LAPIN AMK
Lapland University of Applied Sciences

AUTONOMISTEN ALUSTOJEN PAIKANNUSJÄRJESTELMIEN MÄÄRITYS

Sampo Haapalainen

Opinnäytetyö

Tieto- ja viestintätekniikka
Insinööri (AMK)

2022

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tieto- ja viestintätekniikka
Insinööri (AMK)

Tekijä	Sampo Haapalainen	Vuosi	2022
Ohjaaja	Tommi Kokko		
Toimeksiantaja	Lapland Robotics		
Työn nimi	Autonomisten alustojen paikannusjärjestelmien määritys		
Sivu- ja liitesivumäärä	50 + 2		

Opinnäytetyön tavoitteena oli määrittää paikannusjärjestelmiä autonomisten alustojen käyttöön Lapin ammattikorkeakoulun Lapland Robotics -hankkeelle. Hankkeen alustat oli suunniteltu prototyypiksi kunnossapitotehtäviin, kuten lumen auraamiseen ja linkoamiseen sekä fat bike -radan kunnossapitoon.

Paikannusjärjestelmien toimivuuden vuoksi työssä oli tarkoitus kokeilla erikseen sisä- ja ulkokäyttöisiä paikannusjärjestelmiä. Paikantamista testattiin kampuksen kellarikerroksessa ja rakennuksen takapihalla.

Alkuperäisissä suunnitelmissa oli hyödyntää Marvelmind-, Novatel- ja Poxyz-järjestelmiä sekä erillistä GPS-moduulia ESP32-mikrokontrollerin kanssa. Näistä järjestelmistä tai laitteistoista Marvelmind- ja Poxyz-järjestelmät suunniteltiin soveltuvan sisäkäyttöön, ja Novatel-järjestelmä sekä erillinen GPS-moduuli oli tarkoitus määrittää ulkokäyttöön. Alun perin suunniteltua Pozyx-laitteistoa ei ollut mahdollista tutkia tai testata laitteiston saatavuuden vuoksi. Työssä kului paljon aikaa ulkopaikannusjärjestelmän kehittämisessä, ja osa ajasta kului myös Marvelmind-laitteiston toiminnallisuuksien tutkimisessa.

Työn lopputuloksena oli GPS-paikannin, joka soveltui ulkokäyttöön. Tätä paikanninta kehitettiin ROS-ympäristöstä virtuaaliseen Carla-ympäristöön asti, jossa dataa on mahdollista käsitellä jatkossa lisää alustan toiminnoille. Sisäkäyttöisistä paikannusjärjestelmistä valitulla Marvelmindilla ehdittiin tekemään muutamia kokeiluja ja selvityksiä.

Avainsanat autonomiset järjestelmät, GPS, paikannus

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Degree Programme in Information
and Communication Technology
Bachelor of Engineering

Author Sampo Haapalainen Year 2022
Supervisor Tommi Kokko
Commissioned by Lapland Robotics
Subject of thesis Configuration of Autonomous Platform Positioning Systems
Number of pages 50 + 2

The aim of this thesis study was to configure positioning systems for autonomous platforms. The study was done to a Lapland University of Applied Sciences project named Lapland Robotics. Project's platforms were designed as prototypes in winter maintenance tasks such as snow pushing and blowing and for fat bike trail maintenance.

Indoor and outdoor positioning systems were tested separately due to the functionality of the positioning systems. Positioning was tested in the basement of the campus and in the backyard of the building.

The original plan was to test Marvelmind, Novatel and Pozyx systems as well as separate GPS module with ESP32 microcontroller. From these systems or hardware Marvelmind and Pozyx were designed for indoor use, and the Novatel system as well as separate GPS module were to be set up for outdoor use. It was impossible to test or research the originally planned Pozyx hardware due to problems in hardware availability. A lot of time was spent in the development of the outdoor positioning system, and part of the time went also in the research of Marvelmind hardware functionality.

