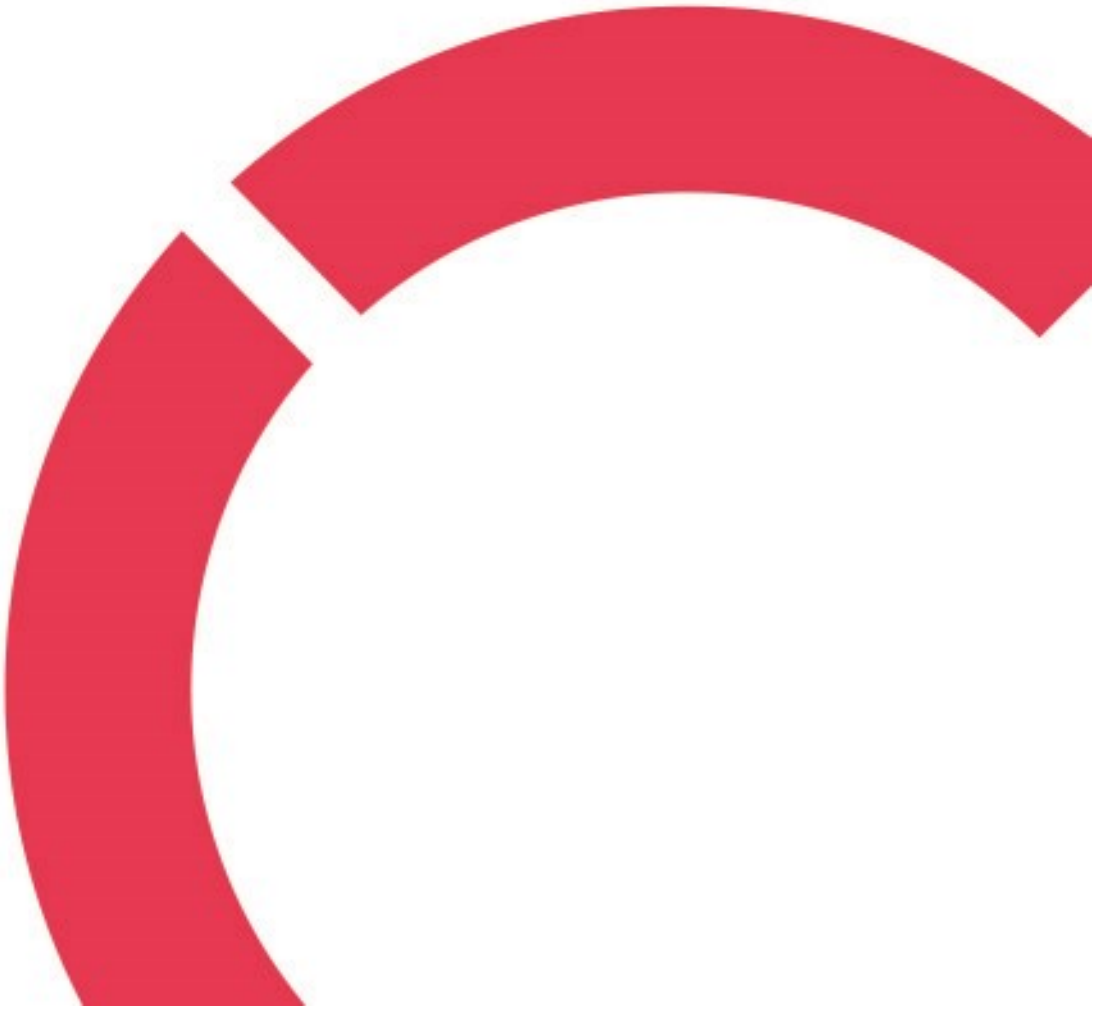
Esa Aaltonen

GPS-POHJAISEN AUTOMAATTIOHJAUKSEN KÄYTTÖÖNOTTO MOBIILIROBOTISSA

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutus
Huhtikuu 2024



TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Huhtikuu 2024	**Tekijä/tekijät** Esa Aaltonen
Koulutus Insinööri (AMK), tieto- ja viestintätekniikka		☒ AMK ☐ YAMK
TYÖN NIMI GPS-POHJAISEN AUTOMAATTIOHJAUKSEN KÄYTTÖÖNOTTO MOBIILIROBOTISSA		
Työn ohjaaja Sari Lipsanen		**Sivumäärä** 47 + 8
Työelämäohjaaja Marjut Koskela		

Opinnäytetyössä tavoitteena oli toteuttaa satelliittipaikannukseen perustuva automaattiohjausjärjestelmä telakäyttöiseen mobiilirobottiin, jonka liikkumista ohjattiin CAN-väylän välityksellä. Kehitystyön pohjaksi valitussa ratkaisussa hyödynnettiin ajoneuvossa olevaa Linux-tietokonetta ja siihen asennettua ROS-järjestelmää sekä Pixhawk-autopilottiohjainta ja ArduRover-laiteohjelmistoa.

Työn tilaajana oli Centria-ammattikorkeakoulun Dronet ja autonomiset laitteet -tutkimusryhmä, joka hyödyntää mobiilirobottia mm. maastomittauksissa ja muissa TKI-toiminnoissa.

ROS-pohjaisen sovelluksen kehittämiseen käytettiin pääosin Python-ohjelmointikieltä sekä MAV-ROS-ohjelmistopakettia. CAN-väylän ja ROS-järjestelmän välistä tietoliikennettä varten tehtiin muutoksia myös mobiilirobotin valmistajan toimittaman ohjelmiston C++-lähdekoodiin.

Lopputuloksena on autopilottisovellus, jonka avulla laitteella voi ajaa automaattisesti satelliittipaikannukseen perustuvia ajoreittejä. Käyttöliittymänä hyödynnetään laitteen omaa radio-ohjainta, ja Linux-järjestelmän komentorivin kautta voidaan käyttää ROS-palvelulla toteutettuja reitinmuokkauksen lisätoimintoja.

Asiasanat
ArduRover, autopilotti, GPS-paikannus, MAVLink, MAVROS, mobiilirobotti, Pixhawk, ROS

ABSTRACT

Centria University of Applied Sciences	**Date** April 2024	**Author** Esa Aaltonen
Degree programme Bachelor of Engineering, Information and Communications Technology		
Name of thesis DEPLOYMENT OF A GPS-BASED AUTOMATIC CONTROL SYSTEM FOR AN UNMANNED GROUND VEHICLE		
Centria supervisor Sari Lipsanen		**Pages** 47 + 8
Instructor representing commissioning institution or company Marjut Koskela		

The purpose of this thesis was to implement an automatic control system for a tracked unmanned ground vehicle (UGV) manoeuvred through a CAN bus. The development work was based on a solution utilising the vehicle's on-board Linux computer with Robot Operating System (ROS) as well as a Pixhawk flight control unit (FCU) with ArduRover firmware.

The project was commissioned by Centria's Drones and Autonomous Device research group. The UGV is used by the research group in various RDI activities, such as terrain measurements.

The ROS-based solution was mainly developed using the Python programming language and the MAVROS software package. The C++ source code of the software supplied by the UGV manufacturer was also modified to accommodate necessary data communication between the CAN bus and the ROS system.

The resulting autopilot solution enables the vehicle to perform automatic routes ("missions") making use of satellite positioning. The vehicle's original radio controller was used to provide a user interface, and the auxiliary functions for manipulating the mission through a ROS service can be executed on the Linux command line interface.

Key words
ArduRover, autopilot, FCU, GPS, MAVLink, MAVROS, Pixhawk, ROS, UGV

KÄSITTEIDEN MÄÄRITTELY

Aktivointitila (arming)
ArduPilot-autopilottiohjelmiston toimintatila, jolla sallitaan tai estetään alusta tai ajoneuvoa liikuttavien moottorien toiminta.

Autopilotti
Laite tai järjestelmä, joka huolehtii automaattisesti ajoneuvon ohjaamisesta määritellyllä reitillä. Tässä työssä termillä autopilotti tarkoitetaan autopilottiohjaimen ja sen laiteohjelmiston kokonaisuutta.

Autopilottiohjain (flight controller unit (FCU))
Autopilottitoimintoa suorittava fyysinen laite, joka sisältää autopilottiohjelmiston suorittamiseen tarvittavat laskentaresurssit, erilaisia liike- ym. antureita sekä tarvittavat oheislaiteliitännät.

Controller Area Network (CAN)
Yleinen ajoneuvoissa, koneissa ja teollisuuslaitteissa käytetty automaatioväylä.

Global Positioning System (GPS)
Maailmanlaajuiseen käyttöön vakiintunut satelliittipaikannusjärjestelmä.

Mobiilirobotti (unmanned ground vehicle (UGV))
Miehittämätön kauko-ohjattava tai automatisoitu maa-ajoneuvo.

Ohjaustila (control mode)
ArduPilot-autopilottiohjelmiston toimintatila, joka määrittää aluksen tai ajoneuvon käyttö- ja ohjaustavan. Esim. käsiohjaus (MANUAL) tai automaattiohjaus (AUTO).

Robot Operating System (ROS)
Robottijärjestelmien kehitystyön helpottamiseksi kehitetty useita eri käyttöjärjestelmiä tukeva väliohjelmisto.

KUVIOT

KUVAT

TAULUKOT

KOODIT

1 JOHDANTO

Tämän opinnäytetyön aiheena on satelliittipaikannukseen perustuvan automaattiohjauksen toteuttaminen Controller Area Network -väylän (CAN) välityksellä ohjattavaan mobiilirobottiin eli miehittämättömään maa-ajoneuvoon. Työn toimeksiantajana on Centria-ammattikorkeakoulun Dronet ja autonomiset laitteet -tutkimusryhmä (”Centria Drone Lab”), joka tarjoaa alueen yrityksille käyttösovellusten demonstrointia ja kehittelyä maassa, ilmassa ja vedessä liikkuvien kauko-ohjattavien tai autonomisesti toimivien laitteiden avulla. Käyttötapauksia ovat mm. maastomittaukset, kuvantaminen lidarilla tai lämpökameralla, kunnossapito-, valvonta- ja tarkastustehtävät sekä ihmiselle vaarallisissa kohteissa suoritettavat tehtävät etäohjauksella.

Opinnäytetyössä on tarkoituksena kehitellä ja ottaa käyttöön toiminnallisuus, jolla Centria Drone Labin AgileX Bunker -mallista mobiilirobottia eli UGV-laitetta (unmanned ground vehicle (Gage 1995, 1)) voidaan ohjata satelliittipaikannuksen avulla ja suorittaa laitteella automaattisia, identtisesti toistettavia ajotehtäviä. AgileX Bunker on teloilla liikkuva mobiilirobotti, jossa on manuaalinen radiokaukoohjaus sekä pohjaosan päällä olevaan kiinnityskehikkoon asennettu Linux-tietokone oheislaitteineen. Tietokone on yhdistetty roiskesuojatussa pohjaosassa olevaan ohjausyksikköön CAN-väylän välityksellä. Työn teoriaosuudessa esitellään mobiilirobotin ja sen oheislaitteiden teknisiä ominaisuuksia sekä perehdytään Robot Operating Systemin (ROS) rakenteeseen ja Pixhawk-autopilottiohjaimen, ArduPilot-laiteohjelmiston ja MAVROS-ohjelmistopaketin tarjoamiin ohjausmahdollisuuksiin. Toiminnallinen osuus koostuu pääasiassa ROS-järjestelmässä toimivan Python-ohjelmiston kehittämisestä, olemassa olevan C++-ohjelmiston muokkaamisesta sekä järjestelmän testaamisesta ja havaintojen dokumentoinnista. Tavoiteltuna lopputuloksena on laitteiston omia resursseja hyödyntävä helppokäyttöinen GPS-pohjainen automaattiohjaustoiminto. Lisäksi työn yhteydessä tutkitaan valitun ratkaisumallin tarjoamia laajempia hallinta- ja jatkokehitysmahdollisuuksia.

Tehtävänanto oli alun perin varsin väljä, joten olen ottanut minimitavoitteeksi sen, että automaattisen ajoreitin pystyisi asettamaan ja käynnistämään kannettavassa tietokoneessa käytettävällä maa-asemasovelluksella sekä radio-ohjaimen avulla ja että reittipisteitä voisi asettaa radio-ohjaimella manuaalisesti ajon aikana kulloisenkin paikannussijainnin mukaan. Laajemmin ajatuksena on myös kokeilla, mitä sellaisia ohjelmallisia reittitoimintoja sovelluksen oheen voisi kehitellä, joihin autopilottiohjaimeen yhdistetyssä maa-asemasovelluksessa ei ole valmista toteutusta. Työssä keskitytään toteutuksen kannalta oleellisimpiin ArduPilot-ohjelmiston ja muiden järjestelmien ominaisuuksiin sekä

tarkastellaan pääpiirteittäin sellaisia lisäominaisuuksia, joihin liittyy ilmeisiä jatkokehitysmahdollisuuksia. Kehittyneemmät autonomisen liikkumisen toiminnot, kuten esteiden väistäminen lidarin tai kameran tuottaman datan avulla, eivät kuulu tämän työn laajuuteen, mutta näitä mahdollisuuksia sivutaan pintapuolisesti. Satelliittipaikannuksen ja muun paikkatiedon tarkkuutta ja luotettavuutta tarkastellaan käytännön havaintojen kautta ja sovelluksen käytettävyyden ehdoilla.

Tärkeimpinä teoriasisältöä ja käytännön toteutusta tukevina lähteinä ovat ROS-järjestelmää koskeva kirjallisuus (Pyo, Cho, Jung & Lim 2017) ja ROSin dokumentaatio (ROS 2018a–b ja ROS 2023a–b sivustossa http://wiki.ros.org), ArduPilotin dokumentaatio (ArduPilot 2024a–x sivustossa https://ardupilot.org) sekä satelliittipaikannusta käsittelevä kirjallisuus, varsinkin Poutanen (2016).

2 AGILEX BUNKER

AgileX Bunker on AgileX Roboticsin valmistama ja markkinoima tela-alustainen kauko-ohjattava miehittämätön maa-ajoneuvo eli mobiilirobotti, ns. UGV-laite (AgileX Robotics 2024). Laitteen tekniset tiedot on esitetty taulukossa 1.

TAULUKKO 1. AgileX Bunkerin tekniset tiedot (mukaillen AgileX Robotics 2024)

Mitat	pituus 1 023, leveys 778, korkeus (pohjaosa) 400 mm
Akseliväli	360 mm
Paino	145–150 kg
Maavara	90 mm
Hyötykuorma	70 kg
Suurin nousukulma	36°, voi nousta portaita
Kääntösäde	0 m
Käyttölämpötila	–10 ... +45 °C
Ohjaustapa	differentiaaliohjaus teloilla (kumiset telamatot)
Akku	48 V, 60 Ah (kapasiteetiltaan perusmallia suurempi akku)
Moottorit	harjattomat, 2 x 650 W
Enkooderi	1 024 pulssia/pyörähdys
Tiedonsiirto	CAN-väylä
Suojausluokka	IP52

2.1 Laitteistokokoonpano

Laitteen kehittelykokoonpanossa (KUVA 1) pohjaosan päällä oli alumiininen kiinnityskehikko, johon oli asennettu tietokone, langaton lähiverkkoreititin, USB-jakaja (”hubi”), näyttö, lidar (valotutka) sekä 12 voltin jännitemuunnin oheislaitteiden virransyöttöä varten. Kiinnityskehikko on irrotettava, ja sen tilalle voi tarpeen mukaan sijoittaa muuta hyötykuormaa.

KUVA 1. Centria Drone Labin AgileX Bunkerin laitteistokokoonpano

Tietokoneessa on Intel Core i7-9700 -suoritin (8 ydintä, kellotaajuus 3,00 GHz) ja 16 Gt:n keskusmuisti, ja käyttöjärjestelmä on 64-bittinen Ubuntu 16.04 LTS (Xenial Xerus). Opinnäytetyön aloitushetkellä koneessa oli valmiiksi asennettuna ROS-väliohjelmisto (jakeluversio Kinetic Kame) ja AgileX Roboticsin toimittama ROS-pohjainen hallintaohjelmisto sekä NoMachine-etätyöpöytäsovellus, jolla koneen työpöytäympäristöä voi käyttää samassa langattomassa lähiverkossa olevalta toiselta tietokoneelta.

Lisäksi kiinnityskehikon päälle on asennettu lidar eli valotutka. Lidaria ei hyödynnetty tämän työn yhteydessä, mutta sen tuottamia mittaustietoja olisi mahdollista käyttää apuna esimerkiksi esteiden tunnistamisessa ja reitin hallinnassa ja kartoitustehtävissä.

2.2 Ohjauslaitteet ja tietoliikenne

Bunkeria voi ohjata manuaalisesti radio-ohjaimella tai CAN-väylään syötettävien ohjausviestien välityksellä. Ohjaustapa valitaan radio-ohjaimen kolmiasentoisella vipukytkimellä SWB (ks. KUVA 3). Kun vipu on keskiasennossa, laitteen liikettä ohjataan suoraan radio-ohjaimen sauvoilla. Kun vasenta ohjaussauvaa liikutetaan ylös- tai alaspäin, laite liikkuu eteen- tai taaksepäin. Oikealla ohjaussauvalla säädetään käytännössä vasemman ja oikean telan välistä nopeuseroa, eli kun oikeaa ohjaussauvaa liikutetaan vasemmalle tai oikealle, laite kääntyy vastaavasti. Jos laitetta ei käännöksen aikana liikuteta samanaikaisesti eteen- tai taaksepäin, laite kääntyy paikallaan siten, että telat pyörivät eri suuntiin. Kun vipukytkin SWB on yläasennossa, ohjausyksikkö lukee ohjauskomentoja CAN-väylästä ja käyttää sähkömoottoreita komentojen mukaisesti. Tällöin kauko-ohjaimen ohjaussauvoilla annetut komennot eivät vaikuta laitteen liikkeeseen, mutta laitteisto välittää kauko-ohjaimen syötetiedot CAN-väylään.

3 NAVIGOINTI- JA PAIKKATIETOJÄRJESTELMÄN TOIMINTAPERIAATE

Navigoinnin eli aluksen tai ajoneuvon paikanmääritykseen perustuvan ohjaamisen tarkoituksena on saada ajoneuvo siirtymään tiettyyn, usein kartalta osoitettuun määränpäähän. Ohjaustoimenpiteillä säädellään ajoneuvon nopeutta ja suuntaa. Tätä varten tarvitaan tiedot ajoneuvon kulloisestakin sijainnista ja asennosta suhteessa kartan koordinaatteihin. Koska sijaintia ja – kuten tämän työn käytännön toteutuksen yhteydessä kävi havainnollisesti ilmi – myös suuntaa koskeviin tietoihin liittyy epätarkkuutta, laadukkaiden navigointipäätösten tueksi tarvitaan lisäksi reaaliaikaisia kinemaattisia mittaustietoja.

3.1 GNSS ja GPS

Satelliittipaikannusjärjestelmä (Global Navigation Satellite System (GNSS)) perustuu Maata kiertäviin navigaatiosatelliitteihin, jotka lähettävät tarkkoja, synkronoituja kellonaikaviestejä. Vastaanotin vertaa eri satelliiteista saapuvia kellonaikaviestejä toisiinsa sekä referenssikellonaikaan ja laskee viestien viiveiden eli signaalien kulkuaikojen perusteella satelliittien laskennalliset etäisyydet havaintopisteestä ja oman sijaintinsa Maan pinnalla. Etäisyyksien laskentaan liittyy useita virhelähteitä, jotka on eliminoitava käyttämällä useampaa satelliittia; sijainnin tarkkaa määrittämistä varten tarvitaan kellonaikaviestejä vähintään neljästä satelliitista (Poutanen 2016, 12). Satelliittipaikannusjärjestelmiä on kehitetty useita, mutta Yhdysvaltain puolustusministeriön kehittämä ja ylläpitämä Global Positioning System (GPS) on eräs tunnetuimmista ja myös kuluttajasovelluksissa hyvin yleinen järjestelmä. Järjestelmän kehittäminen aloitettiin 1970-luvulla, ja se otettiin täysimittaisesti käyttöön 1990-luvulla. (Poutanen 2016, 11, 19.) Arkikielessä termiä GPS käytetään toisinaan myös satelliittipaikannuksen synonyyminä (Maanmittauslaitos 2024b).

Kuluttajasovelluksissa käytetyt GPS-vastaanottimet hyödyntävät satelliittien lähettämien kellonaikaviestien sisältöä eli ns. koodipaikannusta. Niiden paikannustarkkuus maanpinnan tasossa on muutaman metrin luokkaa. Kalliimmat, signaalin kantoaallon vaihetietoa hyödyntävät vastaanottimet pystyvät määrittämään sijainnin alle metrin tarkkuudella, ja vielä tarkempia tuloksia on mahdollista saada tukiaseman tai tukiasemaverkon avulla. (Poutanen 2016, 16, 255–257.)

3.2 Kompassianturi

Sijaintitiedon lisäksi navigointia varten tarvitaan tieto ajoneuvon asennosta suhteessa kartan koordinaatistoon. Tähän voidaan käyttää magneettikentän muutoksiin reagoivaa kompassianturia. Käyttöönoton yhteydessä kompassianturi on kalibroitava liikuttelemalla sitä eri asentoihin, jotta voidaan määrittää, missä suunnassa magneettikenttä on voimakkaimmillaan. Näin saadaan tieto magneettisen pohjoisnavan suunnasta. Anturilaitteiston läheisyydessä olevat sähkölaitteet ja metalliesineet vaikuttavat mittaustulokseen, ja tästä aiheutuva virhe suunnan määrittämisessä on pyrittävä eliminoimaan (ArduPilot 2024n). Sen vuoksi kompassin kalibrointi pitäisi tehdä anturin ollessa asennettuna ajoneuvon lopulliseen käyttökokoonpanoon (ArduPilot 2024f).

3.3 Kiihtyvyysanturit

GPS-vastaanottimen ja kompassin ohella kolmantena paikkatiedon tuottamista tukevana komponenttityyppinä on kiihtyvyysanturi. Kiihtyvyysanturien avulla saadaan tietoa ajoneuvon asennosta suhteessa gravitaatiokiihtyvyyden suuntaan sekä hetkellisistä nopeuden muutoksista (ks. kohta 5.7.5). Kiihtyvyysantureilla on tärkeä rooli ajoneuvon automaattiohjauksessa, sillä niiden tuottamien reaaliaikaisten mittaustietojen avulla voidaan reagoida nopeasti ajoneuvon äkillisiin heilahduksiin tai pyörähdyksiin sekä täydentää satelliittipaikannuksen ja kompassianturin tuottamia tietoja. Parhaan tuloksen saamiseksi myös kiihtyvyysanturit on kalibroitava käyttökokoonpanossa (ArduPilot 2024a).

