drono nusileidimas ant palyginamojo žymeklio. Iš drono valdymo algoritmo agresyvumo parametrų, su kuriais buvo sėkmingai atliktas nusileidimas, buvo sukurta „reali“ agresyvumo konfigūracija. Su šia konfigūracija buvo atlikti testai simuliacijoje. Kaip matome iš lentelės 5 reikšmių, simuliacijoje ši konfigūracija parodo panašius rezultatus, kaip ir kitos efektyviausios drono valdymo algoritmo efektyvumo konfigūracijos.

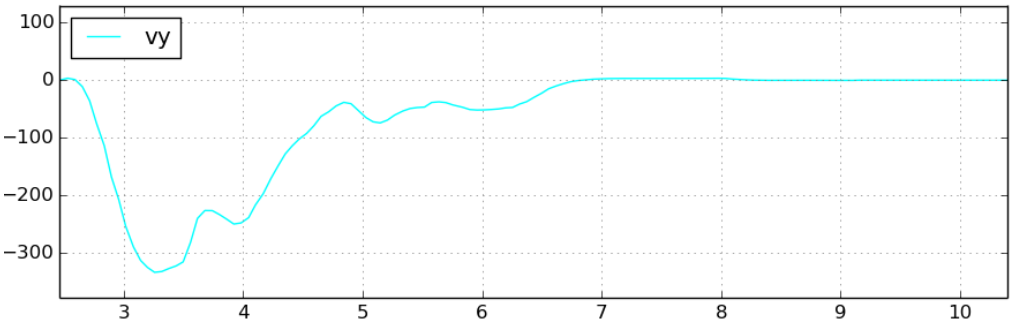
Atliekant tyrimus pastebėta, kad didinant algoritmo dažnį, su dauguma agresyvumo konfigūracijų, gerėja nusileidimo laikas ir švelnėja drono judesiai. Tai matome paveiksliukuose, kuriuose vaizduojamas drono judesių greitis. Juose pavaizduotas drono judesių greitis Y ašimi, atliekant nusileidimo manevrą ant palyginamojo markerio. Tyrimas atliktas su “realia” agresyvumo konfigūracija ir trimis skirtingais algoritmo greitaveikos dažniais: “žemu”, “aukštu” ir “labai aukštu”. Analizuojant grafikus, galima pastebėti, kad, perėjus iš “žemo” greitaveikos dažnio į “aukštą”, dronas nebedaro per didelio kompensavimo ir beveik nebekerta X ašies. 36 pav. vY reikšmė beveik nepatenka į teigiamų skaičių aibę, o tai reiškia, kad dronas visą laiką juda viena kryptimi. 35 pav. su žemesne alogritmo greitaveika, matomos teigiamos reikšmės, o tai reiškia, kad dronas pernelyg stipriai pajudėjo ir jam reikėjo sugrįžti į ankstesnę poziciją. Grafike, pavaizduotame 37 pav., matome drono greičius Y ašimi, naudojant “labai aukštą” algoritmo greitaveiką. Analizuojant grafiką, reiktų pastebėti, kad dronas niekada nekerta X ašies, nes visos grafiko reikšmės yra neigiamoje skaičių aibėje. Grafikas yra tolydesnis nei “aukšto” dažnio greitaveikos grafikas, o tai reiškia, kad drono judesiai yra tolygesni.



35 pav. Orlaivio judesių greitis Y ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę. "Realus" agresyvumas ir "žemas" dažnis.



36 pav. Orlaivio judesių greitis Y ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę. "Realus" agresyvumas ir "aukštas" dažnis.



37 pav. Orlaivio judesių greitis Y ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę. "Realus" agresyvumas ir "labai aukštas" dažnis.

5.5. Nusileidimo algoritmo žymeklio paieškos tyrimas

Pasinaudojus gautais rezultatais iš drono valdymo algoritmo agresyvumo parametrų ir algoritmo greitaveikos tyrimų, buvo sukurta kita tyrimų grupė - markerio paieškos tyrimai. Dronas su “aktyviu” markerio paieškos algoritmu skrenda iš taško A į tašką B ir tarp jų yra padėtas palyginamasis žymeklis, žymintis nusileidimo vietą. Drono pozicija A taške yra (0; 0; 2.8; 225), čia X=0, Y=0, Z=2.8, pokrypis = 225. Drono pozicija taške B yra (-3; -3; 2.8; 225), čia X=-3, Y=-3, Z=2.8, pokrypis = 225.

Tyrimai buvo atlikti su keliomis agresyvumo konfigūracijomis. “Standartinis” ir “aukštas” agresyvumai buvo pasirinkti, nes jie yra ribiniai agresyvumai. “Standartinis” agresyvumas yra suvaržytas X ir Y ašimis, o “aukštas” agresyvumas, labai agresyvus savo judesiais. Buvo įdomu sužinoti, kaip dronas elgsis su šiomis agresyvumo konfigūracijosmis skrydžio metu, aptikęs nusileidimo žymeklį. “Tarpinis” agresyvumas buvo pasirinktas todėl, kad geriausai pasirodė tyrimuose su nejudančia drono startine pozicija. “Reali” agresyvumo konfigūracija buvo pasirinkta, nes su ja pavyko įgyvendinti tikro drono nusileidimo manevrą realybėje, todėl norėjosi palyginti simuliacijos ir realaus skrydžio skirtumus. Detalesni tyrimų rezultatai yra pateikti priede Nr. 7. Tyrimai buvo atlikti naudojant „aukštą“ algoritmo greitaveiką.

7 lentelė. Agresyvumo parametrų įtaka rezultatams.

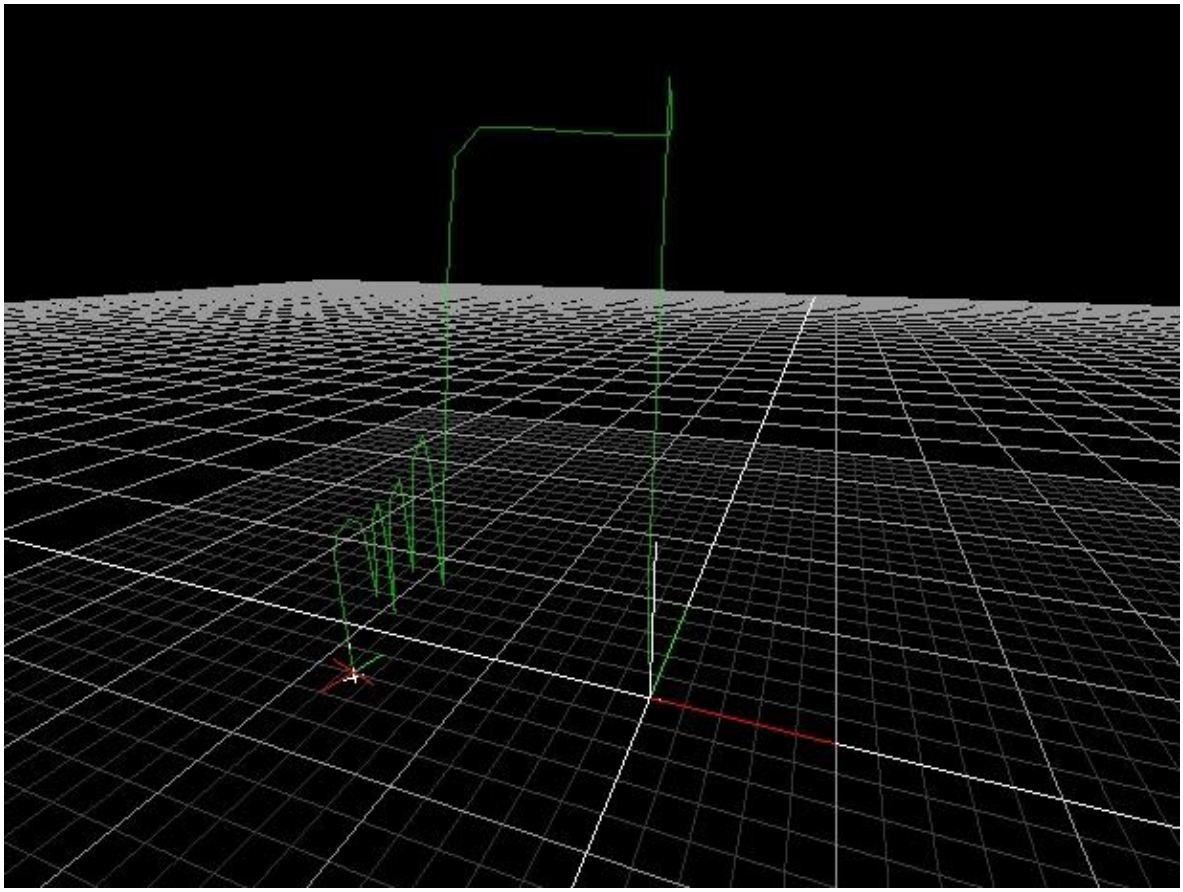
Agresyvumas	*Standartinis*	*Realus*	*Tarpinis*	*Aukštas*
Nusileidimo trukmė sekundėmis	39 s	6 s	5 s	10 s

Atlikus nusileidimo žymeklio paieškos tyrimus, išryškėjo panašios tendencijos kaip ir anksčiau atliktame agresyvumo konfigūracijų ir greitaveikos tyrime (7 lentelė). Geriausiai pasirodė “tarpinio” agresyvumo konfigūracija. Dronas, skrisdamas link taško B, ir pamatęs nusileidimo vietos markerį, ant palyginamojo žymeklio vidutiniškai nusileisdavo per 5 sekundes.