The result of the study was a GPS locator which is suitable for outdoor use. This locator was developed all the way to the virtual Carla environment where it is possible to process the data further for the functions of the autonomous platforms. As far as indoor positioning was concerned there was time to conduct a few pieces of experiments and studies of the chosen Marvelmind hardware.

Key words autonomous systems, GPS, positioning

SISÄLLYS

KÄYTETYT LYHENTEET JA TERMIT

aihe	ROS-järjestelmässä solmun tiedonvälityksen väylä (ROS 2022g)
baudinopeus	sarjamonitorin ja mikrokontrollerin tiedonvälityksessä käytetty nopeus
BeiDou	vuonna 2020 Kiinan kansantasavallan käyttöönottama satelliittijärjestelmä (National Coordination Office for Space-Based Positioning, Navigation, and Timing 2021)
Carla	Unreal Engine -pelimoottoriin ja Rosbridge-protokollaan perustuva virtualisointiympäristö
Export	Linux-järjestelmissä käytetty hakemistojen tai tiedostojen polkuja määrittävä komento
Firmware	laitteistoon perustuva ohjelmisto
Galileo	vuonna 2016 Euroopan Unionin käyttöönottama satelliittijärjestelmä (National Coordination Office for Space-Based Positioning, Navigation, and Timing 2021)
Glonass	Globalnaya Navigazionnaya Sputnikovaya Sistema, Neuvostoliiton aikana kehitetty, nykyisin Venäjän federaation hallinnoima satelliittipaikannusjärjestelmä (National Coordination Office for Space-Based Positioning, Navigation, and Timing 2021)
GNSS	Global Navigation Satellite System, usean maan tai maanosan satelliiteista koostuva yhtenäinen satelliittipaikannusjärjestelmä (Maanmittauslaitos 2022)

GPS	Global Positioning System, Yhdysvaltain kehittämä satelliittijärjestelmä
Hedge	Marvelmind Robotics -yhtiön kehittämä ultraääneen perustuva kannettava majakka
HUD-arvo	Unreal Editorin kamerassa näkyvät arvot
IA	Marvelmind-majakoiden hyödyntämä käänteinen arkkitehtuurin rakenne
NavIC	Intian mantereen alueella toimiva, Intian hallinnon omistama satelliittijärjestelmä (National Coordination Office for Space-Based Positioning, Navigation, and Timing 2021)
NIA	Marvelmind-majakoiden hyödyntämä ei-käänteinen arkkitehtuurin rakenne
Predator	robotiikkalaboratorion hyödyntämä työasema Acer Predator G3-710
Publisher	ROS-ympäristössä dataa julkaiseva tai lähettävä aiheen tyyppi
QZSS	Japanin hallituksen omistama ja QZS System Service Inc. -yhtiön hallinnoima satelliittijärjestelmä (National Coordination Office for Space-Based Positioning, Navigation, and Timing 2021)
ROS	robotiikan kehittämiseen suunniteltu järjestelmä (ROS 2022a)

Rosbridge	Rosserial-liikenteen siltana ja Carla-ympäristön tukena toimiva protokolla (ROS 2022c)
Roscore	ROS-järjestelmän ydin sisältää järjestelmässä suoritettavat solmut ja ohjelmat (ROS 2022d)
Rosserial	ROS-ympäristön verkkoyhteyksien muodostamisessa käytetty protokolla (ROS 2022f)
ROS-palvelin	ROS-järjestelmään perustuva palvelin
solmu	ROS-järjestelmän laskentaan perustuva prosessi (ROS 2022b)
Source	Linux-komentotulkin käyttämien tiedostojen polkuja tallentava komento (GeeksforGeeks 2021)
Submap	majakoiden alikartta Marvelmind Dashboard -ohjelmistossa
Subscriber	ROS-järjestelmän tietoa vastaanottava tai tilaava aiheen tyyppi
TCP	ESP32-mikrokontrollerin ja WiFi-tukiaseman hyödyntämä verkkoprotokolla
UWB	Ultra-wideband, ultraääneen perustuva radioteknologia
VNC	Linux-järjestelmien väliselle etätyöpöytäyhteydelle käytetty protokolla
WiFi-suoritinkanta	WiFi-yhteys tai *WiFi-socket*, jonka kautta mikrokontrolleri pystyy lähettämään tietoa ROS-palvelimelle