4 GPS-POHJAISEN AUTOMAATTIOHJAUKSEN RATKAISUMALLI

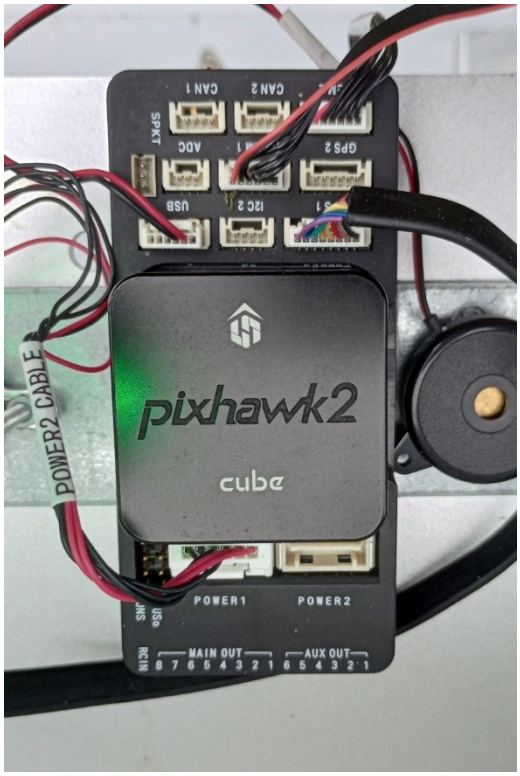
Aluksi oli valittava ratkaisumalli, jolla GPS-pohjainen automaattiajo voitaisiin toteuttaa. Erillisiä GPS-moduuleja, kompassiantureita ym. komponentteja on saatavilla runsaasti erilaisia. Eräs mahdollinen ratkaisu voisi perustua itsenäisesti kehitettyyn ohjelmaan, joka lukee sijainti- ja asentotietoja anturikomponenteilta ja laskee tietojen perusteella tarvittavan liikesyötteen kohti haluttua reittipistettä. Tällä menetelmällä kaikkien haluttujen toimintojen toteuttaminen ja laitteen luotettavan toiminnan aikaansaaminen olisi kuitenkin hyvin työlästä mm. sen vuoksi, että anturien tuottaman raakadatan saattaminen käyttökelpoiseen muotoon edellyttää tietojen suodattamista laskennallisesti. Lisäksi moottorien nopeuden ja ohjauksen säätö olisi toteuttava niin, että ajoneuvo liikkuu sujuvasti ilman mekaanisia komponentteja kuormittavaa nykimistä, ohjauksen heiluriliikettä tai muuta häiriötä. Tällä menetelmällä opinnäytetyön laajuus ei välttämättä riittäisi tarvittavien toimintojen kehittelyyn ja testaamiseen, ja lopputulos jäisi ominaisuuksiltaan väistämättä melko alkeelliseksi. Käyttötarkoitukseen on kuitenkin saatavilla valmiita ohjelmistoja, kuten avoimeen lähdekoodiin perustuva ArduPilot ja sen maa-ajoneuvoihin tarkoitettu versio ArduRover. Eri valmistajilta on saatavilla erilaisia autopilottiohjaimia, joihin on integroitu antureita, liitäntäominaisuuksia sekä tarvittavat prosessointiresurssit ArduPilotin kaltaisen laiteohjelmiston ajamista varten. Usein tällaiseen laitteeseen liitetään erillinen GPS-vastaanottimen ja kompassin sisältävä moduuli. Kuten testausvaiheessa kävi ilmi, autopilotin tekemä ajonaikainen virheenkorjaus ja monipuoliset säätöparametrit ovat tärkeitä ominaisuuksia, jotka parantavat ratkaisun käyttökelpoisuutta merkittävästi.

Automaattiohjausratkaisua ryhdyttiin kehittämään Pixhawk-autopilottiohjaimen pohjalta, ja tällainen oli Centrian kalustossa käytettävissä. Samantyyppisiä Pixhawk-laitteita käytetään varsinkin multikoptereissa moottorien ohjaukseen ja liikkeen vakauttamiseen manuaalisessa ohjauksessa sekä automaattisten reittien lentämiseen. Laite soveltuu monenlaisiin muihinkin liikkuviin laitteisiin, kuten maa-ajoneuvoihin, veneisiin ja sukellusveneisiin (ArduPilot 2024x). Ominaisuuksiltaan samankaltaisia laitteita on saatavilla monilta eri valmistajilta.

4.1 Pixhawk-autopilottiohjain ja siihen liittyvä laitteisto

Pixhawk 2 Cube (KUVA 2) koostuu varsinaisesta ohjainmoduulista (”Cube”) sekä alustamoduulista, jossa on liittimet mm. radiovastaanottimelle, moottorinohjaimille, servomoottoreille, GPS-moduulille

ja monille muille lisälaitteille. Cube-moduulissa on mm. Cortex-M4F-ytimellä varustettu 32-bittinen STM32F427-prosessori (kellotaajuus 168 MHz), 32-bittinen STM32F103-apuprosessori sekä kaksi tärinäeristettyä ja lämpötilavakioitua inertiamittausyksikköä (inertia measurement unit, IMU) ja yksi kiinteästi asennettu referenssiyksikkö. Inertiamittausyksikkö sisältää lineaarista ja kulmakiihtyvyyttä mittaavia kiihtyvyys- ja gyroskooppiantureita sekä sisäisen kompassianturin. (PX4 Dev Team 2020.)



KUVA 2. Pixhawk 2 Cube -autopilottiohjain

Pixhawk 2 Cubeen syötettiin virta AgileX Bunkerissa olevan 12 V:n jännitteensäätimen ja erillisen, vakioitua 5 V:n tasavirtaa tuottavan virtalähdemoduulin kautta. Laitteeseen oli kytketty summeri ääni-merkkejä varten, 433 MHz:n telemetriaradio sekä Here+-GPS-moduuli, jossa on sisäänrakennettu kompassianturi. Tietoliikenne Pixhawkin ja Bunkeriin asennetun tietokoneen välillä toteutettiin sarja-liikennemoduulilla, joka oli kytketty Pixhawkin sarjaliitäntään ja tietokoneen USB-porttiin. Vaihtoeh-toisesti kytkentään voisi käyttää Pixhawkin USB-liitäntää, ja testausvaiheessa tämä vaihtoehto todet-tiin toimivaksi. Myös virransyöttö olisi mahdollista ottaa tietokoneen USB-väylästä, mutta tällöin on varmistettava, että vakaa virransyöttö varmasti riittää autopilottiohjaimen ja siihen liitetyn GPS-mo-duulin häiriintymätöntä toimintaa varten.

4.2 Autopilottiohjelmisto

Yleisesti saatavilla olevia, avoimeen lähdekoodiin perustuvia autopilottiohjaimiin sopivia laiteohjelmistoja on tarjolla useita. Tässä opinnäytetyössä käytetty Pixhawk-laite tukee ArduPilot- ja PX4-laiteohjelmistoja, ja monissa malliversioissa PX4 on esiasennettuna. Molemmat ohjelmistot soveltuvat sekä ilma-aluksiin että maa-ajoneuvoihin. PX4 on julkaistu BSD-lisenssillä (laiteohjelmiston lähdekoodiin tehtyjä muutoksia ei tarvitse julkaista) ja ArduPilot GPL-lisenssillä (koodiin tehdyt muutokset on julkaistava), joten PX4 saattaa soveltua paremmin yritysten omaan kehitystyöhön, jos laiteohjelmistoa on tarve muokata ja yritys haluaa hyödyntää koodia vapaammin kaupallisesti. (Drone Dojo 2022.) ArduPilot on harrastajien keskuudessa suosittu, sillä se on ominaisuuksiltaan monipuolinen ja helppokäyttöinen ja projektissa on kattava dokumentaatio (ArduPilot 2024b). Tässä työssä käytettiin ArduPilotin maa-ajoneuvoihin tarkoitettua alaversiota ArduRover.

4.3 Maa-asemasovellus

Autopilotin tilan seuraamista sekä ajoreitin suunnittelua ja hallintaa varten tarvitaan yleensä tietokoneessa toimivaa maa-asemasovellusta. ArduPilot-laiteohjelmiston yhteydessä käytetään usein erityisesti sitä varten kehitettyä Mission Planner -sovellusta (ArduPilot 2024q). PX4:n yhteydessä tyypillinen maa-asemasovellus on QGroundControl, joka tukee myös ArduPilotia (QGroundControl 2023). Muitakin sovelluksia on saatavilla, mutta tässä työssä käytettiin Mission Planneria, joka osoittautui varsin tarkoituksenmukaiseksi. Reittisuunnittelun ja laitteen reaaliaikaisen seuraamisen lisäksi Mission Plannerilla voi asentaa ArduPilot-ohjelmiston laitteen Flash-muistiin sekä hallita laitteen microSD-kortilla olevia tiedostoja, esim. skriptejä.

Maa-asemasovellus voidaan yhdistää autopilottiohjaimeen USB-kaapelilla, jolloin tietoja voidaan siirtää ennen ajoneuvon käyttämistä, tai ns. telemetriaradioilla, joista toinen on yhdistetty autopilottiohjaimeen ja toinen maa-asemana toimivaan tietokoneeseen. Telemetriaradiot välittävät tietoja reaaliaikaisesti niin, että ajoneuvon sijaintia, nopeutta ym. tietoja voidaan seurata jatkuvasti maa-asemasovelluksen kautta. Reittitietoja ja muita komentoja voidaan lähettää laitteeseen telemetriaradion kautta, joten USB-kaapeliyhteyttä ei tarvita laiteohjelmiston asentamisen jälkeen. Telemetriayhteys voidaan toteuttaa myös Bluetooth-yhteydellä siten, että autopilottiohjaimeen kytketään tarvittaessa Bluetooth-moduuli (ArduPilot 2024e). Tämän opinnäytetyöprojektin yhteydessä käytettiin 433 MHz:n SiK-telemetriaradioita.

4.4 MAVLink, ROS ja MAVROS

Robot Operating System (ROS) on avoimeen lähdekoodiin perustuva ns. metakäyttöjärjestelmä; se on isäntäkoneen käyttöjärjestelmän (Linux, Windows, macOS jne.) palveluja käyttävä ja sovelluksessa ajettavia prosesseja ohjaava väliohjelmisto, joka hoitaa mm. järjestelmän ajoituksen, prosessien käynnistämisen ja lopettamisen, prosessien välisen tiedonvälityksen ja paketinhallinnan (ROS 2018b).

ROSin hajautettu rakenne perustuu isäntäprosessiin (*master*) ja itsenäisesti toimiviin ajettaviin prosesseihin eli solmuihin (*node*), jotka tuottavat ja vaihtavat tietoja järjestelmän muiden osien kanssa. ROS-ohjelmiston perusyksikkönä on paketti (*package*), johon solmut, viesti- ja palvelutyyppien määritelmät ym. kootaan ja jonka kautta solmujen hallinta hoidetaan keskitetysti. Solmut välittävät keskenään tietoa tyypillisesti määrämuotoisten asynkronisten viestien (*message*) avulla. Pakettikirjastoissa on saatavilla erilaisia vakioviestityyppejä, kuten diagnostiikka-, geometria- ja anturitietoja välittäviä viestityyppejä, ja paketin kehittäjä voi lisätä tarpeen mukaan omia pakettikohtaisia viestityyppejä. Viestit kuuluvat hierarkkisessa järjestelmässä tiettyyn aiheluokkaan (*topic*, jäljempänä ”kanava”), ja tietty solmu voi toimia kanavalle lähetettävien viestien julkaisijana (*publisher*) tai vastaanottajana (*subscriber*) tai molempina. Jos tarvitaan kaksisuuntaista synkronista viestintää, tätä varten voidaan perustaa palvelu (*service*). Palvelua kutsutaan lähettämällä palvelun viestimääritelmän mukainen määrämuotoinen kutsuviesti, joka sisältää palvelun tarvitsemat syötetiedot, ja palvelu lähettää vastauksena saman viestimääritelmän mukaisen vastausviestin. Palvelua kutsuva prosessi odottaa vastausviestin saapumista ennen oman toimintansa jatkamista. Jos kutsuttava prosessi on luonteeltaan pitkäkestoinen, kolmantena viestintävaihtoehtona on toiminto (*action*), joka lähettää asiakasprosessille toiminnon tilan ilmaisevia palauteviestejä. Asiakasprosessi reagoi viesteihin niiden saapuessa mutta jatkaa muuten toimintaansa normaalisti. Lisäksi ROS-järjestelmässä voidaan välittää parametreja eli eräänlaisia järjestelmän laajuisia muuttujia, joiden arvoja eri prosessit voivat lukea tai kirjoittaa ilman erityistä julkaisijaroolia. (Pyo, Cho, Jung & Lim 2017, 42–45.)

ROSissa on komennot osaprosessien hallintaan, esim. *rosrun [paketin nimi] [ajettavan tiedoston nimi]* solmujen käynnistämiseen ja *rosservice call [palvelun nimi] [parametrit]* palvelun kutsumiseen komentoriviltä. Nämä helpottavat järjestelmän tilan seuraamista, sovelluksen kehittämistä ja ongelmien diagnosoimista. ROS-kehitysympäristössä on laaja tuki varsinkin Pythonille ja C++:lle.

ROS-versioita on julkaistu monien Linux-jakelujen tavoin tietyn teeman mukaan nimettyinä jakeluversioina. Kukin Ubuntun jakeluversio tukee yleensä tiettyä ROS-jakeluversiota. Ubuntu 16.04 LTS

Xenial Xerus, joka oli Bunkerin tietokoneessa asennettuna tämän opinnäytetyöprojektin aloitushetkellä, tukee toukokuussa 2016 julkaistua ROS-jakeluversiota Kinetic Kame (ROS 2023a). Tällä hetkellä tuettu ROS-versio ja samalla viimeinen Open Roboticsin julkaisema virallinen ROS 1 -versio on Noetic Ninjemys, jota tukevat Ubuntu-versiot 20.04 LTS Focal Fossasta alkaen. Jatkossa ROSin kehitystyö keskittyy ROS 2:een, jota on vuodesta 2018 julkaistu rinnakkain alkuperäisen ROSin eli ROS 1:n kanssa. (ROS 2023a.) ROS 1:ssä ja ROS 2:ssa on eräitä rakenteellisia eroja, eivätkä ne ole suoraan yhteensopivia. Tässä opinnäytetyössä keskityttiin ROS-jakeluversioon Kinetic Kame, ja ohjelmat kirjoitettiin Pythonilla. Tiettyjen toimintojen toteuttaminen edellytti lisäksi muutoksia AgileX Roboticsin julkaisemiin C++-koodeihin.

Tässä sovelluksessa ROSia käytetään Bunkerin liikekomentojen antamiseen laitteeseen valmiiksi asennettujen ROS-ohjelmien avulla, radio-ohjaimen syötteen lukemiseen (ohjelmiston muunnetun version avulla) sekä mobiilirobotin tietokoneen ja autopilotin väliseen viestintään. Sovellusta voisi laajentaa lisäämällä uusia komponentteja ja moduuleja, esim. antureita tai toimilaitteita, jotka välittävät viestejä saman ROS-järjestelmän kautta. Samassa järjestelmässä toimivia, saman isäntäprosessin alaisia ROS-solmuja voidaan ajaa lähiverkon kautta eri tietokoneilta, ja yksinkertaisia anturi- ja toimilaitesolmuja voidaan suorittaa myös mikrokontrollereissa.

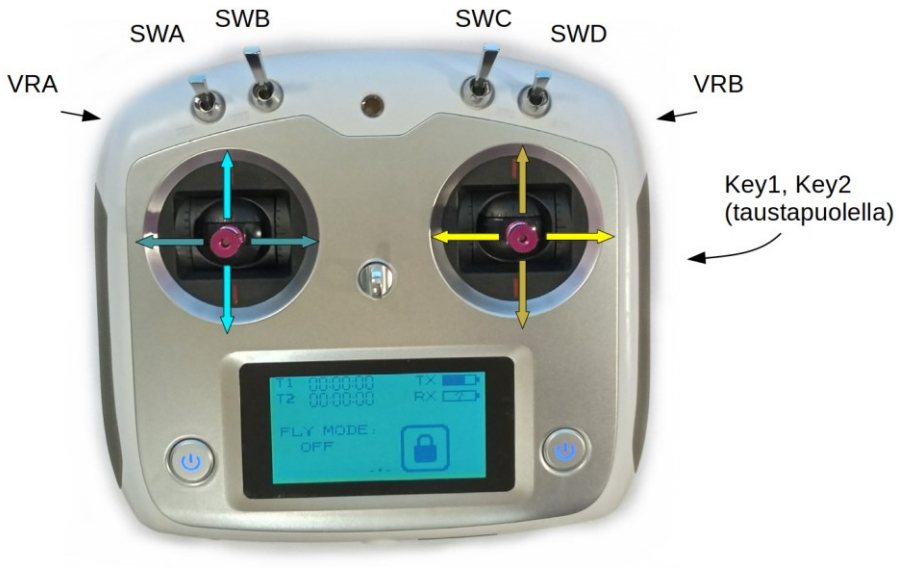
Micro Air Vehicle Link -protokollaa (MAVLink) käytetään varsinkin maa-asemasovelluksen ja miehittämättömän ajoneuvon tai ilma-aluksen eli droonin välisessä tietoliikenteessä. Protokollan kautta välitetään esim. paikannustietoja, ajoneuvon anturidataa, komentoja ym. (ArduPilot 2024o.) ArduPilot-ohjelmisto käyttää MAVLinkiä maa-asemasovelluksen kanssa viestimiseen, ja tätä protokollaa voitaisiin käyttää myös mobiilirobotissa olevan tietokoneen ja autopilotin väliseen viestimiseen. ROS-järjestelmää hyödyntävän oheistietokoneen ja autopilotin välisen viestinnän helpottamiseksi on kehitetty MAVROS-paketti, joka nimensä mukaisesti toimii MAVLinkin ja ROSin välisenä yhdysväylänä (ROS 2018a). MAVROS välittää ROS-järjestelmään jatkuvasti erilaisia autopilotin toimintaan liittyviä tietoja kymmenillä eri kanavilla, esim. ajantasaisia paikannustietoja, anturien mittaustietoja, laskennallisia nopeus- ja navigointitietoja ja automaattisen ajoreitin tietoja. Autopilotin toimintaa voidaan ohjata erilaisilla MAVROS-palveluilla.

4.5 Tietoliikenne Bunkerin ohjausyksikön kanssa

Autopilottiohjainta käytetään tyypillisesti ohjaamaan ajoneuvon sähkömoottorien elektronisille nopeudensäätimille (electronic speed controller (ESC)), ohjausservoille ym. lähteviä signaaleja suoraan laitteessa olevien pulssinleveysmodulaatiota hyödyntävien signaalilähtöjen (pulse width modulation (PWM)) kautta. Pulssinleveyden vaihteluväli on tyypillisesti noin 1 000 – 2 000 µs ja signaalin taajuus 50 Hz. (Robot Platform 2024.) Yleensä myös radiovastaanotin kytketään suoraan autopilottiohjaimen sarjamuotoiseen tuloliitäntään. Radiovastaanottimissa on usein oma PWM-signaalilähtö kullekin ohjauskanavalle, mutta nykyaikaisissa vastaanottimissa käytössä on näiden lisäksi tai näiden sijaan sarjamuotoinen signaali, jossa voidaan välittää kaikkien radiokanavien arvot samassa signaalissa. Tavallisia protokollia ovat mm. PPM, IBUS ja SBus, joita kaikkia ArduPilot-ohjelmisto tukee. ArduPilot-ohjelmistossa voidaan autopilotin ohjaukseen hyödyntää jopa 16:ta radiokanavaa. Pixhawk-autopilottiohjain ei tue kanavakohtaisia PWM-tuloja. (ArduPilot 2024s.)

Bunkerissa moottoreita ohjaava ohjausyksikkö on pohjaosan roisketiiviissä (IP52) suljetussa kotelossa. Moottorien nopeussäätimien kytkeminen suoraan autopilottiohjaimeen edellyttäisi laitteen avaamista ja sähköjärjestelmän muuttamista, mikä vaikuttaisi myös suojausluokitukseen, eikä tällaiseen muutostyöhön ollut tarkoitus ryhtyä. Myös radio-ohjaimen vastaanotin on sijoitettu tähän koteloon, ja samoista käytännön syistä myöskään radiovastaanotinta ei voitu kytkeä suoraan autopilottiohjaimeen eikä sen signaaleja voitu lukea suoraan. Sen sijaan ohjauskomennot ja radio-ohjaimen tiedot oli välitettävä CAN-väylän kautta. CAN-väylässä ohjausdata ilmaistaan standardin ISO 8855 mukaisen koordinaattijärjestelmän mukaan eli akselien X (pitkittäisakseli), Y (pystyakseli) ja Z (poikittainen vaaka-akseli) suuntaisena lineaarisena tai kiertoliikkeenä (AgileX Robotics 2020, 4–5). Näiden mahdollistamista kuudesta liikesuureesta tässä ajoneuvotyypissä relevantteja ovat ainoastaan X-akselin suuntainen lineaarinen liike (eteen-taakse) ja Y-akselin suuntainen kiertoliike (kääntyminen pystyakselin suhteen myötä- ja vastapäivään, ts. ajoneuvon kääntäminen oikealle tai vasemmalle).

Bunkerin radio-ohjain on 10-kanavainen FlySky i6S (KUVA 3), joka tukee iBus- ja sBus-sarjaprotokollia (Flysky Technology co., ltd 2020, 18). Radiovastaanottimen tyyppiä tai tiedonsiirtoprotokollaa ei mainita Bunkerin teknisessä dokumentaatiossa, mutta myöhemmin Bunkerin huoltotyön yhteydessä todettiin vastaanottimen olevan FlySky FS-A8S, josta ohjausyksikölle menee sarjamuotoinen IBus/SBus-signaali (GetFPV 2024). Ohjausyksikkö välittää CAN-väylään (sanomakehykseen 241) taulukon 2 mukaiset radiokanavien signaaleja vastaavat arvot (AgileX Robotics 2021a, 10).



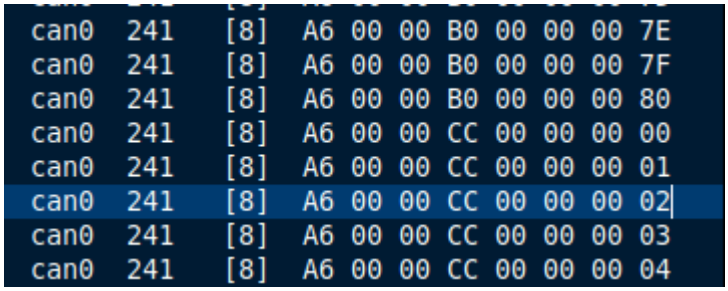
KUVA 3. FlySky i6S -radio-ohjain

TAULUKKO 2. CAN-väylään välitettävät radio-ohjaimen signaalitiedot

tavu	tietotyyppi	Kanava/ohjauslaite	arvot
[0], bitit [0, 1]	uint 8	Vipukytkin SWA, 2-asentoinen	2, 3
[0], bitit [2, 3]		Vipukytkin SWB, 3-asentoinen	2, 1, 3 (bin. 10, 01, 11) (ylhäällä–keskellä–alhaalla)
[0], bitit [4, 5]		Vipukytkin SWC, 3-asentoinen	2, 1, 3
[0], bitit [6, 7]		Vipukytkin SWD, 2-asentoinen	2, 3
[1]	int8	Oikea sauva, vasemmalle-oikealle	[-100, 100]
[2]	int8	Oikea sauva, ylös-alas	[-100, 100]
[3]	int8	Vasen sauva, vasemmalle-oikealle	[-100, 100]
[4]	int8	Vasen sauva, ylös-alas	[-100, 100]
[5]	int8	Vasen potentiometri VRA	[-100, 100]

Itse radio-ohjain tukee kymmentä kanavaa, ja ohjaimen fyysisiä kytkimiä ja säätimiä olisi periaatteessa mahdollista määrittää käyttäjän valitsemille radiokanaville (Flysky Technology co., ltd 2020, 15). Bunkerin ohjausyksikkö lukee kuitenkin vain mainittujen kytkinten ja säätimien tiedot ja välittää CAN-väylään siis yhdeksää kanavaa vastaavat tilatiedot. Oikeanpuoleisen potentiometrin VRB ja ohjaimen taustapuolella olevien kahden painikkeen (joiden tunnukset radio-ohjaimen valikossa ovat Key1 ja Key2) asentojen lukumahdollisuutta ei ole toteutettu.

CAN-väylän tietoja voidaan lukea Bunkerin tietokoneella, kun CAN-kaapeli on kytketty pohjaosaan ja CAN-yhteys on avattu komennolla ”sudo ip link set can0 up type can bitrate 500000”. Yhteyden voi todentaa tulostamalla esim. sanomakehykseen 241 tulevat viestit komennolla ”candump can0,241:7ff” (KUVA 4).



KUVA 4. CAN-viestiliikenteen tuloste

4.6 Radio-ohjaimen hyödyntäminen

Edellisessä kohdassa käsiteltiin radiovastaanottimen tietoja, jotka Bunkerin ohjausyksikkö välittää CAN-väylään sanomakehyksessä 241. Kohdassa 5.6 selostetaan muutokset, joita valmiina olevaan ROS-ohjelmistoon tarvittiin, jotta CAN-väylästä luetut radio-ohjaimen tiedot saatiin sovelluksen käyttöön ROS-viesteinä. Siten ohjaimen vipukytkimien, ohjaussauvojen sekä vasemman potentiometrin asentoja voidaan lukea ohjelmallisesti ja lähettää tietojen perusteella autopilotille komentoja ROS-viestien ja ROS-palvelujen välityksellä.