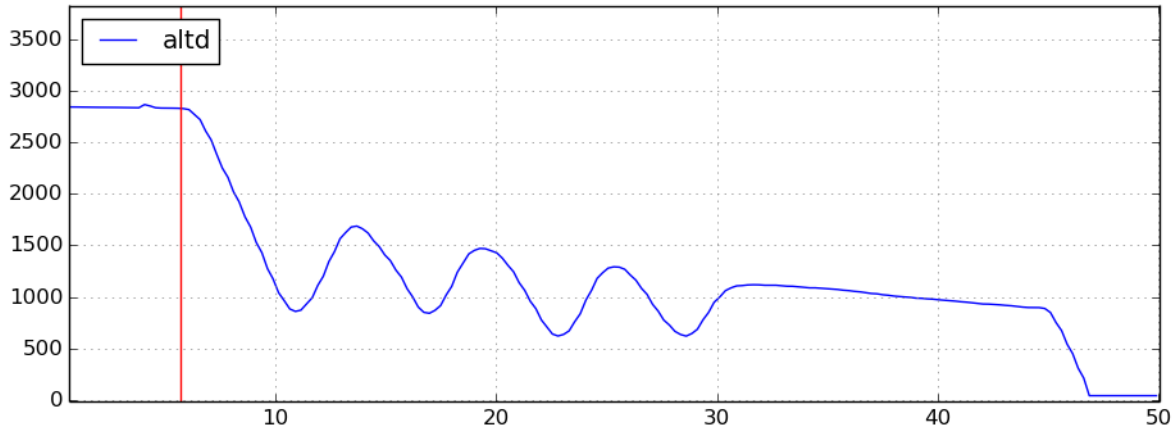
Bepilotis orlaivis, su “realia” agresyvumo konfigūracija simuliacijoje, nuo geriausio rezultato vidutiniškai atsiliko tik 1 sekunde. “Standartinis” ir “aukštas” agresyvumai parodė prastesnius rezultatus.

Dėl didelio judesių suvaržymo X ir Y ašimis, bepilotis orlaivis, su “standartine” agresyvumo konfigūracija, nepataikė nusileisti ant žymeklio iš pirmo karto. Tai matome drono judesių trajektorijoje (38 pav.) ir drono aukščio grafike (39 pav.). Dronas, pradėjęs eksperimentą, pakyla į 2,8 metro aukštį ir pradeda judėti į tašką B, kuris yra nutolęs per 3 metrus X ašimi ir 3 metrus Y ašimi. Pamatęs nusileidimo vietos marker, leidžiasi link jo, bet kelis kartus nepataiko, todėl pakyla aukštyn ir bando leistis vėl. Su “standartiniu” agresyvumu bepilotis orlaivis užtruko 39 sekundes

nuo markerio aptikimo iki nusileidimo ant jo. Nusileidimo aukščio grafike (27 pav.) raudona vertikali linija indikuoja laiką, kai bepiločio orlaivio valdymo algoritmas aptiko nusileidimo vietos žymeklį.

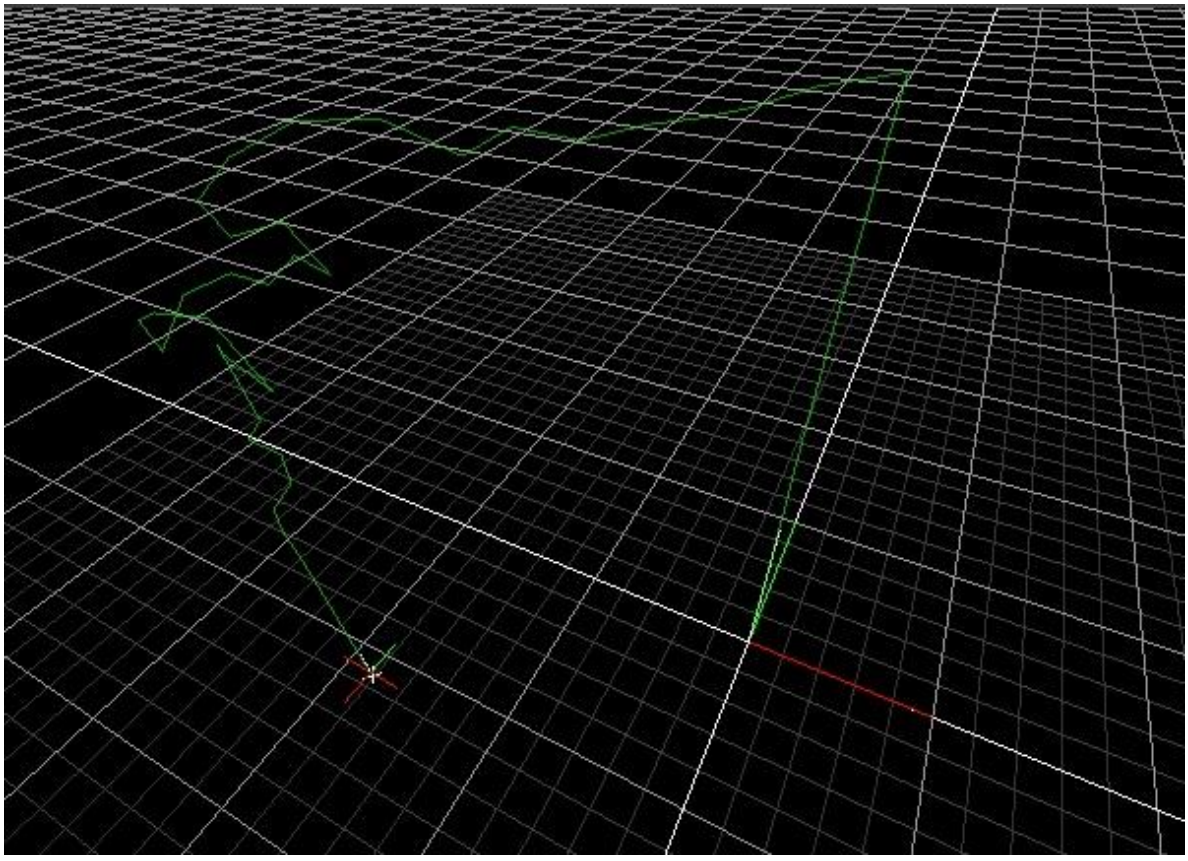


38 pav. Drono judėjimo trajektorija.”Standartinis” agresyvumas.

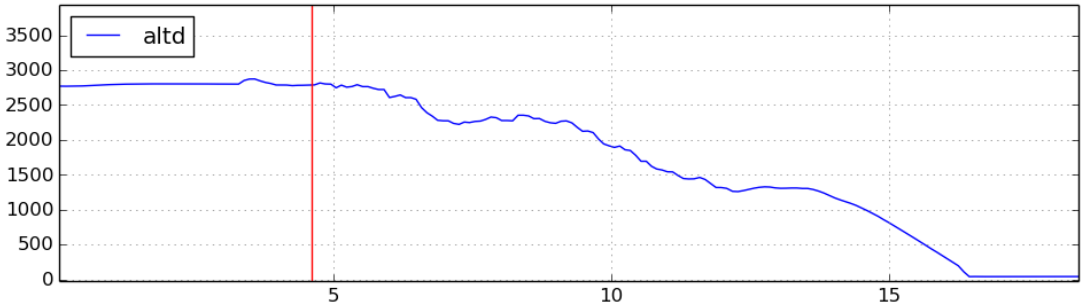


39 pav. Orlaivio nusileidimo aukščio grafikas. X ašyje – laikas sekundėmis, Y ašyje - drono aukštis milimetrais. ”Standartinis” agresyvumas.

Atliekant markerio paieškos manevrą su “aukšto” agresyvumo konfigūracija, dronas, nuo markerio aptikimo iki nusileidimo ant jo, užtruko 10 sekundžių. Nuo geriausio rezultato, kurį parodė „tarpinis“ agresyvumas, atsiliko 4 sekundėmis. Analizuojant skrydžio trajektoriją (40 pav.) matome, kad dronas pakilo į 3 metrų aukštį ir pradėjo skristi link taško B, skrydžio metu aptikęs nusileidimo taško žymeklį, pradeda ant jo leistis. Dėl agresyvių leidimosi manevrų dronas dažnai pajuda X arba Y ašimi per daug stipriai ir po to bando grįžti į tinkamą trajektoriją. Nors bepilotis orlaivis leidžiasi Z ašimi pakankamai tolygiai (41 pav.), per daug agresyvūs manevrai X ir Y ašimis prailgina nusileidimo trajektoriją ir dronas, besileisdamas ant žymeklio, užtrunka 40% ilgiau, nei tai darydamas su „tarpinio“ agresyvumo konfigūracija.

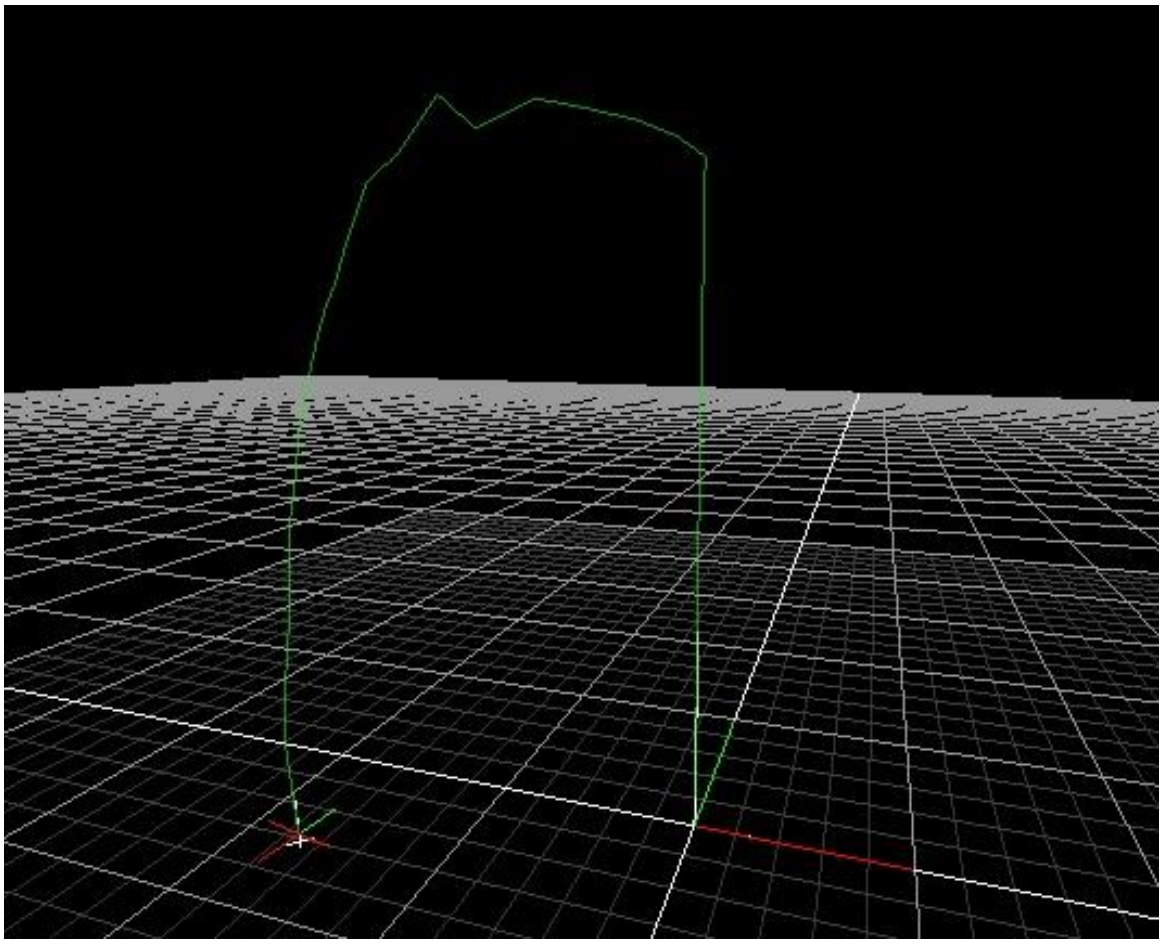


40 pav. Drono judėjimo trajektorija.”Aukštas” agresyvumas.

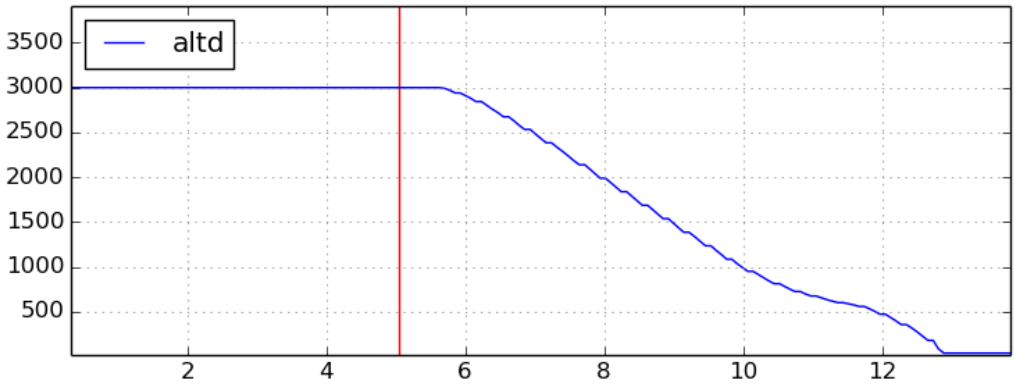


41 pav. Orlaivio nusileidimo aukščio grafikas. X ašyje – laikas sekundėmis, Y ašyje - drono aukštis milimetrais. ”Aukštas” agresyvumas.

Atliekant markerio paieškos tyrimą simuliacijoje, su tais pačiais agresyvumo parametrais, su kuriais pavyko atlikti tikro drono nusileidimą realybėje, ši “reali” agresyvumo konfigūracija pasirodė gerai ir nuo geriausio rezultato atsiliko tik 1 sekunde. Nuo nusileidimo žymeklio pamatymo iki nusileidimo ant jo, su šia konfigūracija, dronas užtruko 6 sekundes. Su šia agresyvumo konfigūraciją bepilotis orlaivis link nusileidimo taško, Z ašimi judėjo pakankamai tolygiai (43 pav.). Nusileidmo manevro trajektorija taipogi buvo tolydi (42 pav.).



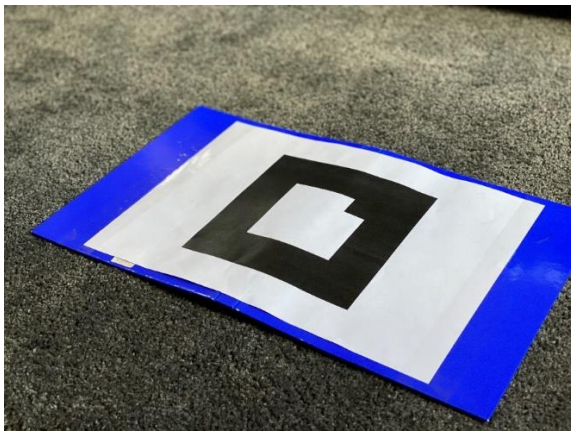
42 pav. Drono judėjimo trajektorija.”Realus” agresyvumas.



43 pav. Orlaivio nusileidimo aukščio grafikas. X ašyje – laikas sekundėmis, Y ašyje - drono aukštis milimetrais. ”Realus” agresyvumas.

5.6. Valdymo modulio tyrimas su tikru dronu

Buvo atliktas drono valdymo tyrimas su tikru dronu lauke. Nusileidimo taško žymeklis buvo atspausdintas ir pritvirtintas prie storo kartono lakšto (44 pav.), kad nebūtų nupūstas bepiločio orlaivio varikliukų sukuriamos keliamosios jėgos. Drono standartinės baterijos greit praranda dalį savo įkrovos ir pakanka tik maždaug 8 minučių skrydžiams. Todėl buvo nupirkta papildoma didesnės talpos baterija, skirta RC orlaiviams. Nors jos įtampos ir srovės parametrai atitinka AR.Drone baterijų parametrus, bet jos jungtis netiko. AR.Drone naudoja nestandartinę dviejų kaiščių modifikuotą Molex jungtį. Naujoji baterija naudoja standartinę XT60 jungtį. Norint panaudoti naująją bateriją su AR.Drone platforma, teko sulituoti perėjimą iš Molex jungties į XT60 jungtį. Tiesa, naujoji jungtis su baterija užėmė šiek tiek daugiau vietos, nei standartinės baterijos, todėl ant drono nebetilpo apsauginiai korpusai. 45 paveiksliuke matome AR.Drone bepilotį orlaivį su nauja baterija ir jos jungtimi.