Sovelluksessa radio-ohjainta on hyödynnetty seuraavien toimintojen käyttämiseen: autopilotin aktivointitilan (*arming*: ARMED, DISARMED) sekä ohjaustilan (*control mode*: AUTO, MANUAL ym.) vaihtaminen, reittipisteen lisääminen laitteen nykyisen sijainnin kohdalle, yksittäisen reittipisteen poistaminen ja koko reittipistelistan tyhjentäminen. Jatkokehittelyä varten ajatuksena oli myös ohjaussauvojen käyttäminen graafisen käyttöliittymän ohjaamiseen Bunkerin tietokoneen näytöllä samalla periaatteella kuin pöytätietokoneessa käytettäisiin hiirtä, vieritysrullaa ja hiiren painikkeita. Tällainen graafinen käyttöliittymä (ks. kohta 5.7.5) toteutettiinkin syyslukukaudella 2023 Käyttöliittymien suunnittelu ja toteutus -kurssia varten, ja sen lähdekoodi on saatavilla GitHub-ohjelmavarastosta (Aaltonen 2023a).

5 JÄRJESTELMÄN KÄYTTÖÖNOTTO JA OHJELMISTOKEHITYS

Käyttöönotto- ja testausvaiheeseen sisältyi perehtymistä järjestelmien ja ohjelmistojen ominaisuuksiin, varsinkin MAVROS-ohjelmiston hyödyntämiseen ja itse ROS-järjestelmään, eri toiminnallisuuksia varten tarvittavien Python-solmujen kirjoittamista ja kehittämistä sekä AgileX Roboticsin kehittämien C++-solmujen muokkaamista radio-ohjaimen syötteen hyödyntämiseksi. MAVROSin dokumentaatio on sinänsä melko kattavasti saatavilla (ROS 2018a), mutta se on käyttötapausten ja ominaisuuksien selostuksen osalta varsin niukka. Jotkin ominaisuudet on kehitetty ja testattu toimimaan yhdessä PX4-laiteohjelmiston mutta ei välttämättä ArduPilot-ohjelmiston kanssa. Useiden ominaisuuksien toimintaa oli selvitettävä kokeilemalla ja tekemällä ohjelmista prototyyppejä sekä selvittämällä eräiden ominaisuuksien toimintaperiaatetta lähdekoodista.

Kehittelytyössä yleisenä visiona oli se, että maa-asemasovelluksella käytettävien reittitoimintojen lisäksi voitaisiin hyödyntää Bunkerin omaa radio-ohjainta niin, että ohjaimen kytkimillä voisi tallentaa reittipisteitä ajon aikana sekä käynnistää ROS-pohjaisia toimintoja. Tämä eliminoisi tarpeen pitää mukana erillistä maa-asematietokonetta. Tarkoituksena oli myös tutkia, millä tavoin autopilottia voi hallita mobiilirobotissa olevan tietokoneen kautta ROS-komentojen ja solmuprosessien avulla MAVROSin välityksellä, joten sovellukseen kehitettiin myös eräitä pelkästään komentorivipäätteen kautta käytettäviä lisätoimintoja.

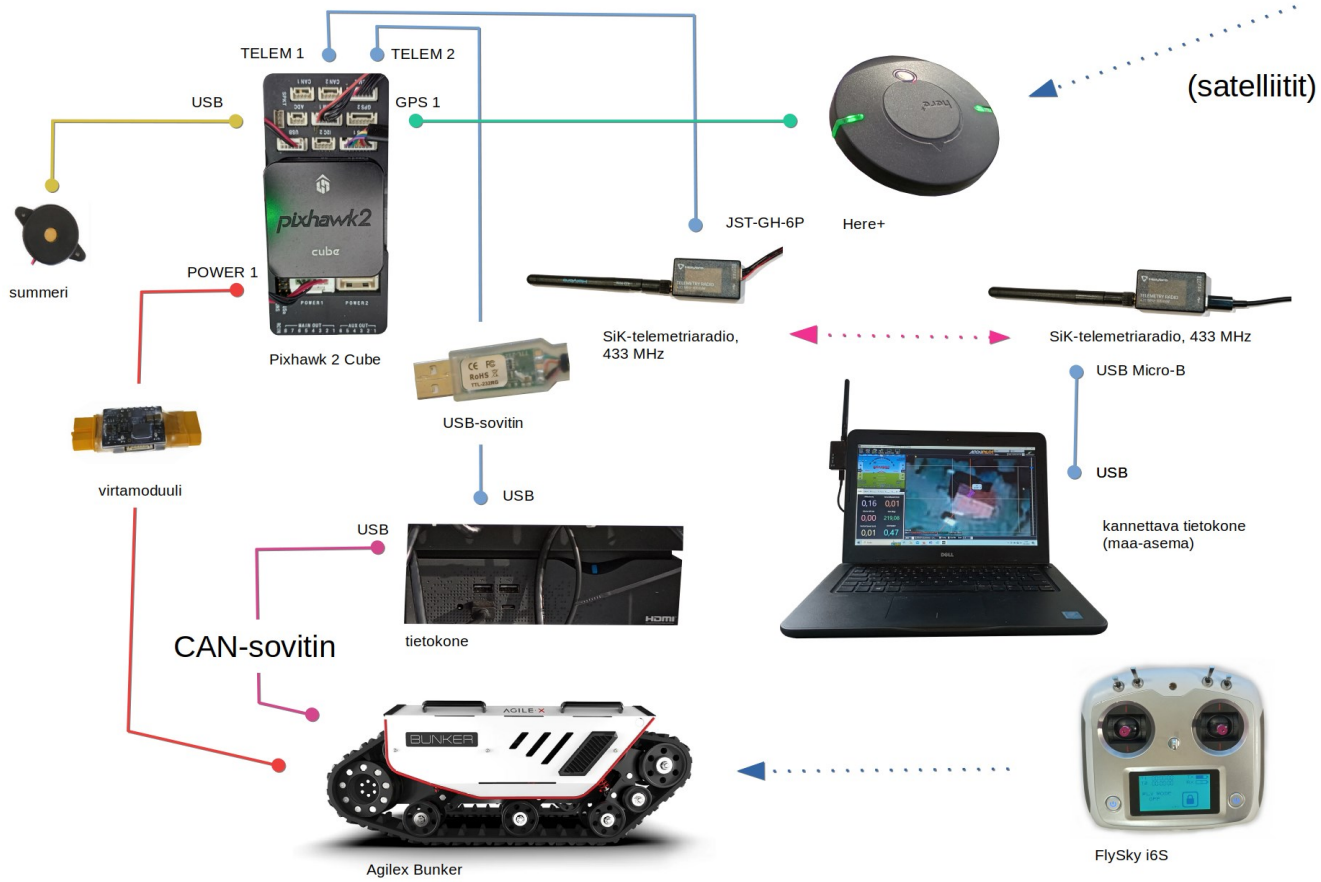
5.1 Autopilottiohjaimen ja GPS-moduulin sijoittelu ja asennus

ArduPilot-laiteohjelmiston maa-ajoneuvoihin tarkoitettu versio on nimeltään ArduRover. Asennusohjeiden mukaan autopilottiohjain pitäisi mahdollisuuksien mukaan pyrkiä sijoittamaan lähelle ajoneuvon painopistettä ja keskilinjaa kulkusuunnan mukaisesti suunnattuna. Ellei tämä ole mahdollista, ArduRoverin parametreihin voidaan asettaa poikkeavan sijainnin tai asennon tiedot, jotta autopilottiohjelmisto voi ottaa poikkeaman huomioon. (ArduPilot 2024t.)

GPS-vastaanotin- ja kompassimoduuli kytketään yleensä laitteen tätä varten osoitettuun sarjaliitäntään (Pixhawk 2 Cubessa liitännässä merkintä ”GPS 1”). Moduuli voidaan kytkeä myös muuhun sarjaliitäntään tai I^2C-liitäntään ja osoittaa liitäntäkohta parametrilla. Järjestelmään kuuluu turvakytkin hätäpysäytystä varten sekä passiivinen pietsosähköinen summeri, jolla voi toistaa äänimerkkimelodioita. Joissakin

autopilottiohjaimissa näille on osoitettu omat liitännät. Tässä järjestelmässä summeri kytkettiin liitäntään ”USB”. Turvakytkin on integroitu Here+-GPS-moduuliin. Eräissä malleissa myös summeri voi olla integroituna GPS-vastaanotin- ja kompassimoduuliin. SiK-telemetriaradio kytkettiin liitäntään ”TELEM 1” ja USB-tietoliikennesovitin liitäntään ”TELEM 2”. Virtamoduulin kaapeli kytkettiin liitäntään ”POWER 1”. Kytkennät on esitetty kuviossa 1, ja liittimet näkyvät myös kuvassa 2.

Pixhawk 2 Cube, Here+-moduuli ja summeri kiinnitettiin teräslevyyn, joka asennettiin Bunkerin kiinnityskehikon etuosaan lähelle ajoneuvon keskilinjaa siten, että laitteisto ei rajoittanut kehikon päälle sijoitetun lidarin näkökenttää. Tässä asennuspaikassa Pixhawk-laitteiston pystyetäisyys Bunkerin pohjaosan koteloinnin yläpinnasta oli noin 25 cm. Etäisyys Bunkerin etuosassa sijaitsevaan akustoon oli noin 40 cm ja takaosassa sijaitseviin moottoreihin noin 70 cm. Centria Drone Labin Pentti Eteläaho teki laitteistoasennukset Centrian toimipaikassa Ylivieskassa ennen kenttätestauksen alkua. Järjestelmän rakenne on esitetty kuviossa 1.



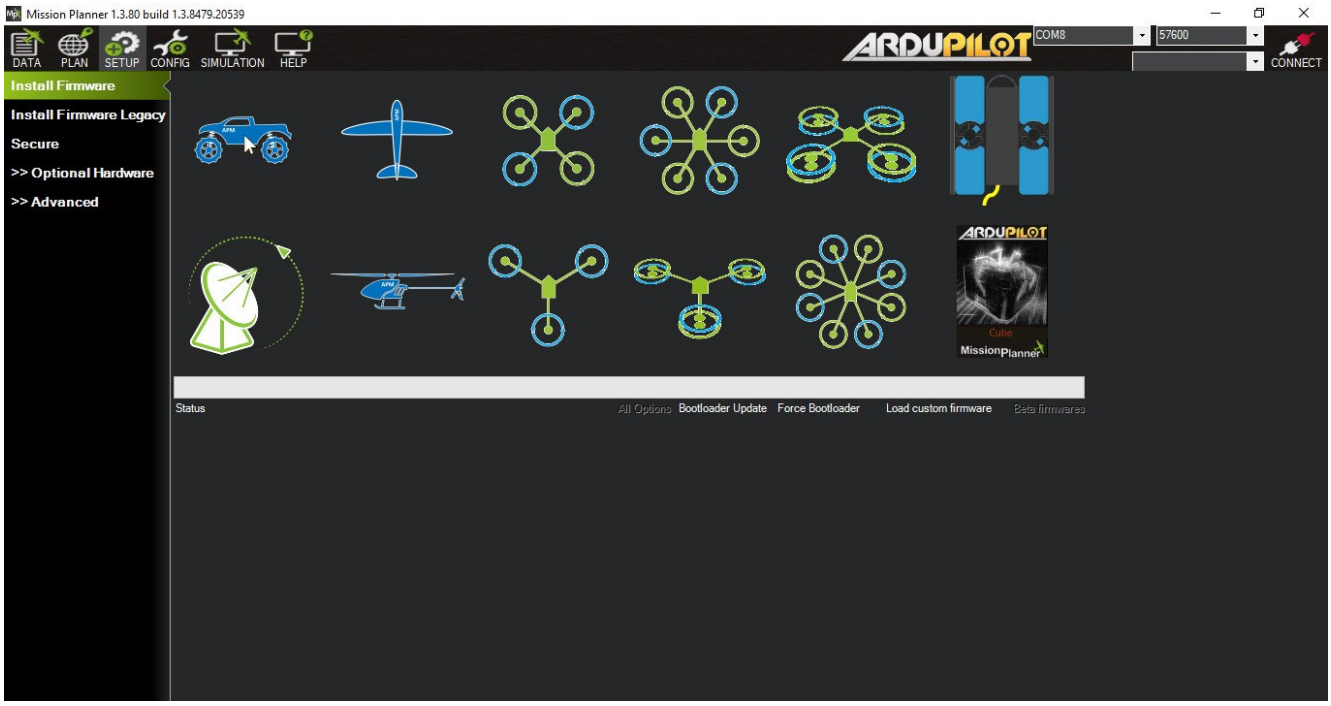
KUVIO 1. Komponenttikaavio

5.2 Mission Planner ja telemetriaradiot

Ohjelmiston asennus aloitetaan tavallisesti asentamalla maa-asemasovellus Mission Plannerin uusin versio (ohjeet: ArduPilot 2024q). Centrian kannettavassa Windows-tietokoneessa oli työn aloitushetkellä kesäkuussa 2023 Mission Plannerin tuore versio (1.3.80, julkaistu maaliskuussa 2023) jo valmiiksi asennettuna. SiK-telemetriaradiot toimivat moitteettomasti oletusasetuksilla. Mission Plannerista on saatavilla myös Android-versio, ja telemetriaradion voi kytkeä matkapuhelimeen USB On-The-Go (OTG) -sovittimen avulla. Tätä vaihtoehtoa kokeiltiin testausvaiheessa lyhyesti, ja telemetriaradioyhteys saatiin toimimaan konfigurointiasetusten säätämisen jälkeen. Ohjeet on esitetty ArduPilotin dokumentaatiossa (ArduPilot 2024h). Android-versio oli sinällään käyttökelpoinen, mutta koska sovelluksen ulkoasu oli käytännössä kopio työpöytäversiosta, käyttöliittymän painikkeet olivat matkapuhelimen ruudulla melko pienikokoisia. Kokeilun perusteella tabletti olisi ollut tätä varten sopivampi laite.

5.3 ArduRover-laiteohjelmiston asennus

Laiteohjelmiston asennusmenettelyyn vaikuttaa se, onko autopilottiohjaimessa ollut valmiiksi asennettuna jokin ArduPilot-versio. Myytävissä Pixhawk-malleissa on usein esiasennettuna PX4-laiteohjelmisto sekä mahdollisesti myös ArduPilot-valmius. Tässä työssä käytetyssä Pixhawk 2 -laitteessa oli ennestään käytetty ArduCopteria, joten asennus eteni aikaisemmin asennettua ArduPilot-versiota koskevien ohjeiden mukaisesti. Tällöin laiteohjelmiston asennus on varsin yksinkertainen toimenpide: autopilottiohjain kytketään laitteen USB-liitännän kautta tietokoneeseen, avataan Mission Planner -maa-asemasovellus, muodostetaan yhteys laitteeseen valitsemalla oikea COM-portti ja käynnistetään laiteohjelmiston asennustoiminto. Asennustoiminnossa valitaan ja vahvistetaan autopilottiohjaimen malli sekä ajoneuvon tyyppi eli tässä tapauksessa maa-ajoneuvo (KUVA 5). Valintojen perusteella maa-asemasovellus lataa verkosta oikean laiteohjelmistovariantin uusimman ohjelmistoversion ja lataa sen laitteeseen. Asennettu ohjelmistoversio oli ArduRover V4.2.3. (ArduPilot 2024m.)



KUVA 5. ArduPilot-laiteohjelmiston asennus Mission Plannerissa

Jos autopilottiohjaimessa ei ole aikaisemmin ollut ArduPilot-ohjelmistoa asennettuna, ladataan ensin tarvittavat laiteajurit ja asennetaan ne laitteen Flash-muistiin erityisellä asennusohjelmalla. ArduPilotin dokumentaatiossa on havainnolliset ja seikkaperäiset asennusohjeet (ArduPilot 2024q). Laiteohjelmiston asennuksen ja uudelleenkäynnistyksen jälkeen autopilotti on käyttövalmis.

5.4 Parametrit ja kalibrointi

Käyttöönoton yhteydessä autopilotin kiihtyvyys- ja gyroskooppianturit sekä kompassi on kalibroitava, ja Mission Plannerissa on tätä varten toiminnot. Normaalisti pitäisi tarkistaa ja kalibroida myös radiovastaanottimen kanavat, mutta koska radiota ei nyt ollut kytketty suoraan autopilottiin, tässä vaiheessa piti sen sijaan poistaa radiosignaalin puuttumiseen liittyvät vikasuojaparametrit käytöstä. Ilman tätä toimenpidettä autopilottia ei olisi voinut kytkeä ARMED-aktivointitilaan. (ArduPilot 2024r.) Käyttöönottovaiheessa asetetut parametriarvot selitteineen on lueteltu taulukossa 3.

TAULUKKO 3. Käyttöönottovaiheessa muutetut autopilotin parametrit

Parametrin nimi	Selite	Muutettu arvo
FS_ACTION	Vikasuojatilanteessa aktivoitava toiminto	0 (ei mitään)
FS_THR_ENABLE	Nopeuden säädön (*throttle*) vikasuoja	0 (ei käytössä)
FS_GCS_ENABLE	Maa-aseman telemetriayhteyden vikasuoja	0 (ei käytössä)
BRD_SAFETYENABLE	Onko turvakytkin käytössä	0 (ei käytössä)
SYSID_ENFORCE	Hyväksytäänkö paketteja muulta kuin oletus-tunnuksella merkityltä maa-asemalta	1 (käytössä)
SYSID_MYGCS	MAVLink-maa-aseman tunnus ID	255
SYSID_THISMAV	Tämän ajoneuvon MAVLink-järjestelmätunnus	1

Testausvaiheen aikana tehtyjä parametrien säätöjä on käsitelty kohdassa 6.4. Mission Plannerin avulla asetetut autopilotin parametrit olivat MAVROSin välityksellä käytettävissä myös ROS-järjestelmässä. Palvelu */mavros/param/pull* kopioi kaikki parametrit ROS-parametreiksi, joten palvelun kutsumisen jälkeen niitä oli mahdollista lukea tavanomaisten ROS-parametrien tavoin. Yksittäisen parametrin arvot voi hakea palvelulla */mavros/param/get*. Autopilotin parametreja voi muuttaa joko yksittäin palvelulla */mavros/param/set* tai lähettämällä ROS-järjestelmästä kaikki */mavros*-hierarkialuokan parametrit palvelulla */mavros/param/push*.

5.5 MAVROS-paketin asennus

AgileX Bunkerin tietokoneessa oli ROS valmiiksi asennettuna, ja sitä oli hyödynnetty mm. lidar-kartoituksissa. Yhteensopivan ROS-version voi asentaa Ubuntun ohjelmistopakettivarastosta ROS-dokumentaatiossa annettujen ohjeiden mukaisesti (ROS 2023b). Lisäksi koneessa oli valmiiksi asennettuna AgileX Roboticsin ja Weston Roboticsin kehittämä ROS-ohjelmisto, jonka avulla Bunkerin ohjausyksikölle voi mm. lähettää liikekomentoja julkaisemalla ROS-viestejä kanavalle */cmd_vel*. Tätä varten avataan CAN-yhteys laitteeseen can0 (ks. kohta 4.5) ja ajetaan pakettiin *bunker_bringup* sisältyvä käynnistystiedosto *bunker_minimal.launch* komennolla ”roslaunch bunker_bringup bunker_minimal.launch”. Sen jälkeen voidaan lähettää liikekomentoviesti. Koodin 1 esimerkkikomento käskee Bunkeria liikkumaan eteenpäin nopeudella 0,2 m/s kääntyen samalla myötäpäivään kulmanopeudella 0,3 rad/s.

```
rostopic pub /cmd_vel geometry_msgs/Twist "linear:
  x: 0.2
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.3"
```

KOODI 1. Twist-tyyppisen viestin julkaiseminen rostopic-komennolla

Tietokoneessa oli asennettuna ROS-versio Kinetic Kame, joten seuraavaksi asennettiin MAVROSin ROS-jakeluversiokohtaiset deb-paketit *ros-kinetic-mavros* ja *ros-kinetic-mavros-extras*. Lisäksi tarvittiin Maan pinnanmuotoon eli geoidiin perustuvia laskelmia varten erityinen kirjastolisäosa. Asennus tehtiin ArduPilot-dokumentaatiossa esitetyn ohjeen (ArduPilot 2024l, luku ”Installing MAVROS”) mukaisesti (KOODI 2).

```
sudo apt install ros-kinetic-mavros ros-kinetic-mavros-extras
wget https://raw.githubusercontent.com/mavlink/mavros/master/
  mavros/scripts/install_geographiclib_datasets.sh
chmod a+x install_geographiclib_datasets.sh
./install_geographiclib_datasets.sh
```

KOODI 2. MAVROS-paketin asennus

Lisäksi käyttäjätunnus oli lisättävä *dialout*-ryhmään, jotta voitiin muodostaa tietoliikenneyhteys autopilottiin komennolla ”sudo usermod -a -G dialout [käyttäjätunnus]”.

Pixhawk oli kytketty USB-sarjaliitäntäsovittimen kautta tietokoneen USB-porttiin, ja se näkyi laitteena */dev/ttyUSB0*. Solmu *mavros_node* voitiin nyt käynnistää komennolla ”rosrun mavros mavros_node _fcu_url:=/dev/ttyUSB0:57600 _system_id:=1”. Tässä rosrun-komennon yhteydessä annetaan parametri *fcu_url*, jonka arvo koostuu autopilotin laitenimestä (*/dev/ttyUSB0*) ja tiedonsiirtonopeudesta baudeina (57 600), ja parametri *system_id*, jonka arvon on vastattava autopilottiin asetettua parametria *SYSID_THISMAV* (ks. kohta 5.4). Tarvittaessa oikea laitenimi voidaan tarkistaa esim. *journalctl*-komennolla. Jos laite kytketään suoraan Pixhawkin Cube-moduulin USB-liitännästä tietokoneen USB-porttiin, laitenimeksi tulee */dev/ttyACM0*.

5.6 AgileX Bunkerin alkuperäisen ROS-ohjelmiston muutokset

Kuten edellä kohdassa 5.5 on selostettu, Bunkerissa oli valmiiksi asennettuna ohjelmisto, jonka avulla ohjausyksikölle voidaan antaa liikekomentoja ROS-järjestelmän välityksellä, kun CAN-yhteys on avattu ja tarvittavat ROS-solmut on käynnistetty esim. *bunker_bringup*-pakettiin kuuluvan *bunker_minimal.launch*-käynnistystiedon avulla. Radio-ohjaimen tilatietoja oli mahdollista tarkastella suoraan CAN-väylän viestiliikennettä lukemalla, mutta alkuperäisessä ohjelmistossa tietojen välittämistä ROS-järjestelmään ei ollut toteutettu. Järjestelmä kuitenkin julkaisi mm. moottorin sekä lisävarusteisiin kuuluvan valojärjestelmän tietoja kanavalla */bunker_status*. Jotta radio-ohjaimen kytkimiä voitaisiin hyödyntää ROS-järjestelmän välityksellä autopilottisovelluksen ohjaamisessa, alkuperäiseen ohjelmistoon oli tehtävä eräitä lisäyksiä ja korjauksia.