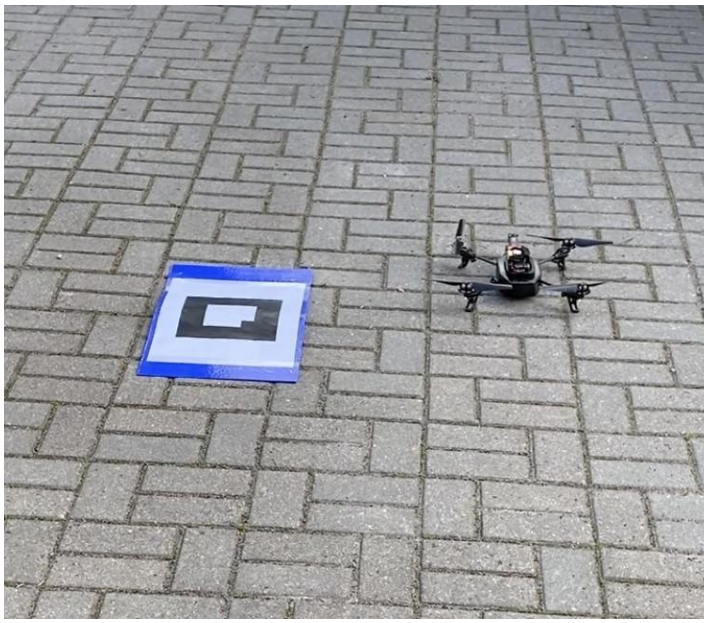


44 pav. Nusileidimo taško palyginamasis žymeklis.

Tyrimai su tikru dronu buvo atlikti lauke, nusileidimo taško žymeklį padėjus ant pilkos spalvos trinkelių (46 pav.). Kaip ir simuliaciniuose tyrimuose, dronas buvo nuskraidinamas įstrižai, virš nusileidimo markerio. Drono valdymo algoritmas buvo sukonfigūruotas su 25 cm nusileidimo paklaidomis, X ir Y ašims. Galutinio nusileidimo manevras, kai dronas tolygiai leidžiasi žemyn, buvo atyvuojamas dronui pasiekus 50cm aukštį, virš nusileidimo taško.

45 pav. Dronas su nauja baterija ir savadarbe jungtimi.

Pirminiai tyrimai su tikru dronu ir “vidutiniu” agresyvumu buvo nesėkmingi. Bepilotis orlaivis perdaug agresyviai artėdavo prie nusileidimo taško ir pamesdavo jį beartėdamas. Panašiai elgėsi dronas ir su “tarpiniu” agresyvumo nustatymu. Pradėjus bandymus su “optimalia” agresyvumo konfigūracija, drono nusileidimo ant markerio trajektorija pagerėjo. Tačiau bepilotis orlaivis, besileisdamas, vis dar pamesdavo nusileidimo žymeklį, dėl pernelyg agresyvių judesių X ir Y ašimis. “Optimali” agresyvumo konfigūracija naudoja parametrus K_rp = 10 ir K_direct = 5. Šiuos agresyvumo parametrus sumažinus maždaug dvigubai, kai K_rp = 5 ir K_direct = 3, bepilotis orlaivis sugebėjo autonomiškai nusileisti ant palyginamojo žymeklio. Šie agresyvumo parametrai darbe įvardijami kaip “reali” agresyvumo konfigūracija. “Realus” agresyvumas buvo nustatytas bandymų metu su tikru dronu ir vėliau retrospektyviai padaryti tyrimai simuliacijoje. Taip mes galime palyginti realaus drono ir simuliacijos skirtumus su tuo pačiu valdymo algoritmu.



46 pav. Dronas ir palyginamasis žymeklis lauke.

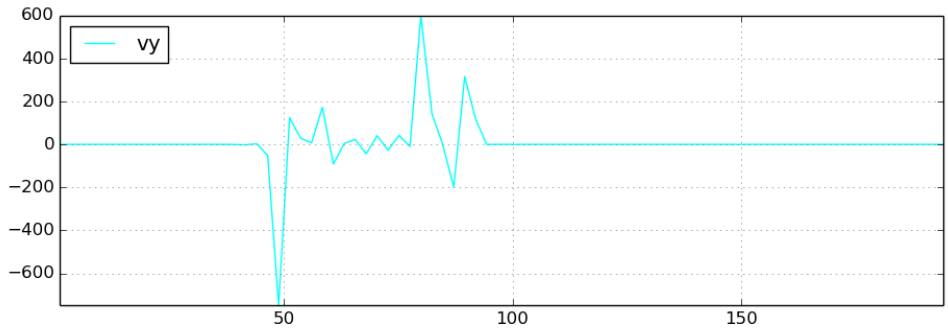
Sėkmingi bepiločio orlaivio nusileidimai ant palyginamojo žymeklio, su “realiu” agresyvumo nustatymu, buvo pakartoti dar kelis kartus. Palyginamojo žymeklio aptikimo algoritmas žymeklį aptinka pakankamai gerai ir greitai, net su žemos rezuliucijos apatine drono kamera (47 pav.). Jis aptinkamas vos visas pasirodęs drono kameros vaizdo aprėptyje. Pastebėta, kad didelę įtaką markerio atpažinimui turi saulės apšvietos intensyvumas. Net ir šviesią debesuotą dieną markeris aptinkamas iš mažesnio atstumo, nei dieną su tiesioginiais saulės spinduliais. Deja, lokaliai apšvietai išmatuoti reikia specialios įrangos, todėl detalesni tyrimai šiuo aspektu nebuvo atlikti.

47 pav. Dronas eksperimento metu ir jo matomas vaizdas iš apatinės kameros.

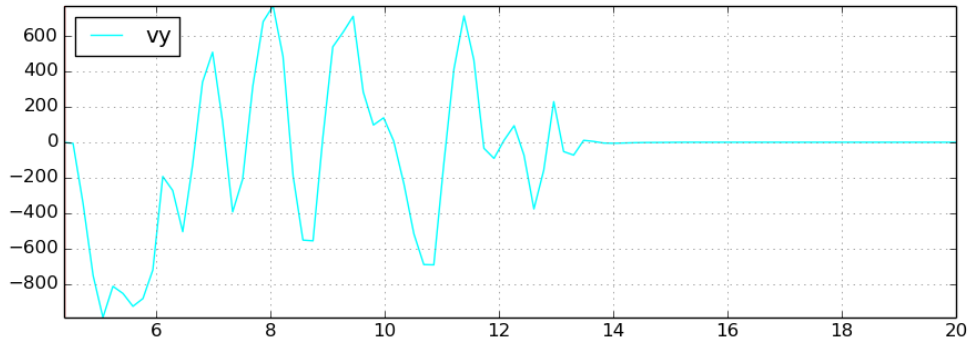
Apžvelkime vieno iš pavykusių bepiločio orlaivio nuleidimo ant palyginamojo žymeklio, skrydžio eigą. Dronas buvo pakeltas į maždaug trijų metrų aukštį, su išjungtu autopiloto moduliu. Autopilotas buvo išjungtas tam, kad dronas neperimtų drono valdymo, bekylant į pradžios koordinates, jam netyčia aptikus nusileidimo žymeklį apatinės kameros vaizdo sraute. Įjungus autopiloto modulį, dronas pradeda judėti link vaizdo sraute aptikto palyginamojo žymeklio. Beveik priartėjęs prie nusileidimo taško, valdymo modulis trumpam pameta nusileidimo tašką, todėl dronas trumpam kilsteli aukštyn (48 pav.), bet vėl, aptikęs nusileidimo taško žymeklį, užbaigia nusileidimo ant palyginamojo žymeklio manevrą.

48 pav. Orlaivio nusileidimo aukščio grafikas. X ašyje – laikas sekundėmis, Y ašyje - drono aukštis milimetrais.

Įdomu palyginti realaus drono ir simuliuojamo drono elgesį. Galime išanalizuoti realaus drono greičius Y ašimi (49 pav.) ir drono simuliacijoje greičius Y ašyje (50 pav.). Realaus drono judesių grafikas paimtas iš, prieš tai aprašyto, sėkmingo nusileidimo sensorių parodymų. Tikro drono grafikas sugeneruotas drono valdymui naudojant "realaus" agresyvumo konfigūraciją ir "aukštą" greitaveikos dažnį. Simuliuojamo drono grafikas buvo sugeneruotas iš nusileidimo ant palyginamojo žymeklio manevro, bepiločio orlaivio valdymo algoritmui veikiant su "aukšto" agresyvumo konfigūracija ir "aukštu" greitaveikos dažniu.



49 pav. Orlaivio judesių greitis Y ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę.

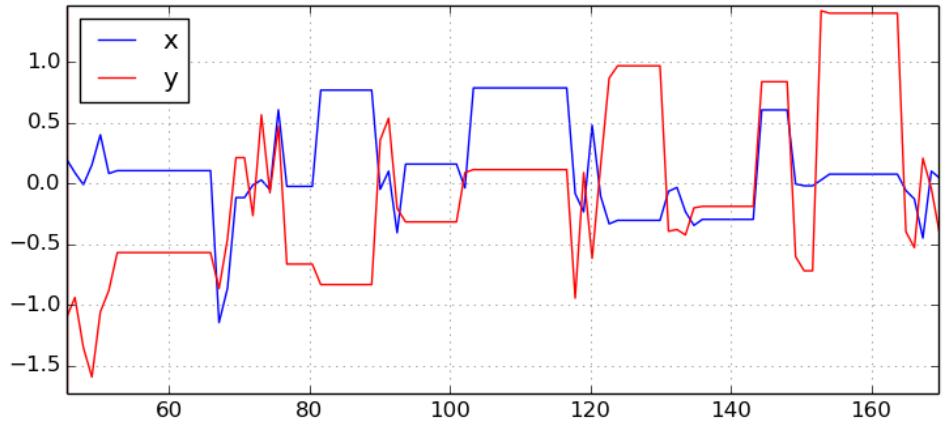


50 pav. Orlaivio judesių greitis Y ašimi. X ašyje – laikas sekundėmis, Y ašyje - drono greitis milimetrais per sekundę.