Bunkerin tilatietoja julkaiseva solmu *bunker_messenger* perustuu WestonRobotin kehittämään pakettiin *ugv_sdk* (versio v1.x (WestonRobot 2021)) ja AgileX Roboticsin kehittämään pakettiin *bunker_base* (AgileX Robotics 2021b). Paketin *ugv_sdk* otsikkotiedostossa *agx_protocol_v2.h* oli muiden tilatietoja koskevien tietorakenteiden yhteydessä valmiina struct-rakenne *RcStateMessage*, jota ei kuitenkaan hyödynnetty viestien julkaisemisessa. Ohjelmistoon tarvittiin muutokset, joilla tämä tietorakenne lisättiin muita tilatietoja sisältävän tietorakenteen *BunkerState* jäseneksi. Muutoksen jälkeen solmu *bunker_messenger* ottaa nämä tiedot huomioon lukiessaan CAN-väylän tietoja ja julkaistessaan niitä ROS-viesteinä. Radio-ohjaimen tilaviestejä varten luotiin uusi viestityyppi *BunkerRCState*, johon määriteltiin struct-rakenteen *RcStateMessage* tietosisältöä vastaavat kentät (KOODI 3).

```
uint8 sws
int8 var_a
int8_t right_stick_left_right
int8_t right_stick_up_down
int8_t left_stick_left_right
int8_t left_stick_up_down
```

KOODI 3. Viestityypin BunkerRCState määritelmä

Lisäksi alkuperäisen *RcStateMessage*-rakenteen määrittelyssä oli eräitä virheellisiä tietotyyppejä, jotka oli korjattava. Kaiken kaikkiaan muutoksia tarvittiin seuraaviin tiedostoihin (polkuviittaukset ovat asennushakemiston /home/agx/catkin_ws/src/base_ros alihakemistoja):

- ugv_sdk/ugv_sdk/include/ugv_sdk/proto/agx_protocol_v2.h: korjattiin struct-tietorakenteen *RcStateMessage* väärät tietotyypit *uint8* muotoon *int8*, jotta ohjaussauvojen analogiset kanavat saavat määrittelyn mukaiset arvot -100...100 (AgileX Robotics 2021a, 10).
- ugv_sdk/ugv_sdk/include/ugv_sdk/bunker/bunker_types.hpp: lisättiin struct-rakenteeseen *BunkerState* jäsen *RCState.*
- ugv_sdk/ugv_sdk/src/bunker_base.cpp: lisättiin switch-case-rakenteeseen kohta, joka vie CAN-väylästä luetun *RCStateMessage*-tiedon viestiolion *status_msg* jäseneen *body.rc_state.*
- bunker_base/bunker_msgs_msg/BunkerRCState.msg: luotiin uusi viestityyppi kanavalla */bunker_rc_status* julkaistavia radio-ohjaimen tilatietoja varten.
- bunker_base/bunker_msgs/CMakeLists.txt: lisättiin edellisessä kohdassa luotu uusi tiedosto BunkerRCState.msg projektin viestitiedostoluetteloon.
- bunker_base/src/bunker_messenger.cpp: lisättiin uusi julkaisija viestikanavaa */bunker_rc_status* varten.
- bunker_base/include/bunker_messenger.hpp: lisättiin uutta julkaisijaa vastaava määritelmä otsikkotiedostoon.

Muokatut lähdekooditiedostot ovat saatavilla GitHub-palvelussa (Aaltonen 2023b ja Aaltonen 2023c).

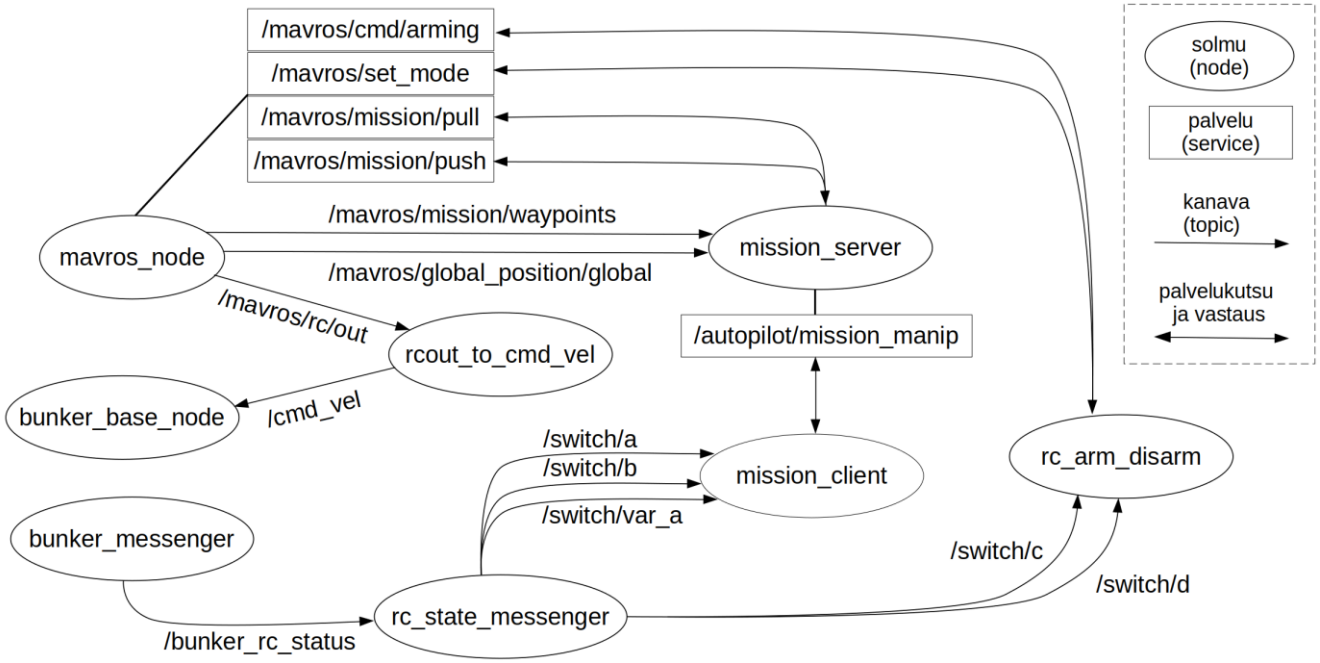
5.7 Automaattiajon toimintoja ohjaavan ROS-ohjelmiston kehittäminen

ROS-järjestelmässä yhteys autopilottiin muodostetaan käynnistämällä solmu *mavros_node* (ks. kohta 5.5). Solmu julkaisee autopilotin tietoja välittäviä viestejä ja kuuntelee ROS-järjestelmästä tulevia ohjausviestejä *mavros*-hierarkiahaaraan kuuluvilla kanavilla sekä käynnistää toimintaa ohjaavia ROS-palveluja. Jotta Bunkeria voidaan ohjata tietokoneen ja CAN-väylän avulla, käynnistetään paketin *bunker_base* solmut *bunker_messenger* ja *bunker_base_node*.

Automaattiohjaustoimintojen kehittämistä varten luotiin aluksi ROS-paketti *bunker_autopilot*. Uusi paketti luodaan komennolla catkin_create_pkg, jonka parametreina annetaan uuden paketin nimi sekä paketin riippuvuudet muista paketeista ja moduuleista komennolla ”catkin_create_pkg bunker_autopilot std_msgs rospy mavros_msgs message_runtime”.

Komento luo tarvittavan hakemistorakenteen ja asetustiedostot. Riippuvuuksia ja muita asetuksia voi muuttaa – esim. viestityyppien lisäämistä varten – muokkaamalla tiedostojen *package.xml* ja

CMakeLists.txt sisältöä. Tarkemmat ohjeet on esitetty ROSin dokumentaatiossa. Muutosten jälkeen paketti päivitetään komennolla ”catkin_make”. Tiedostoille annetaan ajo-oikeus esim. komennolla ”chmod +x [tiedoston nimi]”. Tämän jälkeen solmu voidaan ajaa komennolla ”rosrun [paketin nimi] [solmun tai tiedoston nimi]”.



KUVIO 2. Autopilottisovelluksen ROS-järjestelmän solmut, palvelut ja kanavat

Kuviossa 2 on esitetty autopilottisovelluksen ROS-järjestelmän looginen rakenne ja prosessien keskinäiset suhteet. Ellipsit esittävät solmuja, suorakulmiot palvelinsolmujen tarjoamia palveluja ja tekstiselitteellä varustetut nuolet viestikanavia. Kaksisuuntaiset nuolet esittävät palvelukutsua ja palvelun vastausviestiä.

Järjestelmä laadittiin mahdollisimman vikasietoiseksi ja modulaariseksi siten, että kullakin viestiliikennettä muokkaavalla ja välittävällä solmulla oli tietty tehtävä, kuten moottorinohjausdatan välittäminen (*rcout_to_cmd_vel*), radio-ohjaimen syötteen lukeminen (*rc_state_messenger*) tai reittimuutospyyntöjen välittäminen (*mission_client*) ja autopilotin tilojen ohjaaminen (*rc_arm_disarm*) ohjaimen syötteen perusteella. Modulaarisen rakenteen ansiosta samoja ohjelmakomponentteja voidaan myös hyödyntää erilaisissa laitteistokokoonpanoissa. Esimerkiksi jos järjestelmä otettaisiin käyttöön ajoneuvossa, jossa moottorien nopeudensäätimet ja mahdollinen ohjausservo kytketään suoraan autopilottiohjaimeen, solmut *rcout_to_cmd_vel* ja *bunker_base_node* jäisivät tarpeettomiksi. Samaten jos radiovastaanotin kytkettäisiin autopilottiohjaimen tuloliitäntään, solmu *rc_state_messenger* voitaisiin korvata versiolla,

joka lukee radiolähettimeltä tulevan raakadatan viestikanavalta */mavros/rc/in*, ja solmu *rc_arm_disarm* voitaisiin korvata autopilotin omilla sisäisillä toiminnoilla. Jos reitin hallintaa varten otettaisiin käyttöön radio-ohjaimen sijaan tai ohelle muu käyttöliittymä, uusi asiakassolmu voisi toimia rinnakkain *mission_client*-solmun kanssa.

5.7.1 Autopilotilta tulevan ohjausdatan saattaminen CAN-väylään

Automaattiohjauksen toiminnan kannalta *bunker_autopilot*-paketin oleellisin solmu on *rcout_to_cmd_vel* (LIITE 1), joka lukee autopilotin AUTO-ohjaustilassa tuottamat vasemman ja oikean moottorin PWM-signaaliarvot kanavalta */mavros/rc/out*, laskee niiden perusteella halutun lineaarisen ja kulmanopeuden ja lähettää tuloksen Twist-tyyppisenä viestinä kanavalle */cmd_vel*. Solmu *bunker_base_node* lukee tälle kanavalle julkaistut viestit ja välittää niitä vastaavat ohjauskomennot CAN-väylän kautta Bunkerin ohjausyksikölle.

Twist-viestin nopeusarvojen laskennan taustatietoina käytetään lähdekoodiin kiinteästi kirjattuja PWM-signaalin minimi- ja maksimiarvoja 900 ja 2100 sekä lineaarisen ja kulmanopeuden säätökertoimia *_speed_factor_lin_x* ja *_speed_factor_ang_y*, joiden kummankin oletusarvoksi on asetettu 1,0. Tarvittaessa kertoimille voidaan antaa muu arvo ROS-parametreilla */autopilot/speed_factor_lin_x* ja */autopilot/speed_factor_ang_y*. Ohjelma tarkistaa parametrit käynnistyksen yhteydessä ja päivittää nopeuskertoimet tämän perusteella. Käytännön testeissä nämä nopeuskertoimet (kumpikin 1,0) osoittautuivat sopiviksi, eikä niitä ollut tarvetta muuttaa.

Funktio *pwm_to_adimensional()* muuntaa vasemman ja oikean moottorin PWM-signaalit (*msg.channels[0]* ja *msg.channels[2]*) minimi- ja maksimiarvojen perusteella arvovälille [-1,1], ja funktio *update_throttle()* laskee näiden sekä yleisten nopeuskertoimien perusteella x-akselin suuntaisen lineaarisen nopeuden ja y-akselin suuntaisen kulmanopeuden. Lineaarinen nopeuskomponentti lasketaan yksinkertaisesti vasemman ja oikean moottorin PWM-signaaleista laskettujen arvojen keskiarvon perusteella (positiivinen tulos tarkoittaa liikettä eteenpäin ja negatiivinen taaksepäin). Kulmanopeus lasketaan oikeaan ja vasempaan moottoriin viittaavien arvojen erotuksen perusteella (positiivinen tulos tarkoittaa kiertoliikettä myötäpäivään ja negatiivinen vastapäivään). Ajastinfunktion *rospy.Rate()* ohjaamana julkaisufunktio lähettää Twist-viestin kanavalle */cmd_vel* 50 kertaa sekunnissa.

5.7.2 Radio-ohjaimen syötteen välittäminen autopilottisovellukselle

Radio-ohjaimen tietojen välittämisestä ROS-järjestelmään vastaa solmu *rc_state_messenger* (LIITE 2). *Subscriber*-lukumetodi *self.sub* lukee ohjaimen raakadataa kanavalta */bunker_rc_status* ja kutsuu funktiota *callback_rc_status()* aina, kun kanavalta on luettu uusi viesti. Funktio vertaa viestin sisältämiä arvoja muuttujiin tallennettuihin arvoihin, ja jos johonkin vipukytkimeen viittaava arvo on muuttunut (kytkin on käännetty eri asentoon), se lähettää uuden arvon kyseisen kytkimen viestikanavalle. Käynnistyksen yhteydessä kullekin kanavalle muodostetaan omat julkaisijansa (*Publisher*-metodi). Jokaista neljää kytkintä ja yhtä potentiometriä varten on oma kanava sen vuoksi, että uusia viestejä julkaistaan ainoastaan kytkimen tai säätimen tilan muuttuessa. Tämä ratkaisu helpottaa viestien käsittelyä viestejä vastaanottavissa solmuissa, sillä tilamuutoksiin voidaan reagoida sitä mukaa kuin *rc_state_messenger* julkaisee kyseisen solmun kannalta relevantin kytkimen tilaa koskevia viestejä.

5.7.3 Autopilotin toimintatilojen ohjaaminen radio-ohjaimella

Jos radiovastaanotin kytketään suoraan autopilottiohjaimeen, aktivointi- ja ohjaustilan muutokset voidaan määrittää tietyille radiokanaville niin, että autopilotti toteuttaa tilamuutokset itse suoraan radiolähettimen syötteen perusteella (ArduPilot 2024d). Koska radiovastaanottimen signaalia ei tässä käyttösovelluksessa saada suoraan autopilotille, tietojen lukeminen hoidetaan ROS-järjestelmässä ja tilamuutoskomennot välitetään autopilotille MAVROS-yhteyden kautta.

Solmu *rc_arm_disarm* (LIITE 3) ohjaa autopilotin kahta eri toimintatilakategoriaa: moottorien aktivointitilaa ja ohjaustilaa. DISARMED-tilassa autopilotti pitää sähkömoottorit paikoillaan, jolloin laite ei liiku. Kääntyvillä etupyörillä varustetussa ajoneuvossa (Ackermann-ohjaus) ohjausservo olisi kuitenkin toiminnassa. ArduRoverissa on useita ohjaustiloja, mutta tässä sovelluksessa radio-ohjaimen kytkimille on määritetty ohjaustilat MANUAL (käsiohjaus) ja AUTO (reitin suorittaminen automaattisesti). MANUAL-ohjaustila ei Bunkerin tapauksessa ole sinällään toiminnallinen, koska autopilotille ei mene radio-ohjaimen syötettä. Kun radio-ohjaimen SWB-kytkin käännetään keskiasentoon, Bunkerin manuaalinen ohjaus ohittaa autopilotin toiminnan. Loppuun suoritettu automaattinen ajoreitti voidaan käynnistää uudelleen, kun ohjaustila kytketään AUTO-tilasta MANUAL-tilaan ja sitten takaisin AUTO-tilaan. Ohjaustila on tarpeellinen myös siksi, että ARMED-aktivointitila voidaan kytkeä vain MANUAL-ohjaustilassa (ArduPilot 2024c).

Solmu *rc_arm_disarm* lukee kanavia */switch/c* ja */switch/d*, joille tulee viestejä aina kytkinten C (ohjaustila) ja D (aktivointitila) asennon muuttuessa, ja kutsuu tämän perusteella palveluja */mavros/cmd/arming* ja */mavros/set_mode*. *Arming*-palvelulle lähetetään bool-muotoinen viesti (*true*: pyydetään tilaa ”armed”, *false*: pyydetään tilaa ”disarmed”) ja *set_mode*-palvelulle halutun ohjaustilan nimi (string) tai lukuarvo (int). Palvelun */mavros/set_mode* kautta voidaan käyttää kaikkia ArduRoverin tukemia ohjaustiloja: MANUAL (0), ACRO (1), STEERING (3), FOLLOW (6), SIMPLE (7), DOCK (8), AUTO (10), RTL (11), SMART RTL (12) ja GUIDED (15) (ArduPilot 2024g).

5.7.4 Reittipisteiden lisääminen ja poistaminen ja muut reittitoiminnot

Reitin muokkaamisesta huolehtii solmun *mission_server* (LIITE 4) tarjoama palvelu */autopilot/mission_manip*. Palvelusolmu lukee kanavaa */mavros/mission/waypoints*, jolla *mavros_node* julkaisee reittipisteiden tiedot, ja kopioi tiedot omaan sisäiseen listaansa. Kun palvelukutsun käsittely johtaa reitin muuttumiseen, solmu tekee muutokset ensin omaan sisäiseen listaansa. Sen jälkeen se kutsuu palvelua */mavros/mission/push* ja lähettää koko listan palvelukutsun argumenttina.

MAVROSin palvelu */mavros/mission/pull* palauttaa reittipisteiden lukumäärän, joka tosin saadaan myös kanavalla */mavros/mission/waypoints* julkaistavan reittipistetaulukon pituuden perusteella. Palvelun */mavros/mission/pull* kutsuminen on kuitenkin tarpeen aina reitin muutosten jälkeen, koska tämä palvelukutsu saa *mavros_node*-solmun päivittämään kanavalla julkaistavien viestien tiedot. Ilman *pull*-palvelun kutsua *mavros_node* voi jäädä julkaisemaan *waypoints*-kanavalla vanhentuneita tietoja. Sen vuoksi funktio *wpPush()* kutsuu */mavros/mission/push*-palvelun kutsun yhteydessä aina myös *mission_server*-solmun funktiota *wpPull()*, joka kutsuu palvelua */mavros/mission/pull*.

Mission_manip-palvelua voidaan kutsua komentoriviltä siten, että haluttu toiminto osoitetaan palvelukutsun parametrilla *task*. Palvelun toiminnot ja niitä varten annettavat parametrit on esitetty taulukossa 4. Koodin 4 mukaisella komennolla kutsuttaisiin toimintoa 1 (ADD_WP) ja pyydettäisiin lisäämään uusi reittipiste olemassa olevan reittipisteen nro 5 jälkeen.

```
rosservice call /mission_manip "{task: 1, use_last: false,
  use_current: false, seq: 5, all_wps: false, distance: 0.0,
  direction_angle: 0.0, scale_factor: 0.0}"
```

KOODI 4. Palvelun mission_manip toiminnon ADD_WP kutsuminen rosservice-komennolla

TAULUKKO 4. Mission_manip-palvelun toiminnot ja parametrit

int8 task	toiminnon nimi	bool use_last	bool use_current	int8 seq	bool all_wps	float32 distance	float32 direction_angle	float32 scale_factor
		X: parametri käytössä ko. toiminnossa						
0	SET_HOME							
	Asettaa nykyisen sijainnin ”kotipisteeksi” (home) eli reittipisteeksi nro 0							
1	ADD_WP	X	X	X				
	Lisää nykyisen sijainnin uudeksi reittipisteeksi reitin loppuun (*use_last = true*), parhaillaan ajovuorossa olevan reittipisteen jälkeen (*use_current = true*) tai osoitetun reittipisteen jälkeen (*seq* = järjestysnro)							
2	REMOVE_WP	X	X	X				
	Poistaa reitin viimeisen, ajovuorossa olevan tai järjestysnumeron osoittaman reittipisteen.							
3	CLEAR_MISSION							
	Poistaa kaikki reittipisteet.							
4	BACKWARDS_MISSION							
	Järjestää kaikki reittipisteet lopusta alkuun.							
5	OFFSET_MISSION			X	X	X	X	
	Siirtää yhtä reittipistettä (*seq*) tai kaikkia (*all_wps*) reittipisteitä annetun metrimäärän (*distance*) annettuun suuntaan (*direction_angle*; 0,0 = pohjoinen, 90,0 = itä jne.).							
6	SCALE_ROTATE_MISSION						X	X
	Kiertää reittiä annetun kiertokulman (*direction_angle*) verran myötäpäivään ja skaalaa reitin koon kertoimen (*scale_factor*) mukaiseksi (1,0 = alkuperäinen koko).							
7	MIRROR_MISSION						X	
	Peilaa reitin sen keskipisteen kautta kulkevan linjan suhteen siten, että linjan suunta vastaa annettua kompassisuuntaa (*direction_angle*).							
1–3: käytettävissä radio-ohjaimen kytkimillä (solmu *mission_client* tekee palvelukutsun)								

Toimintoja 1–3 voidaan kutsua asiakassolmusta *mission_client* (LIITE 5), joka seuraa radio-ohjaimen vipukytkimen A ja potentiometrin A (*var_a*) tilojen muutoksia kanavilta */switch/a* ja */switch/var_a*. Kun kytkin A kytketään ala-asentoon, asiakassolmu pyytää *mission_manip*-palvelulta haluttua toimintoa potentiometrin asennon perusteella: jos potentiometriä on käännetty alaspäin (arvot [-100, -26]), kutsutaan palvelun *mission_manip* toimintoa CLEAR_MISSION, jolla tyhjennetään reittipistelista. Jos

potentiometriä on käännetty ylöspäin (arvot [26, 100]), kutsutaan toimintoa REMOVE_WP eli poistetaan ajovuoroindeksin osoittama (seuraavaksi kohteena oleva) reittipiste. Jos potentiometri on keskiasennossa (arvot [-25, 25]), kutsutaan toimintoa ADD_WP, jolla lisätään reitin loppuun uusi reittipiste nykyisen GPS-sijainnin perusteella. Potentiometrin tila saadaan kokonaislukuna arvoalueella [-100, 100], missä 0 vastaa keskikohtaa. Potentiometrin säätökiekko ei kuitenkaan aina palaudu aivan nollakohtaan, ja sen vuoksi arvojen tulkintaan on lisätty 25 yksikön marginaali kumpaankin suuntaan.

Lisäksi *mission_manip*-palvelusta voidaan kutsua komentorivin kautta seuraavia toimintoja: Toiminto BACKWARDS_MISSION järjestää reittipistelistan alkiot [1]–[n] takaperin järjestykseen [n]–[1]. Listan ensimmäinen alkio [0] on ns. kotipiste (*home*) eli yleensä sijainti, jossa autopilotti on käynnistetty. Se jätetään ennalleen. Tämän toiminnon avulla sama reitti voidaan ajaa lopusta alkuun, ja sen pohjalta voisi pienellä lisätyöllä kehittää ajo-ohjelman, joka esim. ajaa samaa reittiä toistuvasti edestakaisin. Toiminnolla OFFSET_MISSION voidaan siirtää kaikkia reittipisteitä [1]–[n] parametrilla *distance* ilmoitetun metrimäärän parametrilla *direction_angle* ilmoitettuun kompassisuuntaan (0–360). Tämä on hyödyllinen esim. tilanteissa, joissa GPS-koordinaattien todellinen sijainti poikkeaa muutaman metrin edellisen ajokerran tilanteesta, kuten testien aikana usein tapahtui. Toiminnolla SCALE_ROTATE_MISSION voidaan skaalata reittiä suuremmaksi tai pienemmäksi sekä kiertää reittikuviota koordinaattien ääriarvoista lasketun keskipisteen ympäri. Parametreina annetaan kokokerroin *scale_factor* (1,0 = alkuperäinen koko) ja kiertokulma *direction_angle*. Kiertokulma annetaan asteina siten, että negatiivinen arvo kiertää reittikuviota vastapäivään ja positiivinen myötäpäivään. Toiminnolla MIRROR_MISSION peilataan reittikuvio suhteessa em. keskipisteen kautta kulkevaan akseliin, joka kulkee parametrilla *direction_angle* osoitettuun suuntaan. Nämä lisätoiminnot täydentävät ja tukevat radio-ohjaimen kytkimillä käytettäviä perustoimintoja 1–3, jotka on selostettu edellä.

Palvelusolmu laskee metrimääräisiä etäisyyksiä varten likiarvokertoimet *k_lat* ja *k_long*, joilla koordinaattiarvojen erotukset muunnetaan metreiksi. Likiarvo *k_lat* lasketaan maapallon päiväntasaajan halkaisijan (vakio *diagEq*) perusteella. Jotta etäisyydet vastaisivat paremmin Suomen sijaintia, Maan halkaisijan lyheneminen napoja kohti eli litistyneisyys on huomioitu korjaamalla arvoa *diagEq* noin 3 promillea suuremmaksi. Korjauksen suuruus on määritetty mittaamalla koordinaattipisteiden välinen paikallinen etäisyys Maanmittauslaitoksen karttapalvelusta (Maanmittauslaitos 2024a). Kerroin *k_long* lasketaan määrittämällä leveyspiirin pituus kyseisen leveyspiirin ja päiväntasaajan välisen kulman perusteella. Kertoimet lasketaan solmun käynnistyksen jälkeen kanavalta */mavros/global_position/global* luettujen paikkatietojen perusteella siten, että Maa oletetaan pyöreäksi: leveysasteiden välinen etäisyys maastossa on vakio, ja pituusasteiden välinen etäisyys lasketaan leveyspiirin kehän (l) mukaan

$$l = \pi \cos(lat) \cdot 2R \quad (1)$$

missä lat on leveyspiirin ja päiväntasaajan välinen kulma asteina ja R päiväntasaajan säde (6 371 000 m). Sen jälkeen sovelluksessa käytetään paikallisten etäisyyksien laskentaan likiarvomallia, jossa pituuspiirit oletetaan yhdensuuntaisiksi: paikallinen koordinaatisto oletetaan tasomaiseksi xy-koordinaatistoksi, jossa metrimääräiset etäisyydet vastaavat koordinaattiarvojen muutosta (asteina ja desimaaleina) tietyillä muuntokertoimilla.

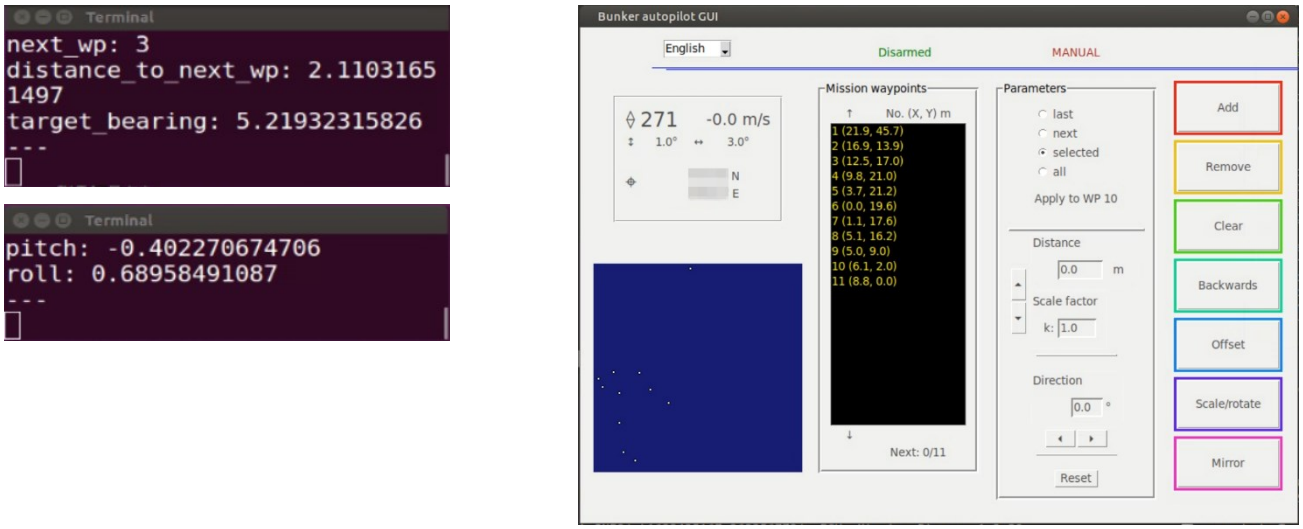
Pallon pinnalla sijaitsevien koordinaattipisteiden välisen etäisyyden (d) laskemiseen voisi käyttää myös haversiniin perustuvaa laskentakaavaa, joka olisikin relevantti pitkillä matkoilla, esim. lentoliikenteessä:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right) \quad (2)$$

missä r on pallon säde, φ_1 ja λ_1 pisteen 1 leveys- ja pituuskoordinaatit ja φ_2 ja λ_2 pisteen 2 leveys- ja pituuskoordinaatit. Tässä sovelluksessa robotti liikkuu kuitenkin varsin pienellä alueella, todennäköisesti joka tapauksessa enintään kilometrin-parin suuruisilla etäisyyksillä. Haversiniin perustuva laskentakaava olisi tarpeettoman raskas operaatio suoritettavaksi kymmeniä kertoja sekunnissa – suorituskykytestissä haversinikaavalla lasketun etäisyyden määrittämiseen kului noin 2,5-kertainen aika likiarvokertoimella laskettuun arvoon verrattuna. Tämänkin laskentakaavan tuloksia olisi korjattava litistyneisyyden osalta, koska kaava perustuu pallon muotoon eikä geoidiin. Maassa kulkevan laitteen kannalta pinnanmuotojen vaihtelut olisivat merkittävä kuljettua matkaa vääristävä tekijä, joten tarkka etäisyys linnuntietä olisi tässä sovelluksessa ja mittakaavassa jokseenkin epärelevantti tieto.

5.7.5 Käyttöliittymätoiminnot

Ajonaikaista järjestelmän seurantaa varten määritettiin uudet viestityypit WPInfo ja Angles (LIITE 6), joiden avulla voitiin seurata ajoneuvon asentoa ja reitillä etenemistä. Reittipistepalvelua tarjoava solmu *mission_server* julkaisee WPInfo-tyyppisiä viestejä, joissa näkyy reittipisteiden kokonaismäärä (*total_wps*), seuraavan reittipisteen numero (*next_wp*) sekä metrimääräinen etäisyys ja kompassisuunta seuraavaan reittipisteeseen (*distance_to_next_wp* ja *target_bearing*). Prototyyppivaiheen aikainen solmu *dashboard* julkaisi Angles-tyyppisiä viestejä, joissa ilmoitettiin ajoneuvon pituus- (*pitch*) ja vaakasuuntainen kallistus (*roll*) asteina. Angles-tyypin tiedot laskettiin *mavros_node*-solmun julkaisemien kiihtyvyystietojen perusteella. Viestityyppejä seurattiin komentoriviltä komennon *rostopic echo* avulla solmun *dashboard* avaamista pääteikkunoista (KUVA 6).



KUVA 6. Käyttöliittymätoimintoja

Myöhemmin järjestelmän täydennykseksi ja *dashboard*-solmun korvaajaksi toteutettiin graafinen käyttöliittymä Python-solmulla *autopilot_gui*, joka laski ja esitti kallistuskulmat samojen anturitietojen perusteella (KUVA 6). Graafisen käyttöliittymän toteutus kuului Centriassa syyslukukaudella 2023 pidetyn Käyttöliittymien suunnittelu ja kehittäminen -kurssin suoritukseen.

ArduRover soittaa autopilottiohjaimeen liitetyllä summerilla äänimerkkejä mm. laitteen käynnistyksen ja satelliittiyhteyden muodostamisen yhteydessä sekä automaattiajossa ajoneuvon saavuttaessa viimeisen reittipisteen. Kehittelyvaiheessa tuli ilmi tarve saada ohjelmallisten toimintojen todentamiseksi äänipalaute myös yksittäisten reittipisteiden saavuttamisesta sekä reittipisteiden lisäämisestä ja poistamisesta radio-ohjaimella. MAVROSissa on summerilla soitettavien melodiaviestien vastaanottamista varten kanava */mavros/play_tune*, mutta toistaiseksi toiminnon toteutus ei toimi ArduPilotin käyttämällä ääniformaatilla. Vaihtoehtoinen toteutus saatiin aikaan lua-skriptillä (LIITE 7), joka tallennettiin Mission Plannerin avulla suoraan Pixhawkin microSD-kortille. Kun reitin pituus muuttuu, skripti antaa laitteelle komentoja, jotka soittavat laitteen summerilla skriptiin kirjoitetun melodian. Reittipisteen lisäämiselle, reittipisteen poistamiselle ja koko reitin tyhjentämiselle ”sävellettiin” kullekin oma melodiansa, joten käyttäjä saa järjestelmän tilan muutoksista tarkemman äänipalautteen.

5.7.6 Käynnistystiedosto

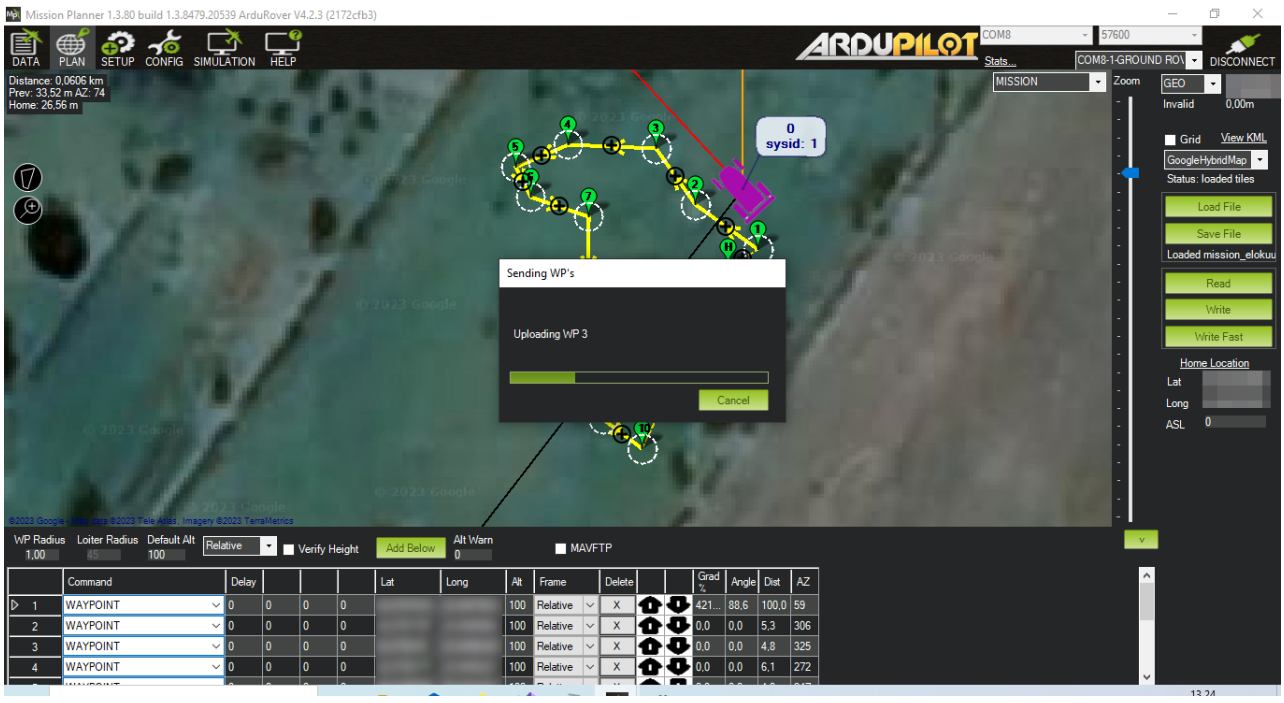
Sovelluksessa tarvittavat ohjelmasolmut parametreineen voidaan käynnistää komentoriviltä erikseen kukin omaan pääteikkunaansa, mutta laajassa järjestelmässä tämä on melko työläs ja sekava menetelmä. Ajettavan kokoonpanon voi koota käynnistystiedostoksi (*launch file*), jossa voidaan määrittää tarpeen mukaan parametreja kullekin solmulle tai koko järjestelmälle. Jos ROS-isäntä ei ole valmiiksi käynnissä, se käynnistetään käynnistystiedoston ajamisen yhteydessä. Solmujen tulosteet voi joko jättää tulostumatta tai ohjata tulostumaan samaan pääteikkunan näkymään (käynnistysparametrilla *output=”screen”*). Tiedostoon voi sisällyttää myös muita käynnistystiedostoja. (Pyo, Cho, Jung & Lim 2017, 98, 189–193.) Siten samoista lähdekooditiedostoista voi koota useita erilaisia käynnistyspaketteja, jotka voi ajaa yhdellä komennolla. Projektissa kehitettyjen solmujen keskitettyä käynnistämistä varten kirjoitettiin käynnistystiedosto (LIITE 8), joka käynnistää AgileX Roboticsin ohjelmistoon kuuluvan käynnistystiedoston *bunker_minimal.launch*, *mavros_node*-solmun tarvittavin parametrein sekä *bunker_autopilot*-paketin omat solmut ja asettaa järjestelmän toimintaa ohjaavia ROS-parametreja.

6 AUTOPILOTTISOVELLUKSEN TESTAUS JA ARVIOINTI

Autopilottisovelluksen käyttämiseen ja testaamiseen kuuluvia työvaiheita olivat mm. reitin suunnittelu maa-asemasovelluksella, mobiilirobotin valmistelu (tietoliikenneyhteyksien avaaminen, autopilottiohjaimen käyttövalmiuden varmistaminen ja ohjelmien käynnistäminen etäyhteydellä) sekä varsinainen kenttätestaus. Käytön yhteydessä testattiin toimintojen käytettävyyttä, satelliittipaikannukseen perustuvan navigoinnin luotettavuutta ja johdonmukaisuutta, ArduRover-ohjelmiston ominaisuuksia sekä muita sovelluksen käyttökelpoisuuteen liittyviä seikkoja.

6.1 Reitin suunnitteleminen ja asettaminen Mission Plannerilla

Ajoreitti suunnitellaan merkitsemällä reittipisteitä karttapohjalle Mission Plannerin Plan-suunnittelunäkymässä (KUVA 7). Tässä vaiheessa autopilottiohjaimen ei tarvitse olla kytkettynä. Sovelluksessa on myös simulaatiotila, jolla ominaisuuksien käyttöä voi kokeilla ja harjoitella ilman autopilottiohjainta. Kun autopilottiohjain on yhdistettynä tietokoneeseen esim. USB-kaapelilla tai telemetriaradion välityksellä, reitti voidaan syöttää laitteeseen suunnittelunäkymän Write-painikkeella. Laitteen muistissa oleva reitti voidaan vastaavasti lukea Mission Plannerin suunnittelunäkymään Read-toiminnolla.



KUVA 7. Reitin suunnitteleminen ja lähettäminen Mission Plannerin Plan-näkymässä

Mission Plannerissä kartalla näkyy laitteen sijainnin kohdalla ajoneuvokuvake sekä sovelluksen asetusten mukaan anturien havaitsema kompassisuunta (punainen linja), navigointisuunta seuraavaan reittipisteeseen (oranssi linja), haluttu kulkusuunta (vihreä linja) sekä GPS-sijainnin muutosten perusteella laskettu todellinen liikkumissuunta (musta linja) (ArduPilot 2024i, luku ”Overview”).

KUVA 8. Reaaliaikaiset hallinta- ja seurantatoiminnot Mission Plannerin Data-näkymässä

Data-näkymässä (KUVA 8) voi seurata karttanäkymän tietojen lisäksi monia erilaisia anturi- ja telemetriatietoja, kuten nopeutta, asentoa (keinohorisontti), etäisyyttä seuraavaan reittipisteeseen, lämpötilaa, akun jännitettä ym. Sovellus piirtää toteutuneen ajolinjan violettina viivana. Data-näkymässä voi vaihtaa autopilotin aktivointitilaa ja ohjaustilaa. ArduPilotin erityyppisiin aluksiin ja ajoneuvoihin tarkoitetut versiot eroavat varsinkin käytettävissä olevien ohjaustilojen osalta (ArduPilot 2024u). Saatavilla on monia varsin erikoistuneitakin ohjaustiloja, mutta tämän sovelluksen ja sen jatkokehitysmahdollisuuksien kannalta merkityksellisiä ovat seuraavat:

- MANUAL – manuaalinen ohjaus
- AUTO – automaattinen navigointi ohjelmoidulla reitillä
- DOCK – ohjaus kohti kiinteää telakointiasemaa (tarvitaan kameramoduuli)
- GUIDED – siirtyminen annettuun paikalliskoordinaattisijaintiin, esim. Mission Plannerin Fly To Here -toiminnolla
- FOLLOW – toisen sijaintitietoa lähettävän ajoneuvon/autopilotin seuraaminen määräetäisyydellä, ks. kohta 6.7

- RTL (Return To Launch) – paluu aloituspaikkaan (*home*) eli reittipistelistan pisteeseen 0
- SMART RTL – paluu aloituspaikkaan ns. turvallista reittiä eli MANUAL-tilassa ajon aikana tallennettuja sijainteja noudatellen.

6.2 Mobiilirobotin valmistelu automaattiajoa varten

Ennen GPS-paikannukseen perustuvan automaattiajon aloittamista käynnistetään ja valmistellaan Bunkerin laitteisto eli avataan etäyhteys Bunkerin tietokoneeseen sekä CAN-yhteys ohjausyksikköön ja käynnistetään tarvittavat ROS-ohjelmat käynnistystiedostosta (ks. kohta 5.7.6). Reittipisteet asetetaan joko maa-asemasovelluksessa, josta reitti lähetetään autopilottiin telemetriaradion tai USB-liitännän kautta, tai manuaalisesti ajamalla ja käyttämällä ROS-pohjaisia reittipistetoimintoja radio-ohjaimen kytkimillä. Jos autopilotti on toimintavalmis eli kalibroinnit on tehty, inertia-anturiyksikkö (IMU) on saavuttanut käyttölämpötilan ja satelliittiyhteys on muodostettu (GPS-fix), autopilotin aktivointitila voidaan vaihtaa (radio-ohjaimen kytkimellä SWD) tilaan ARMED. Kun ohjaustila vaihdetaan (kytkimellä SWC) tilasta MANUAL tilaan AUTO, ajoneuvo lähtee liikkeelle. Kun viimeinen reittipiste saavutetaan, autopilotti antaa äänimerkin ja pysäyttää ajoneuvon. Jos halutaan ajaa sama reitti uudestaan, vaihdetaan ohjaustila MANUAL-tilaan ja uudelleen AUTO-tilaan.

6.3 Kenttätestaus ja paikannuksen ja navigoinnin luotettavuuden arviointi

Pääosa kenttätestauksesta tehtiin ruohopintaisella, tasapohjaisella piha-alueella (KUVA 9), jonka leveys oli noin 20 m ja pituus noin 40 m. Alueella ei ollut suuria pinnanvaihteluja eikä merkittävästi puustoa, mutta osa testeistä tehtiin rakennusten läheisyydessä. Tavoitteena oli kokeilla, kuinka johdonmukaisesti autopilotin ohjaama ajoneuvo pystyy seuraamaan pensaiden rajaamaa reittiä, jonka leveys on n. 1,5–3 m, ja kuinka johdonmukaisia paikannustuloksia GPS-järjestelmä tuottaa. Ohjelmiston kehitystyö tapahtui samanaikaisesti testaamisen kanssa, joten testaamisen toinen päätavoite oli tukea kehitystyötä käytännön havaintojen kautta. Järjestelmän tilaa seurattiin pääasiassa kannettavassa Windows-tietokoneessa käytettävällä Mission Planner -maa-asemasovelluksella siten, että tietokone oli yhteydessä autopilottiin telemetriaradioiden välityksellä.

KUVA 9. AgileX Bunker automaattiajon testireitillä

Järjestelmän täyden toimintakyvyn saavuttamiseen kuluva aika laitteiston käynnistämisen jälkeen vaihteli suuresti. Pixhawk 2 Cubessa on lämpötilavakioitu, sisäisellä lämmitysvastuksella varustettu inertia-anturiyksikkö (IMU), jonka on oletusparametreilla saavutettava +40 °C:n lämpötila ennen käyttöä. Lisäksi GPS-moduulin on saatava signaali paikannussatelliiteista sijainnin määrittämistä varten, ja sähköisen kompassin on määritettävä luotettava suuntima. Jos jokin näistä aloitustoimenpiteistä on kesken, Mission Plannerin tilaikkunassa näkyy virheilmoitus eikä autopilottia voi aktivoida ARMED-tilaan. Eräissä tilanteissa laitteen voi tosin asettaa pakotetusti ARMED-tilaan (”Forced arming”), mutta tällöin paikannuksen luotettavuus on usein tavallista heikompi ja sijainti- tai asentotieto voi vaikuttaa epävakaalta. Jos autopilotin eri inertia- ja kompassianturien lukemat eivät vastaa toisiaan, tästä tulee virheilmoitus, esim. ”Inconsistent yaw rate” (*epäjohdonmukainen vaappumiskulma*). Tällöin kompassitietojen mukainen suunnan muutos ei vastaa kiihtyvyysanturien perusteella tulkittua pystyakselin suuntaista kiertymää. Virheilmoitukset usein poistuvat, kun ajoneuvoa liikutellaan riittävästi.

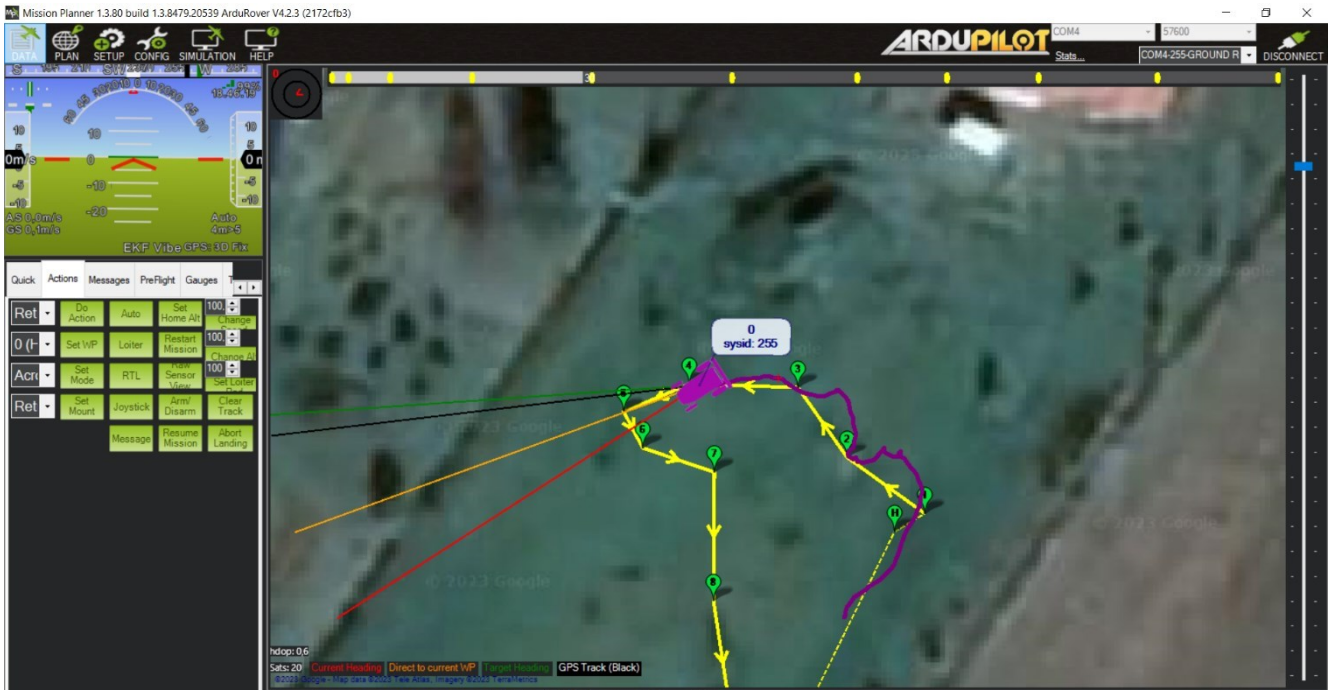
Aika ajoin laitteisto antoi käynnistyksen yhteydessä ilmoituksen, jonka mukaan kompassi tai kiihtyvyysanturit oli kalibroitava uudelleen. Mission Plannerissa on kalibrointia varten toiminnot, mutta kiihtyvyysanturien kalibroinnin yhteydessä autopilottiohjainta pitäisi käännellä eri asentoihin, mm. kyljelleen ja nokalleen, eikä tätä ole mahdollista tehdä, jos autopilottilaitteisto on kiinteästi asennettuna isokokoiseen ajoneuvoon. Sovelluksessa on myös isoille ajoneuvoille tarkoitettu yksinkertaistettu

kompassin kalibrointitoiminto, jossa käytännössä syötetään laitteen ilmoittaman suunnan ja todellisen kompassisuunnan välistä erotusta vastaava korjausvakio. Tällä toiminnolla saadaan aikaan jonkin verran hienosäätöä, mutta kunnollista kalibrointia varten autopilottiohjain ja GPS-moduuli oli irrotettava ajoneuvosta ja kytkettävä erilliseen pieneen akkuun, jotta laitteistoa voi käännellä vapaasti.