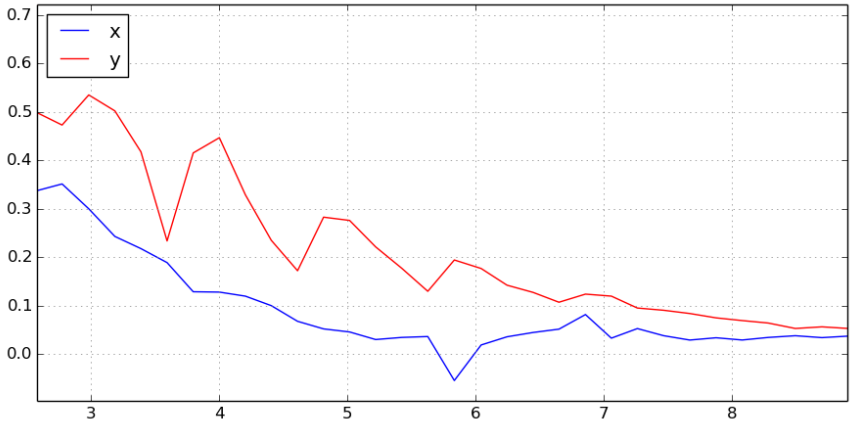
Lyginant bepiločio orlaivio manevrus realybėje, su pakankamai nedideliais agresyvumo nustatymais, ir simuliuojamo drono manevrus Gazebo simuliacijoje, su dideliais agresyvumo nustatymais, matome, kad maksimalios judesių reikšmės yra panašiuose rėžiuose. Realus dronas juda nuo -700 mm/s iki 600 mm/s Y ašimi. Simuliuojamas dronas juda nuo -900 mm/s iki 700 mm/s Y ašimi. Tiesa, realus dronas pasiekęs X ašį greitai nuslopina savo judesius ir stabilizuojasi ties tikslu, o simuliuojamas dronas persistengia ir todėl pradeda osciliuoti, bandydamas pasiekti stabilią X ašies padėtį. Iš šių grafikų galima daryti išvadą, jog realybėje bepilotis orlaivis yra jautresnis judėjimo parametrams. Todėl, prieš perkeliant valdymo algoritmus, sukurtus su šiuo

programinės įrangos sprendimu iš simuliacijos į realybę, reikėtų pasirinkti drono valdymo parametrus, kurie nėra linkę jo manevrus daryti agresyviais.

Realaus drono judesių jautrumą agresyvumo parametrams galima matyti iš nusileidimo taško koordinačių grafiko (51 pav.). Matomas akivaizdus netolygumas, lyginant su simuliuojamo drono (52 pav.). Šiuose grafikuose vaizduojamas poslinkis per X ir Y ašis nuo drono iki nusileidimo taško.



51 pav. X ašyje – laikas sekundėmis, Y ašyje - drono atstumas iki nusileidimo taško metrais.



52 pav. X ašyje – laikas sekundėmis, Y ašyje - drono atstumas iki nusileidimo taško metrais.

Rezultatai ir išvados

Šio darbo metu buvo sukurtas atviro kodo[1] programinės įrangos sprendimas, skirtas dronų nusileidimo testavimui tiek simuliacinėje aplinkoje, tiek realybėje su tikru dronu. Šis programinės įrangos sprendimas susideda iš kelių dalių. Pirma dalis, tai Gazebo simuliacijai skirti „world“ ir „URDF“ tipo failai, aprašantys droną ir jo nusileidimo užduoties aplinką simuliacijoje. Antroji dalis, tai „ard_autopilot“ drono valdymo modulis, parašytas naudojant ROS karkasą. Šis modulis yra sukurtas kaip ROS paketas, todėl programinės įrangos kūrėjai gali jį lengvai panaudoti kitose ROS sistemose. Trečia dalis, tai konfigūraciniai ir „launch“ tipo failai, kurie sujungia visus ROS sistemos modulius į vieną programinės įrangos paketą. Viso šio programinės įrangos sprendimo išeities kodas yra pasiekiamas GitHub[1] platformoje. Kodas yra dokumentuotas, todėl lengvai gali būti naudojamas kitų tyrėjų darbams.

Panaudojus šį programinės įrangos sprendimą buvo atliktas pirminis tyrimas, kurio metu bandoma nustatyti, kaip drono nusileidimo ant palyginamojo žymeklio laikas ir sėkmė priklauso nuo drono judesių agresyvumo X, Y, Z ir pokrypio ašimis. Tyrimai buvo atlikti Gazebo simuliacinėje aplinkoje. Buvo paneigtos prieš tyrimą padarytos prielaidos. Kitaip nei manyta, mokslo tiriamojo darbo antrojoje dalyje, agresyvūs judesiai ties X ir Y ašimis nepadidino markerio pametimo dažnio. Per daug stabilizuoti judesiai ties X ir Y ašimis, nors ir stabilizavo drono judėjimą, bet prailgino jo nusileidimo laiką. Suvaržius drono manevrų agresyvumą itin stipriai, jis negalėjo atlikti nusileidimo ant markerio užduoties. Geriausias drono nusileidimo ant lyginamojo žymeklio rezultatas, atliekant pirminius tyrimus, buvo pasiektas pasirinkus X ir Y ašių agresyvumą, kai K_rp yra 10 ir Z ašies agresyvumą, kai K_direct yra 5. Dronas, kybantis 3 metrų aukštyje, nuo markerio aptikimo iki nusileidimo ant markerio užtruko 8 sekundes. Tai buvo optimaliausia drono nusileidimo konfigūracija, pademonstravusi trumpiausią laiką, atliekant pirminius tyrimus.

Naudojant simuliuojamą droną buvo atlikti drono valdymo algoritmo greitaveikos tyrimai. Kitaip nei nustatyta pirminiuose tyrimuose, didinant drono valdymo algoritmo greitaveiką, su dauguma jo valdymo parametrų konfigūracijų, drono valdymo efektyvumas didėja. Tai reiškia, kad dronas su didesniu greitaveikos dažniu greičiau nusileidžia ant aptikto nusileidimo taško. Šio eksperimento metu geriausi rezultatai buvo pasiekti su 10Hz jutiklių nuskaitymo dažniu ir 1Hz bepiločio orlaivio judesių komandų dažniu. Toliau didinant greitaveikos dažnius rezultatai išliko tokio pačio efektyvumo su vidutinio agresyvumo (K_direct = 5, K_rp = 10) drono judesių konfigūracija.

[1] https://github.com/DariusMiskinis/ard_autopilot

Atlikus algoritmo agresyvumo parametrų pirminius tyrimus ir algoritmo greitaveikos pirminius tyrimus, buvo nutarta atlikti eksperimentą, kuris detaliau ištirtų šių parametrų tarpusavio sąveiką. Kaip ir anksčiau darytame tyrime buvo pastebėta, kad drono valdymo algoritmo greitaveikos didinimas trumpina drono nusileidimo and palyginamojo žymeklio laiką. Kai K_direct = 5, o K_rp = 10, padidinus algoritmo greitaveikos dažnį nuo 1Hz iki 10Hz, drono nusileidimo laikas sutrumpėjo nuo 48 sekundžių iki 6 sekundžių. Tačiau atlikus detalesnius tyrimus pastebėta, kad pernelyg didelė greitaveika neduoda naudos, kai K_direct = 15, o K_rp = 15, padidinus dažnį nuo 30Hz iki 100Hz, nusileidimo laikas nepasikeitė. Su kai kuriomis konfigūracijomis agresyvumo spektro kraštuose, aukštas greitaveikos dažnis blogina drono užduoties atlikimo rezultatus. Naudojant aukšto agresyvumo konfigūraciją (K_direct = 50, K_rp = 50) ir padidinus algoritmo greitaveikos dažnį nuo 30Hz iki 100Hz, drono nusileidimo efektyvumas sumažėjo 33.3%, bepiločio orlaivio nusileidimo laikas išaugo nuo 9 sekundžių iki 12 sekundžių.

Simuliacijoje buvo atlikti drono tyrimai žymeklio paieškos režime, kai bepilotis orlaivis ne kybo vietoje, ieškodamas palyginamojo žymeklio, bet skrenda iš taško A į tašką B ir pakeliui, aptikęs nusileidimo tašką, bando leistis. Šiame tyrime panaudotos konfigūracijos iš ankstesnių tyrimų parodė, kad didelio skirtumo tarp valdymo konfigūracijų panaudojimo kybant ir judant nėra. Naudojant bepiločio orlaivio agresyvumo konfigūraciją, pavadinimu “Tarpinis” (K_direct = 15, K_rp = 15), kuri prieš tai atliktuose tyrimuose su kybančiu dronu parodė vienus geriausių rezultatų, šiame žymeklio paieškos eksperimente buvo pasiektas toks pats nusileidimo ant palyginamojo žymeklio laikas. Abiem atvejais bepilotis orlaivis užtruko 5 sekundes leisdamasis iš 2,8 metrų aukščio.

Galiausiai, simuliacijoje ištyrus ir išrinkus tinkamas bepiločio orlaivio valdymo parametrų konfigūracijas, jos buvo išbandytos ant tikro bepiločio orlaivio lauko sąlygomis. Tiesiai iš simuliacijos perkeltoms konfigūracijoms nesisekė iš karto sėkmingai įvykdyti autonominio drono nusileidimo ant palyginamojo žymeklio. Modifikavus valdymo parametrų konfigūraciją “Optimalus” (K_direct = 5, K_rp = 10), kuri rodė daugiausiai progreso, ir sumažinus bepiločio orlaivio judesių agresyvumą (K_direct = 3, K_rp = 3), pavyko sėkmingai atlikti autonominį tikro drono nusileidimą ant aptikto palyginamojo žymeklio.

Bepiločio orlaivio valdymo parametrų konfigūracija, su kuria pavyko sėkmingai nutupdyti droną and palyginamojo žymeklio, buvo retrospektyviai ištirta simuliacijoje. Ši konfigūracija simuliacijoje parodė tokius pačius rezultatus, kaip ir geriausios simuliacijai pritaikytos konfigūracijos.

Atlikus realaus bepiločio orlaivio ir simuliuojamo bepiločio orlaivio sensorių palyginimus, galima daryti išvadą, kad su konkrečiu programinės įrangos sprendimu reikia naudoti mažiau agresyvius valdymo parametrus, nes realybėje bepilotis orlaivis į juos reaguoja stipriau.

Atliktas darbas demonstruoja, kad galima sukurti bepiločio orlaivio valdymo modulį ir jį ištestuoti, naudojant simuliacinę aplinką, joje parenkant pirminius valdymo parametrus. Šiuos parametrus perkėlus į tikrą droną ir atlikus kelis testinius skrydžius jų reikšmėms finalizuoti, simuliacijoje sukurtą valdymo modulį galima naudoti tikro drono valdymui.

Šio darbo pagrindiniai rezultatai ir išvados:

1. Panaudojant ROS karkasą buvo sukurtas atviro kodo programinės įrangos paketas, skirtas dronų nusileidimo testavimui tiek simuliacinėje aplinkoje, tiek realybėje su tikru dronu. Šis programinės įrangos spredimas pasiekiamas GitHub platformoje.
2. Su šiuo programinės įrangos srendimu buvo atlikti pirminiai tyrimai, kurių metu buvo bandoma nustatyti, kaip drono nusileidimo laikas ir sėkmė priklauso nuo drono judesių agresyvumo X, Y, Z ir pokrypio ašimis. Buvo paneigtos prieš tyrimą padarytos prielaidos. Kitaip nei manyta, mokslo tiriamojo darbo antrojoje dalyje, vidutinio agresyvumo judesiai ties X ir Y ašimis nepadidino markerio pametimo dažnio.
3. Buvo atlikti drono valdymo algoritmo greitaveikos tyrimai. Kitaip nei nustatyta pirminiuose tyrimuose, didinant drono valdymo algoritmo greitaveiką, su dauguma jo valdymo parametrų konfigūracijų, dronas greičiau nusileidžia ant aptikto nusileidimo taško.
4. Atlikus algoritmo agresyvumo parametrų pirminius tyrimus ir algoritmo greitaveikos pirminius tyrimus, buvo nutarta atlikti eksperimentą, kuris detaliau ištirtų šių parametrų tarpusavio sąveiką. Nustatyta, kad greitaveikos didinimas su dauguma agresyvumo konfigūracijų sutrumpina drono nusileidimo laiką. Tačiau egzistuoja riba kai greitaveikos didinimas nebegerina drono nusileidimo laiko. O su agresyvumo konfigūracijomis, kurios yra sprektro kraštuose, greitaveikos didinimas nuo tam tikros ribos netgi blogina drono nusileidimo efektyvumą.
5. Simuliacijoje buvo atlikti drono tyrimai žymeklio paieškos režime, kai bepilotis orlaivis ne kybo vietoje, ieškodamas palyginamojo žymeklio, bet skrenda iš taško A į tašką B ir pakeliui, aptikęs nusileidimo tašką, bando leistis. Šiame tyrime panaudotos konfigūracijos iš ankstesnių tyrimų parodė, kad didelio skirtumo tarp valdymo konfigūracijų panaudojimo kybant ir judant nėra.

6. Simuliacijoje ištyrus ir išrinkus tinkamas bepiločio orlaivio valdymo parametrų konfigūracijas, jos buvo išbandytos ant tikro bepiločio orlaivio lauko sąlygomis.
7. Bepiločio orlaivio valdymo parametrų konfigūracija, su kuria pavyko sėkmingai nutupdyti droną and palyginamojo žymeklio, buvo retrospektyviai ištirta simuliacijoje.
8. Atlikus realaus bepiločio orlaivio ir simuliuojamo bepiločio orlaivio sensorių palyginimus, galima daryti išvadą, kad su konkrečiu programinės įrangos sprendimu reikia naudoti mažiau agresyvius valdymo parametrus, nes realybėje bepilotis orlaivis į juos reaguoja stipriau.
9. Atliktas darbas demonstruoja, kad galima sukurti bepiločio orlaivio valdymo modulį, naudojant ROS karkasą. Šį modulį ištestuoti, naudojant simuliacinę aplinką, joje parenkant pirminius valdymo parametrus. Šiuos parametrus perkėlus į tikrą droną ir atlikus kelis testinius skrydžius jų reikšmėms finalizuoti, simuliacijoje sukurtą valdymo modulį galima naudoti tikro drono valdymui.

Šaltinių sąrašas

[BCS11a] Cooper Bills, Joyce Chen ir Ashutosh Saxena. Autonomous mav flight in indoor environments using single image perspective cues. 2011 IEEE International Conference on Robotics and Automation:5776–5783, 2011.

[BCV+11] Pierre-Jean Bristeau, François Callou, David Vissière, Nicolas Petit. The Navigation and Control technology inside the AR.Drone micro UAV. 18th IFAC World Congress, Milano, Italy, 2011.

[BHO14a] Roman Barták, Andrej Hrasko ir David Obdrzálek. A controller for autonomous landing of ar.drone. *The 26th Chinese Control and Decision Conference* (2014 CCDC):329–334, 2014.

[BHO14b] Roman Barták, A. Hraško ir D. Obdržálek. On autonomous landing of ar.drone: handson experience. *Proceedings of the 27th International Florida Artificial Intelligence Research Society Conference*, FLAIRS 2014:400–405, 2014-01.

[Buc16] R. Büchi. *Fascination Quadcopter*. Books on Demand, 2016.

[CWE15] Stephen M Chaves, Ryan W Wolcott ir Ryan M Eustice. Neec research: toward gpsdenied landing of unmanned aerial vehicles on ships at sea. *Naval Engineers Journal*, 127(1):23–35, 2015.

[CVR+18] Adrian Carrio, Sai Vemprala, Andres Ripoll, Srikanth Saripalli ir Pascual Campoy. Drone detection using depth maps. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*:1034–1037, 2018.

[DV11] Nick Dijkshoorn ir Arnoud Visser. Integrating sensor and motion models to localize an autonomous ar. drone. *International Journal of Micro Air Vehicles*, 3:183–200, 2011.

[ESD12a] Jakob Engel, Jurgen Sturm ir Daniel Cremers. Camera-based navigation of a lowcost quadrocopter. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 2815–2821, 2012.

[FBA+16] Fadri Furrer, Michael Burri, Markus Achtelik ir Roland Siegwart. *Rotors—a modular gazebo mav simulator framework. Robot Operating System (ROS): The Complete Reference (Volume 1)*. Anis Koubaa, redaktorius. Springer International Publishing, Cham, 2016, p. 595–625.

[Fia04] Fiala M. ARTag, An Improved Marker System Based on ARToolkit, 2004.

[Fow16] Ajibola S Fowowe. Trajectory Tracking Control of a Quadrotor During Cooperative Operation Between UAV and UGV, 2016.

[KBP+00] Hirokazu Kato, Mark Billinghurst, I. Poupyrev, K. Imamoto ir K. Tachibana. Virtual object manipulation on a table-top ar environment. *Proceedings of Int. Symp. on Augmented Reality 2000*, p. 111–119, 2000.

[KH04] Nathan P. Koenig ir Andrew Howard. Design and use paradigms for gazebo, an opensource multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 3:2149–2154 vol.3, 2004.