Kompassin osalta kalibroinnin hyöty oli rajallinen; oletettavasti mobiilirobotin sähkömoottorit, virtajohtimet ja akku aiheuttivat magneettikentän häiriöitä, jotka vääristivät kompassin lukemia. Kalibroinnin jälkeenkin kompassin ilmoittama suunta oli melko johdonmukaisesti noin 20 astetta vastapäivään todellisesta suunnasta, kun laite oli paikallaan. Laitteen lähtiessä liikkeelle virhe saattoi voimistua entisestään, mutta tätä oli vaikea tulkita silmämääräisesti telemetriatietoja seuraamalla. Perusteellisempi lokitietojen keruu olisi mahdollisesti tuonut ilmi sen, voimistuiko kompassivirhe oleellisesti juuri liikkeellelähdön hetkellä. Virhe näkyi käytännössä niin, että automaattiajon alkaessa Bunker kääntyi ensin oikeasta navigointisuunnasta voimakkaasti oikealle päin, usein noin 45 astetta tai enemmänkin, mutta korjasi pian kulkusuuntaansa. Mahdollisesti autopilotti kalibroi kompassidataa reaaliaikaisesti kiihtyvyysanturien ja GPS-sijainnin muutoksen perusteella. Joskus kompassin suuntatieto oli edelleen virheellinen liikkeen aikana, ja tällöin laite näytti Mission Plannerin karttanäkymässä liikkuvan osittain sivuttain – samaan tapaan kuin sivuvirtauksessa etenevä vene. Käytön myötä autopilotti korjasi suuntatietoa niin, että kartalla näkyvä kulkusuunta vastasi navigointisuuntaa ja laitteen todellista etenemissuuntaa maastossa. Jos reittipisteen kohdalla oli tarpeen tehdä vain loiva käännös, laite muutti usein sujuvasti suuntaansa, mutta paikallaan tehtävien terävien käännösten kohdalla ilmeni jälleen samantapainen suuntavirhe ajoneuvon lähtiessä käännöksen jälkeen uudelleen liikkeelle.

Kuvassa 10 näkyy kiemurteleva ajolinja reittipisteiden 1 ja 3 välillä. Reittipisteen 3 jälkeen ajolinja stabiloituu reittipistevälillä 3–5, mutta testiä jatkettaessa terävä käännös reittipisteessä 5 aiheutti jälleen suuntavirhettä. Pitkillä suorilla reittiosuuksilla suuntatarkkuus oli yleensä selvästi parempi kuin mutkittelevilla osuuksilla.

Kompassivirheen yhteyttä autopilottilaitteiston sijoituspaikkaan testattiin siirtämällä asennuspaikkaa puisen telineen avulla ylemmäksi sekä irrottamalla laitteisto kokonaan ajoneuvosta ja vertaamalla kompassin ilmoittamaa suuntaa Maanmittauslaitoksen karttapalvelusta (Maanmittauslaitos 2024a) tarkistettuun todelliseen suuntaan, jota verrattiin maastossa oleviin kiinteisiin kohteisiin (TAULUKKO 5). Ajon aikana teline notkui hieman, eikä navigointikäyttäytymistä voitu arvioida luotettavasti eri sijoitteluvaihtoehdoissa.



KUVA 10. Kompassivirheestä johtuvaa ajolinjan kiemurtelua

TAULUKKO 5. Kompassivirheen arvojen vertailu autopilotin eri sijoitteluvaihtoehdoissa

Autopilotin sijoituspaikka	Suuntavirhe z-akselin suhteen
alkuperäinen asennuspaikka	-15°...-25° (runsaasti vaihtelua, tyypillinen virhe -23°)
telineessä 14 cm korkeammalla	-18°...-23° (vähemmän vaihtelua)
telineessä 52 cm korkeammalla	+7°...+8°
irrotettuna ajoneuvosta	0°...+5°

GPS-paikannuksen tarkkuus saman käyttökerran aikana suoritettujen reittipisteiden kesken oli yleensä noin ±1 m tai hieman parempikin: kun laitteella ajettiin samaa reittiä useita kertoja peräkkäin, ajourien keskikohdat sijoittuivat niin, että vaihtelun ääripäät olivat tyypillisesti alle 1,5–2 metrin etäisyydellä toisistaan. Toisinaan koko koordinaatisto näytti siirtyvän hiljalleen niin, että toteutunut reitti siirtyi tiettyyn suuntaan, varsinkin pohjois-eteläsuunnassa. Usein jos samaa tallennettua reittiä ajettiin seuraavana päivänä uudelleen, systemaattinen sijaintivirhe saattoi olla jopa 3 m pohjoiseen tai etelään. Näissä tilanteissa oli apua sovellukseen kehitetystä OFFSET_MISSION-toiminnosta, jolla koordinaattivirhe voitiin korjata siirtämällä reitin kaikkia reittipisteitä tietty metrimäärä haluttuun suuntaan.

Sään vaikutuksesta GPS-paikannuksen tarkkuuteen ei kertynyt riittävästi aineistoa systemaattista arviointia varten, mutta niitä tilanteita, joissa paikannustarkkuus oli heikoimmillaan ja koordinaatisto vaikutti siirtyvän, vaikutti ilmenevän usein pilvisellä säällä. Tällöin reittipisteiden keskinäiset sijainnit saattoivat vaihdella tavallista enemmän, mutta suurempi ongelma oli koko koordinaatiston siirtyminen pois paikaltaan. Vastaavaa ongelmaa ilmeni myös silloin, kun ajoneuvoa käytettiin rakennusten lähellä. Ilmiö johtunee satelliiteista saapuvan signaalin vääristymisestä esim. ilmakehän tuottaman troposfäärirefraktion (Poutanen 2016, 193–196) ja maastoesteistä johtuvan monitieheijastumisen vuoksi (Poutanen 2016, 199–200) sekä siitä, että rakennukset ym. esteet haittaavat satelliittien näkyvyyttä.

6.4 Ajokäyttäytymistä ohjaavien parametrien säätäminen

ArduRoverissa on ohjauksen ja moottorien nopeuden säätöä varten PID-säätimet, joilla voidaan säätää suunnanmuutosten herkkyyttä ja vakautta. Bunkerin tapauksessa näitä parametreja ei tarvinnut muuttaa, vaan ajoneuvo liikkui oletusarvoilla tyydyttävästi. Ajoneuvon iso koko saattoi osaltaan vaikuttaa myönteisesti suuntavakauteen. Varsinkin Ackermann-ohjauksella varustetussa ajoneuvossa väärät PID-arvot voivat aiheuttaa riittämätöntä ohjausliikettä tai häiritsevää ohjauksen heiluriliikettä. ArduRoverin dokumentaatiossa on ohjeet PID-arvojen säätämiseen (ArduPilot 2024w).

Testauksen alkuvaiheessa reitillä pysymisen kannalta oli ongelmallista se, että oletusparametreilla laite kulkee ensin suoraan kohti kutakin reittipistettä ja aloittaa vasta sitten kääntymisen kohti seuraavaa reittipistettä. Tämän vuoksi ajoneuvo usein ajautui jyrkissä käännöksissä ulos halutulta ajoradalta. Kääntösädettä ja ajonopeutta reittipisteen kohdalla voi säätää parametreilla, mutta näillä säädöillä oli alkuperäiseen ongelmaan vain rajallinen vaikutus. Maa-ajoneuvoille tarkoitetussa ArduRoverissa, toisin kuin lentäville laitteille tarkoitetuissa ArduCopterissa ja ArduPlanessa, ei ole tällä hetkellä mahdollista asettaa reittipisteitä, joissa ajoneuvo ennakoi käännöstä siirtymällä liikeradallaan ensin hieman ulkomutkan suuntaan ja kaartaa sitten loivasti reittipisteen läpi (*spline waypoint*) (ArduPilot 2024p, luku ”Right Click Menu”). Ratkaisuksi löytyi parametri WP_PIVOT_ANGLE, jota säätämällä Bunker saatiin telaohjausta hyödyntäen kääntymään jyrkässä käännöksessä paikallaan ja jatkamaan sitten suoraan kohti seuraavaa reittipistettä. Parametrilla asetetaan käytännössä kääntymiskulman kynnysarvo siten, että tätä jyrkemmässä käännöksessä ajoneuvo pysähtyy ja kääntyy paikallaan sen sijaan, että se ajaisi pysähtymättä reittipisteen läpi ja kääntyisi sen jälkeen loivasti kaartaen. Parametrin oletusarvo oli 90 (arvo asteina suhteessa alkuperäiseen liikesuuntaan), ja arvolla 30 saatiin varsin sujuva ajokäytös erilaisia käännöksiä sisältävällä reitillä. Myös kääntösäteen metreinä ilmaisevaa parametria

TURN_RADIUS säädettiin oletusarvosta 1 arvoon 0,4. Näin kääntyminen oli riittävän jyrkkä myös loivemmissa mutkissa, joiden kohdalla ajoneuvo ei kääntynyt paikallaan vaan eteni kaartaen. Paikallaan tehtävien käännösten haittapuolena oli nurmikon kuluminen toistuvien käännösten kohdalla ja kostealla säällä mutapaakkujen kertyminen telastoon. Joissakin käyttöolosuhteissa, kuten herkällä alustalla tai jos telastoon kertyvän maa-aineksen ajatellaan aiheuttavan ongelmia, parametreja olisi säädettävä vähemmän aggressiivisiksi ja käännökset toteutettava vaiheittain lisäämällä ylimääräisiä reittipisteitä.

6.5 Radiokauko-ohjauksen rooli ja rajoitteet

Testauspaikan luonteen vuoksi oli tarve saada Bunker pysymään noin parin metrin levyisellä reitillä ja välttää esteisiin törmäämistä. Varsinkin edellä mainittujen kompassisuunnan virheiden ja myös GPS-paikannuksen vaihtelevan tarkkuuden vuoksi ajoneuvon etenemiseen oli reitin ahtaissa paikoissa puututtava manuaalisesti. Jos ajoneuvo oli automaattiajon aikana törmäämässä esteeseen, nopein keino puuttua tilanteeseen oli kytkeä radio-ohjaimesta Bunkerin ohjaustila CAN-ohjauksesta RC-ohjaukseen, ohjata laite radio-ohjaimella takaisin reitille ja kytkeä sitten laite takaisin CAN-ohjaustilaan. Tämä ei keskeyttänyt autopilotin reitin suorittamista, vaan ajo jatkui samasta kohdasta. Jos tällaista ohjausmahdollisuutta ei olisi, laitteen voisi pysäyttää myös kytkemällä autopilotin ohjaustilan AUTO-tilasta MANUAL-tilaan tai aktivointitilan ARMED-tilasta DISARMED-tilaan. Käyttäjän oli oltava tarkkana myös tilanteissa, joissa reitin suoritus aloitettiin (kytkemällä autopilotti AUTO-ohjaustilaan), kun ensimmäinen reittipiste oli ajoneuvon takana. Ajoneuvon peruuttamisen mahdollistavaa parametria ei ollut asetettu, joten laite lähti aluksi suoraan eteenpäin eli poispäin ensimmäisestä reittipisteestä ja pyrki korjaamaan liikerataansa laajassa kaaressa kohti määränpäätä.

Käyttäjän puuttumistarpeen vuoksi oli tärkeää pitää ajoneuvo radio-ohjaimen kantaman alueella. Bunkerissa radiovastaanotin on sijoitettu roiskesuojattuun metallikoteloon lähelle mm. moottoreita ja muita laitteita. Rungossa on liitäntä radiovastaanottimen lisäantennia varten, mutta kenttätestien aikana lisäantenni ei ollut käytössä. Näin käytettynä radio-ohjaimen kantama oli parhaimmillaan 65 metriä, kun ohjainta pidettiin mahdollisimman korkealla ja Bunkerin peräpää oli kohti käyttäjää. Heikoimmillaan kantama oli noin 22 metriä, kun ohjainta pidettiin lähellä maanpintaa ja Bunkerin jompikumpi etukulma oli kohti käyttäjää siten, että suuri osa telastosta oli signaalin tiellä.

Bunkeriin on toteutettu vikasietotoiminto, joka pysäyttää laitteen liikkeen, kun radio-ohjaimen signaali katkeaa. Tällöin CAN-sanomakehyksessä 241 näkyy vastaava vikakoodi, ja tieto näkyy myös ROS-kanavan */bunker_status* viestikentissä *base_state* ja *fault_code*. Lisäksi radio-ohjaimen kanavien dataa sisältävät CAN-kehyksen tavut saavat arvon 0, mikä vastaa ohjaussauvojen neutraalia asentoa. Valitettavasti vikasietotoiminto toimii ainoastaan manuaalisessa RC-ohjaustilassa. Kun laite on CAN-ohjauksessa ja radio-ohjaimen signaali katkeaa, järjestelmä ei anna vikakoodia ja CAN-kehyksessä välitettävät radiokanavien tiedot jäävät näyttämään samoja arvoja, joita radio-ohjain lähetti ennen signaalin katkeamista. Siten sovellusohjelmistolle ei saada tietoa vikatilanteesta. Tämän puutteen vuoksi käyttäjän on automaattiajon aikana pysyteltävä riittävän lähellä ajoneuvoa, jotta laite voidaan tarvittaessa pysäyttää manuaalisesti.

6.6 Odom-paikkatiedon ja muiden navigointitietojen hyödyntäminen

Mavros_node-solmu julkaisee GPS-koordinaattien lisäksi paikallisia sijaintitietoja, jotka perustuvat kiihtyvyysanturien tuottamaan dataan. Nämä tiedot eivät ole yhtä tarkkoja kuin GPS-paikkatiedot, mutta niiden avulla autopilotilla on mahdollista suorittaa automaattisia ohjaustoimintoja myös ilman satelliittipaikannusta. Kanavalla */mavros/local_position/odom* näkyy ajoneuvon x- ja y-suuntainen sijainti metreinä käynnistämispaikasta siten, että y-akseli osoittaa pohjoiseen. Tietojen tarkkuutta testattiin lyhyesti seuraavasti: autopilotti käynnistettiin ja ajoneuvo ajettiin manuaalisesti kohtaan, joka odom-viestien mukaan sijaitsi kohdassa (x: −13,98; y: −26,6), eli noin 14 metriä itään ja 27 metriä etelään alkupisteestä. Pythagoran lauseen mukaisesti etäisyys lähtöpisteestä oli siten 30,05 m. Mittanauhalla mitattu todellinen siirtymä oli 29,1 m, eli poikkeama oli noin 3,3 %. Odom-paikkatietoja voi hyödyntää asettamalla autopilotin GUIDED-tilaan ja syöttämällä halutun sijainnin metrimääräisessä koordinaattimuodossa kanavalle */mavros/setpoint_position/local*, jota *mavros_node*-solmu kuuntelee. Tällöin autopilotti ajaa ajoneuvon kyseiseen xy-sijaintiin. Koska tämä sijaintitieto perustuu kiihtyvyysanturien mittaamaan liikkeeseen ilman kiinteää referenssipistettä, paikannustarkkuus on aluksi melko hyvä mutta heikkenee kumulatiivisesti sitä mukaa kuin ajoneuvoa liikutellaan. Laitetta ohjattiin tällä tavalla useita kertoja eri kohteisiin pienellä alueella, ja noin kymmenen toiston jälkeen toteutunut ajokohde poikkesi aiotusta sijainnista jo yli kaksi metriä. Tämä paikannusmenetelmä on siis rajallisesti käyttökelpoinen apumenetelmä, esim. jos mobiilirobotilla on tarkoitus ajaa tilapäisesti sisätiloissa tai satelliittisignaalin katvealueen läpi.

Mavros_node-solmu julkaisee varsin suuren määrän autopilotin välittämiä tietoja eri kanavilla. Julkaistavia kanavia on yhteensä 80, mutta tässä sovelluksessa osa näistä ei sisältänyt tietoja. Julkaistavia tietoja ovat mm. ajoneuvon koordinaatit, kompassisuunta ja nopeus, seuraavan reittipisteen koordinaatit, kiihtyvyysanturin tiedot eli x-, y- ja z-akselin suuntaiset lineaariset ja kulmakiihtyvyydet, akun jännite ja autopilotin lämpötila. Navigointitiedoille, kuten seuraavan reittipisteen suuntimalle, etäisyydelle ja tavoiteltavalle kiihtyvyydelle, olisi oma kohtansa kanavalla */mavros/setpoint_raw/target_attitude*, mutta ArduRoverissa nämä viestit eivät ole käytettävissä. MAVROSin dokumentaation mukaan tämä viestikokonaisuus on toteutettu vain PX4-laiteohjelmistoa varten (ROS 2018a).

6.7 Seuraamistoiminto

ArduRover-ohjelmistoon sisältyy erityinen FOLLOW-ohjaustila, jossa ajoneuvon on tarkoitus seurata GPS-paikkatietoa lähettävää kohdetta pysytellen samalla etäisyydellä ja samassa suunnassa kohteeseen nähden. Se siis pyrkii säilyttämään koko ajan saman poikkeaman (offset) seurattavan kohteen sijaintiin nähden. Poikkeama voidaan kytkeä joko absoluuttiseen koordinaatistoon (NED – North, East, Down) tai suhteelliseen, kohteen mukana liikkuvaan koordinaatistoon, jolloin poikkeama katsotaan suhteessa kohteen suuntimaan. (ArduPilot 2024j.) Koe suoritettiin yhdistämällä toinen telemetriaradiolla varustettu Pixhawk-laite samaan maa-asemaan. Seurattavaa laitetta ei ollut asennettu mihinkään ajoneuvoon, vaan sitä pidettiin kädessä. Ennen FOLLOW-tilan käyttämistä oli muutettava FOLL_ENABLE-parametrin arvoksi 1 ja käynnistettävä autopilotti uudelleen. Lisäksi molemmilla autopiloteilla piti olla sama SYSID_MYGCS-parametrin arvo mutta eri SYSID_THISMAV-arvo. Koe tehtiin käyttäen absoluuttista koordinaatistoa, ja FOLLOW-tila kytkettiin, kun Bunker oli noin 1,5 metrin päässä seurattavasta laitteesta sen länsipuolella. Kun seurattavan laitteen kanssa kuljettiin testialueella, Bunker liikkui nykäyksittäin ja oikeaa suuntaa etsien mutta hakeutui seurattavan laitteen siirtämisen jälkeen aina suunnilleen oikeaan paikkaan.

Tässä kokeessa seurattavan laitteen ja Bunkerin GPS-sijainnin epätarkkuudet saattoivat jossain määrin kertautua, ja sen vuoksi kohta, johon Bunker lopulta asettui suhteessa seurattavaan laitteeseen, vaihteli havaittavasti. Liikkumisen sujuvuutta olisi mahdollisesti voinut parannella parametreja (esim. ohjauskiihtyvyyden ”aggressiivisuutta” säätelevää FOLLOW_POS_P-parametria) säätämällä, ja johdonmukaisemman tuloksen olisi todennäköisesti saanut toistamalla kokeen laajemmalla alueella. FOLLOW-tilan ominaisuuksia ei ollut aikaa testata kovin pitkällisesti, joten tässä yhteydessä tyydyttiin toteamaan, että seuraamistoiminto toimii ainakin pääosin odotusten mukaisesti.

7 YHTEENVETO

Tässä opinnäytetyössä saatiin aikaan ratkaisu, joka täytti alkuvaiheessa asetetut tavoitteet, eli AgileX Bunker -mobiilirobotti saatiin ajamaan GPS-paikannukseen perustuvia reittejä automaattisesti autopilotin ohjaamana. Radio-ohjaimen kytkimiä pystyttiin hyödyntämään sovelluksen perustoimintojen hallinnassa, ja lisäksi kehitettiin ajoreittiä muokkaavia lisätoimintoja, joissa hyödynnettiin ROS-järjestelmän ja MAVROS-paketin mahdollisuuksia.

Pixhawk-autopilottiohjain ja siihen asennettu ArduRover-ohjelmisto osoittautuivat toimivaksi toteutusratkaisuksi, ja Mission Planner -maa-asemasovellus tarjosi havainnollisen hallintakäyttöliittymän autopilotin toimintoihin. Alkuvaiheessa vaikein osuus oli löytää keino tuoda autopilotin ohjaussyöte ROS-järjestelmään Bunkerin liikekomentoja varten, ja tässä MAVROS oli tärkeä väline. MAVROSin ominaisuuksiin perehtyminen edellytti jonkin verran selvittelytyötä, mutta mahdollisuus hyödyntää ROS-viestejä ja -palveluja autopilotin hallitsemisessa helpotti lopulta sovelluksen kokonaisuuden rakentamista. Myös CAN-väylään välitettävien radio-ohjaimen tietojen saattaminen ROS-järjestelmään edellytti seikkaperäistä perehtymistä laitteessa ennestään olevaan ohjelmistoon ja erinäisiä muutoksia lähdekoodiin. Projektissa kehitettyjä automaattiohjauksen ohjelmakomponentteja voidaan kohtuullisin muutoksin soveltaa muunkin tyyppisiin ajoneuvoihin. Kehitin opinnäytetyön ohella omaan käyttöön radio-ohjattavaan autoon perustuvaa järjestelmää, jossa oli autopilottiohjaimena toinen Pixhawk-malli ja jossa hyödynnettiin samoja Python-ohjelmiston pääkomponentteja. Tässä sovelluksessa radiovastaanotin, moottorinohjain ja ohjausservo oli yhdistetty suoraan autopilottiohjaimeen. Modulaarisuutta (ks. kohta 5.7) sekä Ackermann- ja telaohjauksen eroja (ks. kohdat 5.7.3 ja 6.4) koskevat huomiot perustuvat tässä kokeilussa saatuihin kokemuksiin. Raporttia kirjoittaessani olen pyrkinyt siihen, että tästä työstä saadut tulokset, havainnot ja teoreettiset huomiot olisivat hyödynnettävissä mahdollisimman laajasti erilaisissa autopilottiin perustuvissa mobiilirobottien ohjausratkaisuissa.

Automatisoitu liikkuminen mahdollistaa ajoneuvon automatisoidut ajoreitit maastossa esim. erilaisia mittauksia, hyötykuorman liikuttelua tai muita robotin suoritettaviksi soveltuvia toimenpiteitä varten. Tätä ominaisuutta voidaan hyödyntää moniin erilaisiin tarpeisiin Centrian TKI-toiminnassa ja mahdollisissa loppusovelluksissa. Kaiken kaikkiaan työn tuloksena syntynyt järjestelmä vaikutti soveltuvalta ja jatkokehityskelpoiselta ratkaisulta satelliittipaikannukseen perustuvaksi automaattiohjaussovellukseksi.

Suurimmat automaattiajon käyttökelpoisuutta rajoittavat ongelmat olivat liikkeellelähdössä ilmenevä kompassivirhe sekä GPS-paikannuksen tarkkuus, joka ei kaikissa tilanteissa riittänyt määritetyllä reitillä pysymiseen. Paikannustarkkuutta olisi mahdollista parantaa ottamalla käyttöön reaaliaikainen kinemaattinen paikannusmenetelmä (RTK), jolloin laitteen sijainti voitaisiin määrittää luotettavasti jopa vain parin senttimetrin tarkkuudella (Poutanen 2016, 16). Kompassivirheen osalta hypoteesina on, että Bunkerin liikkeelle lähtiessä sähkömoottorit tai akulta tulevien sähköjohtimien suuri virta tuottavat kiihdytysvaiheessa tilapäisen vääristymän kompassin havaitsemaan magneettikenttään. Ongelmaa voisi yrittää todentaa tallentamalla ja tutkimalla anturidataa, ja vääristymää voisi mahdollisesti vähentää laskemalla tarvittavat oikaisuparametrit reaaliaikaisesti. Myös RTK-paikannusta voidaan hyödyntää luotettavan kompassisuunnan määrittämisessä (ArduPilot 2024k).

Bunkeriin asennettua lidaria voisi hyödyntää ajoreitillä olevien esteiden tunnistamisessa ja automaattisessa väistämisessä (ArduPilot 2024v), ja automaattiajoa voisi hyödyntää lidar-kartoitussovelluksissa. Kohdassa 5.7.5 mainittua graafista käyttöliittymää voisi käyttää Bunkeriin asennetulta näytöltä, tai sen pohjalta voisi kehittää pienemmän apunäyttölaitteen tai esim. mobiilisovelluksen, jonka kautta käyttäjä voisi seurata kaikkia reittiin liittyviä tietoja ja hallita toimintoja ajon aikana.

Radio-ohjaimen kantamaa voidaan parantaa lisäantennia käyttämällä, mutta CAN-ohjaustilassa ajoneuvon pysäyttävän vikasuojatoiminnon puuttuminen on edelleen jonkinasteinen riskitekijä. Ongelmaa voisi yrittää ratkaista Bunkerin ohjausyksikön laiteohjelmiston päivityksellä, jos sellainen tulee saataville, tai sijoittamalla radiovastaanottimen signaalijohtoon haaroitusjohdon, josta vastaanottimen raakadataa voidaan lukea esim. ROS-järjestelmään yhdistetyllä mikrokontrollerilla. Näin poikkeava radiosignaali voitaisiin havaita Bunkerin ohjaustilasta riippumatta.

Ammatillisen osaamisen ja oppimisen kannalta opinnäytetyöprojekti on ollut erittäin antoisa. Kokemukseni Python-ohjelmoinnista oli ennestään melko rajallinen, joten tämän työn yhteydessä pääsin laajentamaan osaamistani huomattavasti. Ennen työhön ryhtymistä en ollut kuullutkaan ROS-järjestelmästä enkä ArduPilot-ohjelmistosta, mutta sopivan ratkaisumallin ja sovelluksen kehittelyä ja monenlaista ongelmanratkaisua varten järjestelmien ominaisuuksiin oli perehdyttävä melko monipuolisesti. Perusasioiden kunnollinen ymmärtäminen edellytti laaja-alaista tietojen hankkimista, ja kirjoitustyössä suurimpana vaikeutena oli hankitun tiedon valikoiminen ja aiheen rajaaminen. Aihepiirissä on vielä paljon opittavaa, mutta jo tämän työn yhteydessä saatu osaaminen antaa hyvät valmiudet erilaisten robotiikkasovellusten parissa työskentelyyn.

LÄHTEET

Aaltonen, E. 2023a. Bunker_autopilot. Saatavissa: https://github.com/e-aaltonen/bunker_autopilot. Viitattu 9.11.2023.

Aaltonen, E. 2023b. Bunker_ros_RC. Saatavissa: https://github.com/e-aaltonen/bunker_ros_RC/tree/old. Viitattu 30.3.2024.

Aaltonen, E. 2023c. Ugv_sdk_RC. Saatavissa: https://github.com/e-aaltonen/ugv_sdk_RC. Viitattu 9.11.2023.

AgileX Robotics. 2020. BUNKER User Manual. V.2.0.1. Saatavissa: https://www.generationrobots.com/media/agilex/bunker-user-manual-agilex.pdf. Viitattu 31.3.2024.

AgileX Robotics. 2021a. BUNKERpro User Manual. V.1.0.0. Saatavissa: https://static.generation-robots.com/media/user-manual-agilex-robotics-bunker-pro-en.pdf. Viitattu 31.3.2024.

AgileX Robotics. 2021b. ROS Packages for Bunker Mobile Robot. Saatavissa: https://github.com/agilexrobotics/bunker_ros. Viitattu 9.11.2023.

AgileX Robotics. 2024. AgileX Bunkerin esittelymateriaali AgileX Roboticsin verkkosivuilla. Saatavissa: https://global.agilex.ai/chassis/4. Viitattu 29.3.2024.

ArduPilot. 2024a. Accelerometer Calibration. Saatavissa: https://ardupilot.org/rover/docs/common-accelerometer-calibration.html. Viitattu 17.3.2024.

ArduPilot. 2024b. ArduPilot Documentation. Saatavissa: https://ardupilot.org/ardupilot/. Viitattu 7.2.2024.

ArduPilot. 2024c. Arming / Disarming. Saatavissa: https://ardupilot.org/rover/docs/arming-your-rover.html. Viitattu 24.3.2024.

ArduPilot. 2024d. Auxiliary Functions. Saatavissa: https://ardupilot.org/rover/docs/common-auxiliary-functions.html. Viitattu 24.3.2024.

ArduPilot. 2024e. Bluetooth Telemetry Radio. Saatavissa: https://ardupilot.org/copter/docs/common-mission-planner-bluetooth-connectivity.html. Viitattu 7.2.2024.

ArduPilot. 2024f. Compass Calibration. Saatavissa: https://ardupilot.org/rover/docs/common-compass-calibration-in-mission-planner.html. Viitattu 17.3.2024.

ArduPilot. 2024g. Complete Parameter List. Saatavissa: https://ardupilot.org/rover/docs/parameters.html. Viitattu 24.3.2024.

ArduPilot. 2024h. Configuring a Telemetry Radio using Mission Planner. Saatavissa: https://ardupilot.org/copter/docs/common-configuring-a-telemetry-radio-using-mission-planner.html. Viitattu 4.2.2024.

ArduPilot. 2024i. Flight Data Screen Overview. Saatavissa: https://ardupilot.org/planner/docs/mission-planner-ground-control-station.html. Viitattu 6.4.2024

ArduPilot. 2024j. Follow Mode. Saatavissa: https://ardupilot.org/rover/docs/follow-mode.html. Viitattu 4.2.2024.

ArduPilot. 2024k. GPS for Yaw (aka Moving Baseline). Saatavissa: https://ardupilot.org/copter/docs/common-gps-for-yaw.html. Viitattu 6.4.2024.

ArduPilot. 2024l. Installing ROS. Saatavissa: https://ArduPilot.org/dev/docs/ros-install.html. Viitattu 8.1.2024.

ArduPilot. 2024m. Loading Firmware. Saatavissa: https://ardupilot.org/rover/docs/common-loading-firmware-onto-pixhawk.html. Viitattu 4.2.2024.

ArduPilot. 2024n. Magnetic Interference. Saatavissa: https://ardupilot.org/rover/docs/common-magnetic-interference.html. Viitattu 6.4.2024

ArduPilot. 2024o. MAVLink Basics. Saatavissa: https://ardupilot.org/dev/docs/mavlink-basics.html. Viitattu 18.3.2024.

ArduPilot. 2024p. Mission Planner Fight PLAN. Saatavissa: https://ardupilot.org/planner/docs/mission-planner-flight-plan.html. Viitattu 4.2.2024.

ArduPilot. 2024q. Mission Planner Home. Saatavissa: https://ardupilot.org/planner/index.html. Viitattu 4.2.2024.

ArduPilot. 2024r. Pre-Arm Safety Checks. Saatavissa: https://ardupilot.org/rover/docs/common-prearm-safety-checks.html Viitattu 18.3.2024.

ArduPilot. 2024s. Radio Control Systems. Saatavissa: https://ardupilot.org/rover/docs/common-rc-systems.html. Viitattu 18.3.2024.

ArduPilot. 2024t. Rover Autopilot System Assembly Instructions. Saatavissa: https://ardupilot.org/rover/docs/rover-autopilot-assembly-instructions.html. Viitattu 8.1.2024.

ArduPilot. 2024u. Rover Control Modes. Saatavissa: https://ardupilot.org/rover/docs/rover-control-modes.html. Viitattu 24.3.2024.

ArduPilot. 2024v. Simple Object Avoidance. Saatavissa: https://ardupilot.org/rover/docs/common-simple-object-avoidance.html. Viitattu 6.4.2024

ArduPilot. 2024w. Tuning Turn Rate. Saatavissa: https://ardupilot.org/rover/docs/rover-tuning-steering-rate.html. Viitattu 4.2.2024.

ArduPilot. 2024x. Vehicle Types Supported by ArduPilot. Saatavissa: https://ardupilot.org/ardupilot/docs/common-all-vehicle-types.html. Viitattu 17.3.2024.

Drone Dojo. 2022. ArduPilot VS PX4 | Which Is Better? Saatavissa: https://dojofordrones.com/ardupilot-vs-px4/. Viitattu 17.3.2024.

Flysky Technology co., ltd. 2020. FS-i6S User Manual. Digital Proportional Radio Control System. Saatavissa: https://www.flysky-cn.com/s/FS-i6S-User-manual-20200628-al4y.pdf. Viitattu 24.11.2023.

Gage, D. W. 1995. UGV HISTORY 101: A Brief History of Unmanned Ground Vehicle (UGV) Development Efforts. San Diego, CA, USA: Naval Ocean Systems Center. Saatavissa: https://apps.dtic.mil/sti/tr/pdf/ADA422845.pdf. Viitattu 30.3.2024.

GetFPV. 2024. FlySky FS-A8S V2 2.4G 8CH Mini Receiver with PPM i-BUS SBUS Output. Saatavissa: https://www.getfpv.com/flysky-fs-a8s-fs-a8s-2-4g-8ch-mini-receiver-with-ppm-i-bus-sbus-output.html. Viitattu 4.2.2024.

Maanmittauslaitos. 2024a. Karttapaikka. Saatavissa: https://asiointi.maanmittauslaitos.fi/karttapaikka/. Viitattu 7.2.2024.

Maanmittauslaitos. 2024b. Satelliittipaikannus. Saatavissa: https://www.maanmittauslaitos.fi/tutkimus/teematietoa/satelliittipaikannus. Viitattu 7.2.2024.

Poutanen, M. 2016. Satelliittipaikannus. Helsinki: Tähtitieteellinen yhdistys Ursa ry.

Pyo, Y., Cho, H., Jung, R. & Lim, T. 2017. ROS Robot Programming. Soul: ROBOTIS Co., Ltd.

PX4 Dev Team. 2020. Cube Flight Controller. Saatavissa: https://web.archive.org/web/20231208205115/https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk-2.html. Viitattu 7.4.2024.

QGroundControl. 2023. QGroundControl User Guide. Saatavissa: https://docs.qgroundcontrol.com/master/en/qgc-user-guide/. Viitattu 17.3.2024.

Robot Platform. 2024. Servo Control tutorial. Saatavissa: http://www.robotplatform.com/knowledge/servo/servo_control_tutorial.html. Viitattu 18.3.2024.

ROS. 2018a. MAVROS, Package Summary. Saatavissa: http://wiki.ros.org/mavros. Viitattu 4.2.2024.

ROS. 2018b. Introduction Saatavissa: http://wiki.ros.org/ROS/Introduction. Viitattu 26.11.2023.

ROS. 2023a. Distributions. Saatavissa: http://wiki.ros.org/Distributions. Viitattu 25.11.2023.

ROS. 2023b. Ubuntu install of ROS Noetic. Saatavissa: http://wiki.ros.org/noetic/Installation/Ubuntu. Viitattu 25.11.2023.

Weston Robot. 2021. UGV SDK (v1.x). Saatavissa: https://github.com/westonrobot/ugv_sdk/tree/v1.x. Viitattu 9.11.2023.

rcout to cmd vel.py

```
#!/usr/bin/python

"""
rcout_to_cmd_vel.py
Esa Aaltonen 2023

Convert skid steering servo signal to Twist messages

This node reads FCU skid steering throttle output from /mavros/rc/out and publishes corresponding Twist
values to /cmd_vel.
Channel[0]: throttle left  (SERVO1_FUNCTION = 73)
Channel[2]: throttle right (SERVO3_FUNCTION = 74)

Requirements:
- CAN up (sudo ip link set can0 up type can bitrate 500000)
- bunker_bringup running (bunker_minimal.launch)
- MavROS node running (/mavros/mavros_node)

Default parameter values set in /autopilot_test/launch/autopilot_test_agx.launch:
- autopilot/speed_factor_lin_x: "1.0"   Forward movement rate under FCU control
- autopilot/speed_factor_ang_y: "1.0"   Turning rate under FCU control
"""

import rospy
from std_msgs.msg import UInt16
from geometry_msgs.msg import Twist
from mavros_msgs.msg import RCOut
import time

class RCOutInput():
    def __init__(self):
        rospy.init_node("rcout_to_cmd_vel")
        self.sub_rc = rospy.Subscriber("mavros/rc/out", RCOut, self.update_throttle)
        rospy.loginfo("> Subscriber for /mavros/rc/out created")

        self._pwm_min = 900
        self._pwm_max = 2100

        self._speed_factor_lin_x = 1.0
        self._speed_factor_ang_z = 1.0

        if rospy.has_param('autopilot/speed_factor_lin_x'):
            self._speed_factor_lin_x = rospy.get_param('autopilot/speed_factor_lin_x')
        if rospy.has_param('autopilot/speed_factor_ang_y'):
            self._speed_factor_ang_y = rospy.get_param('autopilot/speed_factor_ang_y')

        self.pub_twist = rospy.Publisher("cmd_vel", Twist, queue_size=1)

        self.twist_msg = Twist()

    def update_throttle(self, msg):
        self.twist_msg.linear.x = self.pwm_to_adimensional((msg.channels[0] + msg.channels[2]) * 0.5) *
self._speed_factor_lin_x
        self.twist_msg.angular.z = self.pwm_to_adimensional(msg.channels[2]) -
self.pwm_to_adimensional(msg.channels[0]) * self._speed_factor_ang_z

    def pwm_to_adimensional(self, pwm):
        pwm = max(pwm, self._pwm_min)
        pwm = min(pwm, self._pwm_max)
        pwm = pwm - ((self._pwm_max + self._pwm_min) * 0.5)
        pwm = pwm / ((self._pwm_max - self._pwm_min) * 0.5)
        return pwm

    def run(self):
        rate = rospy.Rate(50)
        while not rospy.is_shutdown():
            self.pub_twist.publish(self.twist_msg)
            rate.sleep()


if __name__ == "__main__":
    rc_proxy = RCOutInput()
    rc_proxy.run()
```

rc state messenger.py

```
#!/usr/bin/python
"""
rc_state_messenger.py
Esa Aaltonen 2023

Read remote control data feed and publish messages at switch state change
Also publish stick values [-100,100] and switch values [0,100] in topic /autopilot/RCchannels (for GUI)

This node subscribes to /bunker_rc_status (cf. BunkerRCState.msg) published by bunker_messenger.cpp
(uint8 sws, int8 var_a), cf. Bunker Pro user manual, p. 10:

uint8 sws, switches: CAN frame 0x241, byte [0]:
bit[0-1]: SWA 2 = up, 3 = down
bit[2-3]: SWB 2 = up, 1 = middle, 3 = down
bit[4-5]: SWC 2 = up, 1 = middle, 3 = down
bit[6-7]: SWD 2 = up, 3 = down

int8 var_a "left knob": CAN frame 0x241, byte[5]:
range [-100,100]: -100 = left/down limit, 0 = middle, 100 = right/up limit

When a value changes, the node publishes the corresponding new state once in topic /switch/a, /b, /c or
/d
in UInt8 format, 2 = up, 1 = middle, 3 = down, and in topic /switch/var_a, Int8 -100 ... 100

Requirements:
- CAN up (sudo ip link set can0 up type can bitrate 500000)
- bunker_bringup running (bunker_minimal.launch) (modified code)

"""

import rospy
from std_msgs.msg import UInt8, Int8
from bunker_msgs.msg import BunkerRCState
from bunker_autopilot.msg import RCchannels

# int literals - switch positions
SW_UP = 2
SW_MIDDLE = 1
SW_DOWN = 3

class SWMessenger():
    def __init__(self):
        rospy.init_node("rc_state_messenger ")
        self.sub = rospy.Subscriber("bunker_rc_status", BunkerRCState, self.callback_rc_status)

        rospy.loginfo("> Subscriber created: RC switches messenger")

        self.pub_swa = rospy.Publisher("switch/a", UInt8, queue_size=1)
        self.pub_swb = rospy.Publisher("switch/b", UInt8, queue_size=1)
        self.pub_swc = rospy.Publisher("switch/c", UInt8, queue_size=1)
        self.pub_swd = rospy.Publisher("switch/d", UInt8, queue_size=1)
        self.pub_var_a = rospy.Publisher("switch/var_a", Int8, queue_size=1)
        self.pub_channels = rospy.Publisher("autopilot/RCchannels", RCchannels, queue_size=1)
        self.channel_msg = RCchannels()
        self.channel_msg.button = 0

        self.last_sws = 0
        self.sws_last = [0, 0, 0, 0]
        self.last_var_a = 0
        self.sw_array = [0, 0, 0, 0]

    def callback_rc_status(self, msg):
        self.channel_msg.right_x = msg.right_stick_left_right
        self.channel_msg.right_y = msg.right_stick_up_down
        self.channel_msg.left_x = msg.left_stick_left_right
        self.channel_msg.left_y = msg.left_stick_up_down

        #Check if any of the switches have changed and publish in topics /switch/a, b, c or d
        if msg.sws != self.last_sws:

            for j in range(4):
                k = 3 - j
                mask = (3 << (k*2))
                sw_value = (msg.sws & mask) >> (k*2)
                self.sw_array[k] = sw_value
```

```
            self.channel_msg.swa = self.sw_to_chan(self.sw_array[0])
            self.channel_msg.swb = self.sw_to_chan(self.sw_array[1])
            self.channel_msg.swc = self.sw_to_chan(self.sw_array[2])
            self.channel_msg.swd = self.sw_to_chan(self.sw_array[3])

            if(self.sw_array[0] != self.sws_last[0]):
                self.pub_swa.publish(self.sw_array[0])
            if(self.sw_array[1] != self.sws_last[1]):
                self.pub_swb.publish(self.sw_array[1])
            if(self.sw_array[2] != self.sws_last[2]):
                self.pub_swc.publish(self.sw_array[2])
            if(self.sw_array[3] != self.sws_last[3]):
                self.pub_swd.publish(self.sw_array[3])

            self.last_sws = msg.sws

        # Check if Left knob has changed and publish in topic /switch/var_a
        if msg.var_a != self.last_var_a:
            self.pub_var_a.publish(self.last_var_a)
            self.last_var_a = msg.var_a

    def sw_to_chan(self, pos):
        chan = 0
        if pos == 1: chan = 50
        if pos == 3: chan = 100
        return chan

    def run(self):
        rate = rospy.Rate(25)
        while not rospy.is_shutdown():
            self.pub_channels.publish(self.channel_msg)
            rate.sleep()



if __name__ == "__main__":
    sw_proxy = SWMessenger()
    sw_proxy.run()
```

rc arm-disarm.py

```
#!/usr/bin/python
"""
rc_arm-disarm.py
Esa Aaltonen 2024

Read remote control switches C & D to control FCU flight mode (C) and arming/disarming (D)

This node reads RC switch states from /switch/c and /switch/d and calls flight mode service
/mavros/set_mode to set MANUAL mode (switch C up) or AUTO mode (switch D middle)
and arming command service /mavros/cmd/arming to arm (switch D down) or disarm (switch D up) the FCU.

Requirements:
- CAN up (sudo ip link set can0 up type can bitrate 500000)
- bunker_bringup running (bunker_minimal.launch) (modified code)
- MavROS node running (/mavros/mavros_node)
- rc_state_messenger.py running (publisher for /switch/c and /switch/d)

"""

import rospy
from std_msgs.msg import UInt8, Int8
from mavros_msgs.srv import CommandBool, SetMode
import time

# int literals - switch positions
SW_UP = 2
SW_MIDDLE = 1
SW_DOWN = 3


class RCArming():
    def __init__(self):
        rospy.init_node("rc_arm_disarm")
        self.sub_swc = rospy.Subscriber("switch/c", UInt8, self.callback_update_swc)
        self.sub_swd = rospy.Subscriber("switch/d", UInt8, self.callback_update_swd)

        rospy.loginfo("> Subscriber created: arm/disarm")

        self.swc = 0
        self.swd = 0
        self.aux_mode = ""

    # Read Switch C value
    def callback_update_swc(self, msg):

        #Check if switch C state has changed
        if msg.data != self.swc:
            self.swc = msg.data
            if msg.data == SW_UP: # 2 = switch up - call manual
                rospy.wait_for_service('mavros/set_mode')
                try:
                    flightModeService = rospy.ServiceProxy('/mavros/set_mode', SetMode)
                    isModeChanged = flightModeService(custom_mode='MANUAL') #return true or false
                except rospy.ServiceException as e:
                    rospy.loginfo("Set mode service call failed: %s"%e)

            if msg.data == SW_MIDDLE: # 1 = switch middle - call auto
                rospy.wait_for_service('mavros/set_mode')
                try:
                    flightModeService = rospy.ServiceProxy('/mavros/set_mode', SetMode)
                    isModeChanged = flightModeService(custom_mode='AUTO') #return true or false
                except rospy.ServiceException as e:
                    rospy.loginfo("Set mode service call failed: %s"%e)

            if msg.data == SW_DOWN and rospy.has_param('autopilot/aux_mode'): # 3 = switch down - call
aux mode if set (in param '/autopilot/aux_mode')
                self.aux_mode = rospy.get_param('autopilot/aux_mode')
                rospy.wait_for_service('mavros/set_mode')
                try:
                    flightModeService = rospy.ServiceProxy('/mavros/set_mode', SetMode)
                    isModeChanged = flightModeService(custom_mode=self.aux_mode) #return true or false
                    if not isModeChanged: # if an invalid mode name is given in the param
                        rospy.loginfo("Invalid aux mode name: {0}".format(self.aux_mode))
                except rospy.ServiceException as e:
                    rospy.loginfo("Set mode service call failed: %s"%e)
```

```
    # Read Switch D value
    def callback_update_swd(self, msg):
        #Check if switch D state has changed
        if msg != self.swd:
            self.swd = msg.data
            if msg.data == SW_DOWN: # 3 = switch down call arm
                rospy.wait_for_service('mavros/cmd/arming')
                try:
                    armService = rospy.ServiceProxy('mavros/cmd/arming', CommandBool)
                    armService(True)
                    rospy.loginfo("Arming service call successful")
                except rospy.ServiceException as e:
                    rospy.loginfo("Arming service call failed: %s"%e)

            if msg.data == SW_UP: # call disarm
                rospy.wait_for_service('mavros/cmd/arming')
                try:
                    armService = rospy.ServiceProxy('mavros/cmd/arming', CommandBool)
                    armService(False)
                    rospy.loginfo("Disarming service call successful")
                except rospy.ServiceException as e:
                    rospy.loginfo("Disarming service call failed: %s"%e)

    def run(self):
        rate = rospy.Rate(20)
        while not rospy.is_shutdown():
            rate.sleep()


if __name__ == "__main__":
    arm_proxy = RCArming()
    arm_proxy.run()
```