[Kou19] Anis Koubaa. *Robot Operating System (ROS) The Complete Reference*. Springer Publishing Company, 2019.

[KTL+12] Geert-Jan M. Kruijff, Viatcheslav Tretyakov, Thorsten Linder, Fiora Pirri ir k.t. Rescue Robots at Earthquake-Hit Mirandola, Italy: a Field Report. *2012 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, p. 1–8, 2012.

[KVF+11] Tomáš Krajník, Vojtech Vonasek, Daniel Fišer ir Jan Faigl. Ar-drone as a platform for robotic research and education. *Communications in Computer and Information Science*, tom. 161, 2011.

[Len18] Joseph Lentin. *Robot Operating System (ROS) for Absolute Beginners*, Apress, 2018.

[LMZ13] Jacobo Lugo, Andreas Masselli ir Andreas Zell. Following a quadrotor with another quadrotor using onboard vision. *2013 European Conference on Mobile Robots, ECMR 2013 - Conference Proceedings*, p. 26–31, 2013.

[LSP09] S. Lange, N. Sunderhauf ir P. Protzel. A vision based onboard approach for landing and position control of an autonomous multirotor uav in gps-denied environments. *2009 International Conference on Advanced Robotics*, p. 1–6, 2009.

[LZ13] Jacobo Jimenez Lugo ir Andreas Zell. Framework for autonomous on-board navigation with the ar.drone. *Journal of Intelligent Robotic Systems*, 73:401–412, 2013.

[MJ15] Chen Minhua ir Guo Jiangtao. Mobile robotics lab spring 2015 automatic landing on a moving target, 2015.

[MPT02] Pierre Malbezin, Wayne Piekarski, Bruce H. Thomas. Measuring ARToolKit Accuracy in Long Distance Tracking Experiments. *In 1st Int'l Augmented Reality Toolkit Workshop*, 2002.

[MSK+12] Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar von Stryk. Comprehensive Simulation of Quadrotor UAVs using ROS and Gazebo, 2012.

[MYW+13] Andreas Masselli, Shaowu Yang, Karl Wenzel ir Andreas Zell. A cross-platform comparison of visual marker based approaches for autonomous flight of quadrocopters. *Journal of Intelligent Robotic Systems*, 73:1–9, 2013.

[NAC+18] North, Ahmad Alissa, Josh Cooper, Adnan Rashied, Eric Rawls, Jason Walters, Utku "Victor" Sahin, Kade Randell, Cheyenne Sancho. Performance Analysis of Brain Control Interface in Drone Applications, 2018.

[OKV15] Miguel A Olivares-Mendez, Somasundar Kannan ir Holger Voos. Vision based fuzzy control autonomous landing with uavs: from v-rep to real experiments. *2015 23rd Mediterranean Conference on Control and Automation (MED)*, p. 14–21. IEEE, 2015.

[PBC14] Johann-Sebastian Pleban, Ricardo Band ir Reiner Creutzburg. Hacking and securing the AR.Drone 2.0 quadcopter: investigations for improving the security of a toy. Mobile Devices and Multimedia: Enabling Technologies, Algorithms, and Applications 2014, tom. 9030, p. 168–179. International Society for Optics ir Photonics, SPIE, 2014.

[RPR14] Luis E. Romero, David F. Pozo, Jorge A. Rosales. Quadcopter stabilization by using PID controllers, 2014.

[RS15] Anze Rezelj ir Danijel Skocaj. Autonomous charging of a quadcopter on a mobile platform. *Proceedings Austrian Robotics Workshop 2015*, p. 65–66, 2015.

[Sel19] Wil Selby. ROS Integration.
[žiūrėta 2019-08-07]. Prieiga per Internetą:
<https://www.wilselby.com/research/ros-integration/>

[SH16] Adrienne Welch Sudbury ir E. Bruce Hutchinson. A Cost Analysis of Amazon Prime Air (Drone Delivery*). Journal for Economic Educators*, 16(1):1–12, 2016.

[SK17] Mohammad Fattahi Sani ir Ghader Karimian. Automatic navigation and landing of an indoor ar. drone quadrotor using aruco marker and inertial sensors. 2017 International Conference on Computer and Drone Applications (IConDA), p. 102–107. IEEE, 2017.

[SKF+12] M. Saska, T. Krajník, J. Faigl, V. Vonásek ir L. Přeučil. Low cost mav platform ar-drone in experimental verifications of methods for vision based autonomous navigation. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 4808–4809, 2012.

[SNR18] Mahdi Shavarani, Mazyar Ghadiri Nejad, Farhood Rismanchian ir Gokhan Izbirak. Application of hierarchical facility location problem for optimization of a drone delivery system: a case study of amazon prime air in the city of san francisco. *The International Journal of Advanced Manufacturing Technology*, 95:3141–3153, 2018.

[SOT+17] Yuki Sato, Shingo Ozawa, Yuta Terasaka, Masaaki Kaburagi ir k.t. Remote radiation imaging system using a compact gamma-ray imager mounted on a multicopter drone. *Journal of Nuclear Science and Technology*, 55:1–7, 2017.

[SS18] Judy Scott ir Carlton Scott. Models for drone delivery of medications and other healthcare items. *International Journal of Healthcare Information Systems and Informatics*, 13:20–34, 2018.

[Stu18] Jürgen Sturm. Visual Navigation for Flying Robots.
[žiūrėta 2018-05-20]. Prieiga per Internetą:
<http://vision.in.tum.de/teaching/ss2012/visnav>

[VCM+16] G. Vasconcelos, G. Carrijo, R. Miani, J. Souza ir V. Guizilini. The impact of dos attacks on the ar.drone 2.0. *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*, p. 127–132, 2016.

[Was19a] Introduction to ARToolKit.
[žiūrėta 2019-06-11]. Prieiga per Internetą:
<http://www.hitl.washington.edu/artoolkit/documentation/userintro.htm>

[Was19b] ARToolKit.
[žiūrėta 2019-06-11]. Prieiga per Internetą:
<http://www.hitl.washington.edu/artoolkit/>

[Wik18] Wikipedia. Aircraft principal axes.
[žiūrėta 2018-05-25]. Prieiga per Internetą:
<https://en.wikipedia.org/wiki/Aircraft_principal_axes>

[WMZ12] K. E. Wenzel, A. Masselli ir A. Zell. Visual tracking and following of a quadrocopter by another quadrocopter. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 4993–4998, 2012.

[YSZ13] Shaowu Yang, Sebastian Scherer ir Andreas Zell. An onboard monocular vision system for autonomous takeoff, hovering and landing of a micro aerial vehicle. Journal of Intelligent Robotic Systems, 69:499–515, 2013.

Priedai

Priedas Nr. 1 Komandos „rosbag info“ išvestis

```
dariusm@ubuntu:~$ rosbag info r4_2020-01-06-18-44-40.bag
path:        r4_2020-01-06-18-44-40.bag
version:     2.0
duration:    26.8s
start:       Jan 01 1970 03:12:19.31 (739.31)
end:         Jan 01 1970 03:12:46.09 (766.09)
size:        630.3 MB
messages:    23454
compression: none [473/473 chunks]
types:       ar_pose/ARMarker              [93c4ce9061a70bc30293e52ac4675f76]
             ardrone_autonomy/Navdata      [e1169f766234363125ac62c9a3f87eeb]
             cvg_sim_msgs/Altimeter        [c785451e2f67a76b902818138e9b53c6]
             geometry_msgs/PointStamped    [c63aecb41bfdfd6b7e1fac37c7cbe7bf]
             geometry_msgs/Twist           [9f195f881246fdfa2798d1d3eebca84a]
             geometry_msgs/Vector3Stamped [7b324c7325e683bf02a9b14b01090ec7]
             sensor_msgs/CameraInfo        [c9a58c1b0b154e0e6da7578cb991d214]
             sensor_msgs/Image             [060021388200f6f0f447d0fcd9c64743]
             sensor_msgs/Imu               [6a62c6daae103f4ff57a132d6f95cec2]
             sensor_msgs/Range             [c005c34273dc426c67a020a87bc24148]
             tf2_msgs/TFMessage            [94810edda583a504dfda3829e70d7eec]
topics:      /altimeter                      266 msgs    : cvg_sim_msgs/Altimeter
             /ar_pose_marker                 254 msgs    : ar_pose/ARMarker
             /ardrone/bottom/camera_info     946 msgs    : sensor_msgs/CameraInfo
             /ardrone/bottom/image_raw       946 msgs    : sensor_msgs/Image
             /ardrone/imu                   2670 msgs    : sensor_msgs/Imu
             /ardrone/navdata               5326 msgs    : ardrone_autonomy/Navdata
             /cmd_vel                        162 msgs    : geometry_msgs/Twist
             /magnetic                       266 msgs    :
geometry_msgs/Vector3Stamped
             /navdata                       5358 msgs    : ardrone_autonomy/Navdata
             /pressure_height                266 msgs    : geometry_msgs/PointStamped
             /sonar_height                   266 msgs    : sensor_msgs/Range
             /tf                            6728 msgs    : tf2_msgs/TFMessage
```