mission server.py

```
#!/usr/bin/env python
"""
mission_server.py
Esa Aaltonen 2023

This service subscribes to /mavros/mission/waypoints to receive waypoint list from the FCU and to
/mavros/global_position/global to receive current GPS position.

After amending local waypoint list, the list is updated to the FCU by calling service
/mavros/mission/push. The waypoint list is cleared (waypoint list emptied except home position) by
calling /mavros/mission/clear. Calling service /mavros/mission/pull is used to verify the
current number of waypoints on the FCU. This is also necessary to update the list and to verify that
the list published in topic /mavros/mission/waypoints is actually correct.

***
Function set by the task field:

SET_HOME = 0            ()
ADD_WP = 1              (bool use_last, bool use_current, int8 seq)
REMOVE_WP = 2           (bool use_last, bool use_current, int8 seq)
CLEAR_MISSION = 3       ()
BACKWARDS_MISSION = 4   ()
OFFSET_MISSION = 5      (bool all_wps, int8 seq, float32 distance, float32 direction_angle)
SCALE_MISSION = 6       (float32 scale_factor)
ROTATE_MISSION = 7      (float32 direction_angle)
***

The node also publishes relevant navigation information in topic /wp_info:
uint8 total_wps             - total number of waypoints on the list
uint8 next_wp               - current navigation index
float32 distance_to_next_wp - distance in metres
float32 target_bearing      - in degrees, 0 = north
Distances are calculated using a 2D approximation for short distances

"""


import rospy
from std_msgs.msg import Int8, Float32
from bunker_autopilot.srv import MissionManip, MissionManipResponse
from mavros_msgs.msg import Waypoint, WaypointList
from mavros_msgs.srv import WaypointPush, WaypointPull, WaypointClear
from sensor_msgs.msg import NavSatFix
from bunker_autopilot.msg import WPInfo
import math
import time

class distVector():
    def __init__(self, dist=0.0, dxion=0.0):
        self.dist = dist
        self.dxion = dxion

class WPmanip():
    def __init__(self):
        rospy.init_node('mission_server')
        s = rospy.Service('mission_manip', MissionManip, self.handle_task)
        rospy.loginfo("Service mission_manip initiated")

        self.sub_mission_wps = rospy.Subscriber("mavros/mission/waypoints", WaypointList,
self.callback_mission_wps)
        self.sub_global_pos = rospy.Subscriber("mavros/global_position/global", NavSatFix,
self.callback_global_position)
        self.pub_wpinfo = rospy.Publisher("wp_info", WPInfo, queue_size=1)
        self.wpinfo_msg = WPInfo()

        self.global_position = NavSatFix()
        self.got_gp = False
        self.wp = Waypoint()
        self.wps = WaypointList()

        #Distance coefficients for distance approximations
        self.k_lat = 111194
        self.k_long = 50519
        self.k_set = False
```

```
    if rospy.has_param('autopilot/k_lat') and rospy.has_param('autopilot/k_long'):
        self.k_lat = rospy.get_param('autopilot/k_lat')
        self.k_long = rospy.get_param('autopilot/k_long')
        self.k_set = True


# Send new WP list to the FCU
def wpPush(self):
    rospy.wait_for_service('mavros/mission/push')
    try:
        wpPushService = rospy.ServiceProxy('mavros/mission/push', WaypointPush)
        result = wpPushService(start_index = 0, waypoints = self.wps.waypoints)
        if result.success:
            rospy.loginfo("Sent new waypoint list. Number of waypoints: {0}".format(self.wpPull()))
            return True
    except rospy.ServiceException as e:
        rospy.loginfo("Service call push waypoints failed: %s"%e)
        return False

# Request current number of WPs
def wpPull(self):
    rospy.wait_for_service('mavros/mission/pull')
    try:
        wpPullService = rospy.ServiceProxy('mavros/mission/pull', WaypointPull)

        wp_num = 0
        res = wpPullService()
        if res.success:
            wp_num = res.wp_received
        return wp_num

    except rospy.ServiceException as e:
        rospy.loginfo("Service call Pull waypoint failed: %s"%e)


# *** SET_HOME ***
# Set Home at current location
def setCurrentHome(self):
    succ = False

    if len(self.wps.waypoints) > 0:
        self.wps.waypoints[0].x_lat = self.global_position.latitude
        self.wps.waypoints[0].y_long = self.global_position.longitude
        self.wps.waypoints[0].z_alt = self.global_position.altitude

        succ = self.wpPush()

    return succ

# *** ADD_WP ***
# Add new WP as the last item on the list, using current GPS location
def amendLocalWP(self, use_last, use_current, addSeq):
    succ = False

    #Update local list
    self.wp.frame = 3 #  FRAME_GLOBAL_REL_ALT = 3
    self.wp.command = 16 # NAV_WAYPOINT = 16
    self.wp.is_current= False
    self.wp.autocontinue = True
    self.wp.param1 = 0.0
    self.wp.param2 = 0.0
    self.wp.param3 = 0.0
    self.wp.param4 = 0.0
    self.wp.x_lat = self.global_position.latitude
    self.wp.y_long = self.global_position.longitude
    self.wp.z_alt = self.global_position.altitude

    if use_last:
        # add current position as last WP
        self.wps.waypoints.append(self.wp)
        succ = self.wpPush()
        return succ
    if use_current:
        # add current position at current seq
        self.wps.waypoints.insert(self.wps.current_seq, self.wp)
        succ = self.wpPush()
        return succ
```

```
    if addSeq < len(self.wps.waypoints):
        # add current position at indicated position (addSeq)
        self.wps.waypoints.insert(addSeq, self.wp)
        succ = self.wpPush()

    return succ

# *** REMOVE_WP ***
# Remove item at current_seq from the WP list
def removeCurrentWP(self, use_last, use_current, remSeq):
    succ = False

    #Update local list
    if len(self.wps.waypoints) > 1:
        if use_last:
            # remove last WP
            self.wps.waypoints.pop()
            succ = self.wpPush()
            return succ
        if use_current:
            # remove WP at current_seq
            self.wps.waypoints.pop(self.wps.current_seq)
            succ = self.wpPush()
            return succ
        if remSeq < len(self.wps.waypoints):
            # remove WP at remSeq
            self.wps.waypoints.pop(remSeq)
            succ = self.wpPush()
    return succ

# *** CLEAR_MISSION ***
# Remove all waypoints on FCU WP list
def clearMission(self):
    succ = False

    # Instead of calling mavros/mission/clear service, which also clears home wp,
    # just clear local list and add current coordinates as new home
    del self.wps.waypoints[:]

    if (self.global_position.latitude > 0.0):
        self.wp.frame = 0 #  FRAME_GLOBAL_REL_ALT = 3
        self.wp.command = 16 # NAV_WAYPOINT = 16
        self.wp.is_current = True
        self.wp.autocontinue = True
        self.wp.param1 = 0.0
        self.wp.param2 = 0.0
        self.wp.param3 = 0.0
        self.wp.param4 = 0.0
        self.wp.x_lat = self.global_position.latitude
        self.wp.y_long = self.global_position.longitude
        self.wp.z_alt = self.global_position.altitude
        self.wps.waypoints.append(self.wp)
        rospy.loginfo("Waypoint list cleared. Current position set as new home location")

        succ = self.wpPush()

    return succ

# *** BACKWARDS_MISSION ***
# Invert WP list after home position (excluding [0])
def invertWPlist(self):
    succ = False

    if(len(self.wps.waypoints) > 2):
        self.wps.waypoints[1:] = self.wps.waypoints[len(self.wps.waypoints):0:-1]

    #Send list to the FCU
    succ = self.wpPush()
    return succ

# *** OFFSET_MISSION ***
# Add new WP as the last item on the list, using current GPS location
def offsetWPlist(self, all_wps, offSeq, distance, direction):
    succ = False

    if not all_wps:
        if (len(self.wps.waypoints) > offSeq):
```

```
                self.wps.waypoints[offSeq] = self.offsetWP(self.wps.waypoints[offSeq], distance,
direction)

        elif (len(self.wps.waypoints) > 1):
            wp_list = WaypointList()
            wp_list.waypoints.append(self.wps.waypoints[0])
            for wpoff in self.wps.waypoints[1:]:
                wpoff = self.offsetWP(wpoff, distance, direction)
                wp_list.waypoints.append(wpoff)
            self.wps.waypoints = wp_list.waypoints

        succ = self.wpPush()
        return succ

    def offsetWP(self, wpset, dist, dxion):
        if (not self.k_set) and self.got_gp:
            self.calculateCoefficients()

        #rospy.loginfo("offsetWP: lat {0} long{1}".format(wpset.x_lat, wpset.y_long))
        k_x = math.sin(math.radians(dxion))
        k_y = math.cos(math.radians(dxion))
        #rospy.loginfo("OFFSET dist: {0} dxion: {1} k_y: {2} k_x: {3}".format(dist, dxion, k_y, k_x))
        distX = dist * k_x / self.k_long
        distY = dist * k_y / self.k_lat
        #rospy.loginfo("OFFSET distY: {0} distX: {1}".format(distY, distX))
        wpset.y_long += distX
        wpset.x_lat += distY
        #rospy.loginfo("now lat {0} long{1}".format(wpset.x_lat, wpset.y_long))
        return wpset

    # *** SCALE_ROTATE_MISSION ***
    # Move each WP away from Home, measured by distance * sFactor, rotating it in relation to Home
    def scaleRotateMission(self, sFactor, offsetAngle):
        succ = False
        vector = distVector()
        rospy.loginfo("sFactor: {0}, offsetAngle: {1}".format(sFactor, offsetAngle))
        midpoint = self.calculateMidpoint()
        rospy.loginfo("Midpoint x_lat {0} - y_long{1}".format(midpoint.x_lat, midpoint.y_long))

        if (len(self.wps.waypoints) > 1):
            for wpoff in self.wps.waypoints[1:]:
                rospy.loginfo("WPalku - x_lat: {0}, y_long: {1}".format(wpoff.x_lat, wpoff.y_long))
            wp_list = WaypointList()
            wp_list.waypoints.append(self.wps.waypoints[0])

            for wpoff in self.wps.waypoints[1:]:
                #rospy.loginfo("Home - y_long: {0}, x_lat: {1}".format(self.wps.waypoints[0].y_long,
self.wps.waypoints[0].x_lat))
                rospy.loginfo("WPoff - x_lat: {0}, y_long: {1}".format(wpoff.x_lat, wpoff.y_long))
                vector = self.calculateVector(midpoint, wpoff)
                rospy.loginfo("vector.dist: {0}, vector.dxion: {1}".format(vector.dist, vector.dxion))
                wpoff.x_lat = midpoint.x_lat
                wpoff.y_long = midpoint.y_long
                #rospy.loginfo("E - y_long: {0}, x_lat: {1}".format(wpoff.y_long, wpoff.x_lat))
                wpoff = self.offsetWP(wpoff, (vector.dist * sFactor), (vector.dxion + offsetAngle))
                #rospy.loginfo("J - y_long: {0}, x_lat: {1}".format(wpoff.y_long, wpoff.x_lat))
                wp_list.waypoints.append(wpoff)
                rospy.loginfo(len(wp_list.waypoints))
            self.wps.waypoints = wp_list.waypoints
            succ = self.wpPush()

        return succ

    # *** MIRROR_MISSION ***
    # Move each WP rotating it in relation to Home, maintaining distance
    def mirrorMission(self, mirrorAngle):
        succ = False
        oVector = distVector()
        midpoint = self.calculateMidpoint()

        if (len(self.wps.waypoints) > 2):
        # Determine mission mindpoint

            # mirror axle heading must be in range [-90, 90]
            while mirrorAngle > 90:
                mirrorAngle -= 90
            while mirrorAngle < -90:
```

```
                mirrorAngle += 90

            # direction of moving each mirrored WP
            offsetAngle = mirrorAngle + 90

            wp_list = WaypointList()
            wp_list.waypoints.append(self.wps.waypoints[0])

            # process each waypoint
            for wpoff in self.wps.waypoints[1:]:
                # hypotenuse of triangle ABC wpoff - midpoint - (the point where the mirror offset
vector crosses the mirror axle)
                oVector = self.calculateVector(wpoff, midpoint)
                # angle BAC
                alpha = math.radians(offsetAngle - oVector.dxion)
                # side AC
                halfDist = math.cos(alpha) * oVector.dist
                # move WP to mirrored location
                wpoff = self.offsetWP(wpoff, (2 * halfDist), offsetAngle)
                wp_list.waypoints.append(wpoff)
            self.wps.waypoints = wp_list.waypoints
            succ = self.wpPush()

        return succ


    # ********************************************************************************************
    # Check requested task
    def handle_task(self, req):
        MissionManipResponse()
        success = False

        rospy.wait_for_service('mavros/mission/pull')
        try:
            wpPullService = rospy.ServiceProxy('mavros/mission/pull', WaypointPull)
            res = wpPullService()
            if res.success:
                rospy.loginfo(res.wp_received)

        except rospy.ServiceException as e:
            rospy.loginfo("Service call pull waypoint failed: %s"%e)

        # set home in current location
        if req.task == 0:
            success = self.setCurrentHome()

        # add WP
        if req.task == 1:
            success = self.amendLocalWP(req.use_last, req.use_current, req.seq)

        # remove WP
        if req.task == 2:
            success = self.removeCurrentWP(req.use_last, req.use_current, req.seq)

        # clear mission
        if req.task == 3:
            success = self.clearMission()

        # backwards mission
        if req.task == 4:
            success = self.invertWPlist()

        # offset mission
        if req.task == 5:
            success = self.offsetWPlist(req.all_wps, req.seq, req.distance, req.direction_angle)

        # scale & rotate mission
        if req.task == 6:
            success = self.scaleRotateMission(req.scale_factor, req.direction_angle)

        # mirror mission
        if req.task == 7:
            success = self.mirrorMission(req.direction_angle)

        # Request actual number of WPs from the FCU
        number_wps = 0
```

```
        rospy.wait_for_service('mavros/mission/pull')
        try:
            wpPullService = rospy.ServiceProxy('mavros/mission/pull', WaypointPull)
            res = wpPullService()
            if res.success:
                number_wps = res.wp_received

        except rospy.ServiceException as e:
            rospy.loginfo("Service call pull waypoint failed: %s"%e)

        # Return service response message
        return MissionManipResponse(success, number_wps)
    # ***********************************************************************************************

    # *** Calculation functions ***
    # Calculate distance and bearing for a distVector object
    def calculateVector(self, wpSource, wpDestination):
        resultVector = distVector()

        if (not self.k_set) and self.got_gp:
            self.calculateCoefficients()

        # here x = longitude, y = latitude
        diffy = (wpDestination.x_lat - wpSource.x_lat)
        diffx = (wpDestination.y_long - wpSource.y_long)

        disty = diffy * self.k_lat
        distx = diffx * self.k_long
        resultVector.dist = math.sqrt(disty**2 + distx**2)

        t_bearing = 0
        try:
            if resultVector.dist > 0:
                theta = math.acos(disty/resultVector.dist)
                t_bearing = math.degrees(theta)

                if distx < 0:
                    t_bearing = 360 - t_bearing
            resultVector.dist = round(resultVector.dist,2)

            resultVector.dxion = t_bearing
        except rospy.ServiceException as e:
            rospy.loginfo("Direction calculation failed: %s"%e)

        return resultVector

    # Calculate good-enough approximates
    def calculateCoefficients(self):
        # adjusted equatorial diameter to compensate differences between the geoid and a sphere, for
lat ~63
        diagEq = 12793172
        self.k_lat = math.pi * diagEq / 360
        try:
            self.k_long = math.pi * math.cos(math.radians(self.global_position.latitude)) * diagEq /
360
        except rospy.ServiceException as e:
            rospy.loginfo("Calculating k_long: %s"%e)

        self.k_set = True
        rospy.set_param('autopilot/k_lat', self.k_lat)
        rospy.set_param('autopilot/k_long', self.k_long)

    # Calculate distance & bearing to next WP
    def distanceNextWP(self):
        if (not self.k_set) and self.got_gp:
            self.calculateCoefficients()

        nextwp = 1
        if (self.wps.current_seq < len(self.wps.waypoints)) and (self.wps.current_seq > 0):
            nextwp = self.wps.current_seq
        # here x = longitude, y = latitude
        diffy = (self.wps.waypoints[nextwp].x_lat - self.global_position.latitude)
        diffx = (self.wps.waypoints[nextwp].y_long - self.global_position.longitude)

        disty = (diffy * self.k_lat)
        distx = (diffx * self.k_long)
        dist = math.sqrt(disty**2 + distx**2)
```

```
        self.wpinfo_msg.distance_to_next_wp = dist

        t_bearing = 0
        try:
            if dist > 0:
                #rospy.loginfo("Disty: {0}, dist: {1}".format(disty,dist))
                theta = math.acos(disty/dist)
                t_bearing = math.degrees(theta)

                if distx < 0:
                    t_bearing = 360 - t_bearing
            dist = round(dist,2)
            #rospy.loginfo("dist: {0}, bearing: {1}".format(dist,t_bearing))
            self.wpinfo_msg.target_bearing = t_bearing
        except rospy.ServiceException as e:
            rospy.loginfo("Bearing calculation failed: %s"%e)

    # ***********************************************************************************************

    def calculateMidpoint(self):
        midpoint = Waypoint()
        midpoint.x_lat = self.wps.waypoints[1].x_lat
        midpoint.y_long = self.wps.waypoints[1].y_long

        if (len(self.wps.waypoints) > 2):
        # Determine mission mindpoint
            maxX = midpoint.y_long
            minX = midpoint.y_long
            maxY = midpoint.x_lat
            minY = midpoint.x_lat

            for wp in self.wps.waypoints[2:]:
                maxX = max(maxX, wp.y_long)
                minX = min(minX, wp.y_long)
                maxY = max(maxY, wp.x_lat)
                minY = min(minY, wp.x_lat)
            midpoint.y_long = (minX + maxX) / 2
            midpoint.x_lat = (minY + maxY) / 2
        return midpoint

    # *** Run function ***
    def run(self):
        rate = rospy.Rate(5)
        while not rospy.is_shutdown():
            if len(self.wps.waypoints) > 1:
                self.distanceNextWP()
                self.pub_wpinfo.publish(self.wpinfo_msg)
            rate.sleep()

    # *** Callback functions ***
    # Read WP list from the FCU
    def callback_mission_wps(self, data):
        if self.wps.current_seq != data.current_seq:
            rospy.loginfo("Current mission waypoint sequence updated: {0}".format(data.current_seq))
        self.wps = data
        self.wpinfo_msg.total_wps = len(self.wps.waypoints)

        self.wpinfo_msg.next_wp = self.wps.current_seq

    # Read GPS position data
    def callback_global_position(self, data):
        self.global_position = data
        self.got_gp = True

def shutdown_msg():
    rospy.loginfo("Exiting server node")

if __name__ == "__main__":
    rospy.on_shutdown(shutdown_msg)
    wpservice = WPmanip()
    wpservice.run()
```

mission client.py

```
#!/usr/bin/python
"""
mission_client.py
Esa Aaltonen 2023

Read remote control switch A and left knob VRA to insert/remove waypoints

This node subscribes to /switch/a and /switch/var_a published by rc_state_messenger.py
(uint8 for /switch/a, int8 for /switch/var_a) and calls service /bunker_autopilot/mission_manip
to manipulate the waypoint list.

***
Functions triggered by switch A down (at state change):
- if VRA middle: append new waypoint at the end of the list, using current position
- if VRA down (< -25): clear waypoint list
- if VRA up (> 25): remove waypoint at current_seq (waypoint currently navigated to if in AUTO mode, or
last waypoint if mission finished)
***

Requirements:
- ugv_sdk (Weston Robot, v1.x) and bunker_base (Agilex Robotics) packages
    (modified versions; see https://github.com/e-aaltonen/bunker_ros_RC/tree/old and
https://github.com/e-aaltonen/ugv_sdk_RC)
- CAN up (sudo ip link set can0 up type can bitrate 500000)
- bunker_bringup running (bunker_minimal.launch)
- rc_state_messenger.py running
- MavROS node (mavros_node) running

"""

import rospy
from std_msgs.msg import UInt8, Int8, Float32
from bunker_autopilot.srv import MissionManip
import time

# int literals - switch positions
SW_UP = 2
SW_MIDDLE = 1
SW_DOWN = 3

# MissionManip.srv literals for task field (int8)
SET_HOME = 0
ADD_WP = 1
REMOVE_WP = 2
CLEAR_MISSION = 3
BACKWARDS_MISSION = 4
OFFSET_MISSION = 5
SCALE_MISSION = 6
ROTATE_MISSION = 7

class WPmanip():
    def __init__(self):
        rospy.init_node("wp_manip")
        self.sub_swa = rospy.Subscriber("switch/a", UInt8, self.callback_update_swa)
        self.sub_swb = rospy.Subscriber("switch/b", UInt8, self.callback_update_swb)
        self.sub_var_a = rospy.Subscriber("switch/var_a", Int8, self.callback_update_var_a)

        rospy.loginfo("> wp_manip node initiated")

        self.swa = 0
        self.swb = 0
        self.var_a = 0


    # Check whether Switch A position has changed
    def callback_update_swa(self, msg):
        if msg.data != self.swa:
            self.swa = msg.data

            if msg.data == SW_DOWN and self.swb != SW_DOWN: # 3 = switch down to activate function,
unless SWB is down
                # left knob turned left (down): clear mission
                if self.var_a < -25:
                    try:
                        rospy.wait_for_service('mission_manip')
```

```
                        wpService = rospy.ServiceProxy('mission_manip', MissionManip)
                        serviceSuccess = wpService(task = CLEAR_MISSION)
                    except rospy.ServiceException as e:
                        rospy.loginfo("Service call CLEAR_MISSION failed: %s"%e)

                # left knob turned right (up): remove current WP
                if self.var_a > 25:
                    try:
                        rospy.wait_for_service('mission_manip')
                        wpService = rospy.ServiceProxy('mission_manip', MissionManip)
                        serviceSuccess = wpService(task = REMOVE_WP, use_last = False, use_current =
True, seq = 0)
                    except rospy.ServiceException as e:
                        rospy.loginfo("Service call REMOVE_WP failed: %s"%e)

                # left knob in the middle: push WP
                if (self.var_a > -26) & (self.var_a < 26):
                    # check whether GPS position was received
                    try:
                        rospy.wait_for_service('mission_manip')
                        wpService = rospy.ServiceProxy('mission_manip', MissionManip)
                        serviceSuccess = wpService(task = ADD_WP, use_last = False, use_current = True,
seq = 0)
                    except rospy.ServiceException as e:
                        rospy.loginfo("Service call ADD_WP failed: %s"%e)

    def callback_update_swb(self, msg):
        if msg.data != self.swb:
            self.swb = msg.data

    # Read left knob value
    def callback_update_var_a(self, msg):
        self.var_a = msg.data


    def run(self):
        rate = rospy.Rate(5)
        while not rospy.is_shutdown():
            rate.sleep()


if __name__ == "__main__":
    wp_proxy = WPmanip()
    wp_proxy.run()
```

MissionManip.srv

```
int8 SET_HOME = 0
int8 ADD_WP = 1
int8 REMOVE_WP = 2
int8 CLEAR_MISSION = 3
int8 BACKWARDS_MISSION = 4
int8 OFFSET_MISSION = 5
int8 SCALE_ROTATE_MISSION = 6
int8 MIRROR_MISSION = 7

# Field used for task       0   1   2   3   4   5   6   7
int8 task               #   X   X   X   X   X   X   X   X
bool use_last           #       X   X                       whether to add/remove last waypoint
bool use_current        #       X   X
int8 seq                #       X   X           X           if use_last = False, then manipulate WP at
[seq]
bool all_wps            #                       X
float32 distance        #                       X
float32 direction_angle #                       X   X   X
float32 scale_factor    #                           X

---
bool success
int8 number_wps
```

Angles.msg

```
float32 pitch
float32 roll
```

RCchannels.msg

```
int8 right_x
int8 right_y
int8 left_x
int8 left_y
int8 swa
int8 swb
int8 var_a
int8 button
```

WPInfo.msg

```
uint8 total_wps
uint8 next_wp
float32 distance_to_next_wp
float32 target_bearing
```

WPTunes.lua

```
local RUN_INTERVAL_MS = 500

-- get number of waypoints, excluding home
local last_mission_length = mission:num_commands() - 1
local last_nav_wp = mission:get_current_nav_index()

-- set tunes for mission changes
local function cleared()
    notify:play_tune('MFT180MLO1L16AC<A')
end

local function added()
    notify:play_tune('MFT180MLO1L16A>CA')
end

local function removed()
    notify:play_tune('MFT180MLO2L16AC<A')
end

local function new_wp()
    notify:play_tune('MFT180MSO2L16G0G')
end

function tunes()

    local mission_length = mission:num_commands() - 1
    local current_wp = mission:get_current_nav_index()

    if mission_length ~= last_mission_length then
        gcs:send_text(0, string.format("Mission: new num commands: %d",mission_length))
        -- mission cleared
        if mission_length < 1 then
            cleared()
        end

        -- new waypoint added
        if mission_length > last_mission_length then
            added()
        end

        -- waypoint removed
        if mission_length < last_mission_length and mission_length > 0 then
            removed()
        end

        last_mission_length = mission_length
    end

    -- current WP has changed
    if current_wp ~= last_nav_wp then
        new_wp()
        last_nav_wp = current_wp
    end

    return tunes, RUN_INTERVAL_MS
end

return tunes, RUN_INTERVAL_MS
```

bunker autopilot agx.launch

```
<launch>
    <include file="$(find bunker_bringup)/launch/bunker_minimal.launch"/>

            <node pkg="mavros" type="mavros_node" name="mavros" output="screen">
                        <param name="fcu_url" value="/dev/ttyUSB0:57600" />
            </node>

    <param name="autopilot/speed_factor_lin_x" value="1.0"/>
    <param name="autopilot/speed_factor_ang_y" value="1.0"/>

    <param name="autopilot/k_lat" value="111194"/>
    <param name="autopilot/k_long" value="50519"/>

    <node pkg="bunker_autopilot" type="rcout_to_cmd_vel.py" name="rcout_to_cmd_vel"/>
    <node pkg="bunker_autopilot" type="rc_state_messenger.py" name="rc_state_messenger"/>
    <node pkg="bunker_autopilot" type="rc_arm-disarm.py" name="rc_arm_disarm"/>

    <node pkg="bunker_autopilot" type="mission_server.py" name="mission_server" output="screen"/>
    <node pkg="bunker_autopilot" type="mission_client.py" name="mission_client" output="screen"/>


</launch>
```