Priedas Nr. 2 Paleidimo konfigūracijos failo pavyzdys

```
<?xml version="1.0"?>
<launch>
  <arg name="drone_ip" default="127.0.0.1"/>

  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find ar_pose)/launch/live_single.rviz"/>

  <node pkg="tf" type="static_transform_publisher" name="world_to_cam"
    args="0 0 0.5 -1.57 0 -1.57 world camera 10" />

  <node name="drone_gui" pkg="uga_tum_ardrone" type="drone_gui">
    <param name="DroneIP" value="$(arg drone_ip)"/>
    <remap from="/ardrone/takeoff" to="/takeoff"/>
    <remap from="/ardrone/land" to="/land"/>
    <remap from="/ardrone/reset" to="/reset"/>
  </node>

  <include file="$(find cvg_sim_gazebo)/launch/empty_world.launch"/>

</launch>
```

Priedas Nr. 3 Supaprastintas nusileidimo algoritmo pseudokodas

```
Sukuriame ROS mazgą "ard_autopilot"
Sukuriame TF tranformacijų buferį
Inicijuojame TF transformacijų prenumeratorių
Inicijuojame drono valdymo komandų publikavimo objektą
Sukuriame markerio radimo su ar_pose temos prenumeratorių
Sukuriame temos skirtos nusileidimo trukmei skaičiuoti publikatorių
Apsibrėžiame drono valdymo komandas kurias naudosime
Nuskaitome x,y,z ašių ir pokrypio paklaidas leidžiantis ant markerio iš parametrų
serverio

Metodas kuris kviečiamas kai mes aptinkame markerį
void MarkerCallback()
{
    Išsaugome kada paskutinį kartą sėkmingai radome markerį

    if (markeris_pamatytas_pirma_karta_sesijoje) {
       Išsisaugome kada suradome pirmąjį markerį
    }
}


Pagrindinis ROS mazgo ciklas
while (ros_mazgo_gyvavimo_salygos_patenkinamos)
{
    try
    {
        Gauname TF tranformaciją tarp drono apatinės kameros ir ar markerio
    }
    catch
    {
        Palaukiame vieną sekundę
        Tesiame programos darbą sekančiame while cikle
    }

    if (pirma_karta_pamatytas_markeris_nera_issaugotas && radome_markeri) {
        Publikuojame į /landing_timeline temą pirmojo markerio aptikimo laiką
    }

    Iš gautos TF trandformacijos tarp drono ir markerio sukuriame kvaterniona
    Iš kvaterniono sukuriame tranformacijos matricą
    Iš tranformacijos matricos gauname posvyrį, polinkį ir pokrypį
    Nustatome drono judesio objeko reikšmes pagal x,y,z ašis bei pokrypį
    Atimame nusileidmo taško paklaidos dydžius

    if (drono atstumas nuo markerio atėmus paklaidas lygus 0)
    {
        Nuleidžiam droną ant markerio
        Publikuojame į /landing_timeline temą drono nuleidimo ant markerio laiką
```

```
        Išjungiame drono autopiloto mazgą
    }
    else
    {
        if (jei markerį pamatėme dabar)
        {
            Judame link markerio
        }
        else
        {
            Pranešame apie pamestą markerį
            Išvalome drono komandų eilę
            Sukuriame komandą kilti 60 cm aukštyn
            Siunčiame dronui komandą kilti aukštyn
        }
    }
}
```

Priedas Nr. 4 Detalesnių nusileidimo algoritmo tyrimų rezultatai

	low rate	mediumm rate	high rate
default agr	45	15	44
optimal agr	24	7	7
medium agr	22	7	26
high agr	19	7	26

starting point coordinates
-2 -2 3 225
Control app version V2

default low	default medium	default high
Params:	Params:	Params:
Default	Default	Default
K_direct = 5	K_direct = 5	K_direct = 5
K_rp = 0.5	K_rp = 0.5	K_rp = 0.5
global rate 1.0	global rate 10.0	global rate 30.0
reach rate 2.0	reach rate 1.0	reach rate 0.1
lost rate 2.0	lost rate 1.0	lost rate 0.1
Results:	Results:	Results:
Detected 956	Detected 1182	Detected 1243
Landed 1001	Landed 1197	Landed 1287
Elapsed 45	Elapsed 15	Elapsed 44

optimal low	optimal medium	optimal high
Params:	Params:	Params:
Default	Default	Default
K_direct = 5	K_direct = 5	K_direct = 5
K_rp = 10	K_rp = 10	K_rp = 10
global rate 1.0	global rate 10.0	global rate 30.0
reach rate 2.0	reach rate 1.0	reach rate 0.1
lost rate 2.0	lost rate 1.0	lost rate 0.1
Results:	Results:	Results:

Detected 981	Detected 964	Detected 994
Landed 1005	Landed 971	Landed 1001
Elapsed 24	Elapsed 7	Elapsed 7

high low	high medium	high high
Params:	Params:	Params:
Default	Default	Default
K_direct = 50	K_direct = 50	K_direct = 50
K_rp = 50	K_rp = 50	K_rp = 50
agr = 5	agr = 5	agr = 5
global rate 1.0	global rate 10.0	global rate 30.0
reach rate 2.0	reach rate 1.0	reach rate 0,1
lost rate 2.0	lost rate 1.0	lost rate 0.1
Results:	Results:	Results:
Detected 1617	Detected 1140	Detected 976
Landed 1636	Landed 1147	Landed 1002
Elapsed 19	Elapsed 7	Elapsed 26

medium high	medium medium
Params:	Params:
Default	Default
K_direct = 35	K_direct = 35
K_rp = 35	K_rp = 35
agr = 1	agr = 1
global rate 30.0	global rate 10.0
reach rate 0,1	reach rate 1.0
lost rate 0.1	lost rate 1.0
Results:	Results:
Detected 976	Detected 1140
Landed 1002	Landed 1147
Elapsed 26	Elapsed 7

Priedas Nr. 5 Pirminių nusileidimo algoritmo tyrimų rezultatai

Refresh rates		Start coords	-1.5 -1 3 225
low rate			
global rate 1.0		Control app version V1	
reach time 2.0			
lost time 2.0			
Results			
default	*medium*	*constrained RP*	*optimal*
K_direct = 5	K_direct = 35	K_direct = 35	K_direct = 5
K_rp = 0.5	K_rp = 25	K_rp = 0.01	K_rp = 10
48s	16s	125+s	8s

Priedas Nr. 6 Nusileidimo algoritmo priklausomybės nuo dažnio tyrimai

Start coords:	-2 -2 2.8 225		Control app version V3	
Refresh rates				
low rate	*medium rate*	*intermediate rate*	*high rate*	*ultra rate*
global rate 1.0	global rate 10.0	global rate 10.0	global rate 30.0	global rate 100.0
reach time 2.0	reach time 1.0	reach time 0.25	reach time 0.1	reach time 0.01
lost time 2.0	lost time 1.0	lost time 0.25	lost time 0.1	lost time 0.01

default	*too agresive Z*	*constrained RP*	*optimal*	*high*	*medium*	*intermediate*	*real*	
K_direct = 5	K_direct = 45	K_direct = 15	K_direct = 5	K_direct = 50	K_direct = 35	K_direct = 15	K_direct = 3	
K_rp = 0.5	K_rp = 15	K_rp = 0.25	K_rp = 10	K_rp = 50	K_rp = 35	K_rp = 15	K_rp = 5	
				agr = 3			max_drop = -0.5	
47	22	276	14	15	12	12	48	*low*
40	6	71	6	9	5	5	6	*high*
38	5	86	5	12	5	5	6	*ultra*

constrained RP	*default*	*real*	*optimal*	*too agresive Z*	*intermediate*	*medium*	*high*	
K_direct = 15	K_direct = 5	K_direct = 3	K_direct = 5	K_direct = 45	K_direct = 15	K_direct = 35	K_direct = 50	
K_rp = 0.25	K_rp = 0.5	K_rp = 5	K_rp = 10	K_rp = 15	K_rp = 15	K_rp = 35	K_rp = 50	
		max_drop = -0.5					agr = 3	
276	47	48	14	22	12	12	15	*low*
71	40	6	6	6	5	5	9	*high*
86	38	6	5	5	5	5	12	*ultra*

Priedas Nr. 7 Nusileidimo algoritmo markerio paieškos tyrimai

Start coords	0 0 2.8 225		Control app version V3	
Destination coords	-3 -3 2.8 225			
Refresh rates				
high rate				
global rate 30.0				
reach time 0.1				
lost time 0.1				
Agresivesness parameters				
default	*real*	*intermediate*	*high*	
K_direct = 5	K_direct = 3	K_direct = 15	K_direct = 50	
K_rp = 0.5	K_rp = 5	K_rp = 15	K_rp = 50	
	max_drop = -0.5		agr = 3	
39	6	5	10	seconds

Priedas Nr. 8 Programinės įrangos sprendimo projekto struktūra