1 JOHDANTO

Nykyisin paikannusteknologiat ovat entistä yleisempiä ja kehityksen myötä tarjolle on tullut monenlaisia ratkaisuja. Eri paikannusteknologioita on mahdollista hyödyntää eri käyttötarkoituksiin, ja siksi niillä on omat vahvuutensa ja heikkoutensa. Paikannusteknologioita hyödynnetään lisääntyvästi teollisuudessa ja autonomisissa ympäristöissä.

Tällä opinnäytetyöllä selvitetään, mitkä valituista paikannusjärjestelmistä soveltuvat parhaiten Lapland Robotics -hankkeen autonomisten alustojen käyttöön. Työssä esitetään niitä paikannusjärjestelmiä, joilla tehdään kokeiluja eri ympäristöissä. Työ edellyttää ohjelmointia ja koodin soveltamista, jotta paikannukseen liittyvää tietoa saadaan käsiteltyä jatkossa autonomisten alustojen muihin toiminnallisiin tarpeisiin. Työssä tulee tutuksi myös autonomisten alustojen kehitykseen käytetty ROS-ympäristö.

Projektissa käytettiin eri työvaiheita, jotka oli jaettu selvitys-, määritys- ja testausvaiheisiin. Selvitysvaiheessa selvitettiin työn aikana kohdattuja ongelmatilanteita, määritysvaiheessa asennettiin tarvittavat ohjelmistot ja toteutusvaiheessa keskityttiin ohjelmointiin tai laitteiston käyttöön.

Paikannusjärjestelmien ansiosta saadaan alustojen tarkka paikkatieto selvitettyä. Paikkatiedosta hyödytään pääasiallisesti talviaikaan tapahtuvissa kunnossapitotehtävissä, kuten lumen auraamisessa ja linkoamisessa sekä fat bike -ratojen kunnossapidossa. Autonomiset alustat käyttävät sisä- ja ulkotilojen paikannuksessa erillisiä laitteistoja.

Ulkokäyttöisen paikannusjärjestelmän tuloksena on ohjelma, jota käytetään ESP32:n ja Grove GPS -moduulin kanssa. Sisäkäyttöisistä paikannusjärjestelmistä Marvelmind-laitteisto saatiin määritettyä ja testattua. Alkuperäisten suunnitelmien Pozyx-järjestelmää ei ollut mahdollista määrittää laitteiston saatavuuden vuoksi. Tämän lisäksi Novatel-järjestelmän määritys otettiin myöhemmin pois paikannusjärjestelmien vaihtoehdoista työn aikataulun vuoksi.

Opinnäytetyössä hyödynnetty robotiikkalaboratorion testialue mahdollistaa paikannusjärjestelmien toimivuuden testaamisen, jonka avulla saadaan varmistettua järjestelmien kelpoisuutta tarvittaviin tilanteisiin. Järjestelmien kelpoisuuden testaamista voidaan kutsua englannin kielen termillä *Proof-of-Concept*, joka tarkoittaa suomeksi konseptitodistusta.

Hankkeen autonomiset alustat pyrkivät tulevaisuudessa hyödyntämään paikantamisen lisäksi myös odometriaa. Odometrian avulla pystytään laskemaan esimerkiksi alustan kulkema matka antureista saatujen tietojen perusteella.

2 GPS-PAIKANTAMINEN

2.1 GPS-paikannusjärjestelmän historia

GPS-paikannukseen perustuvien järjestelmien kehitys juontaa juurensa 1910-luvulle, jolloin ensimmäiset laitteistot olivat radionavigointijärjestelmiä. Järjestelmiä kehitettiin alun perin meri- ja ilmaliikenteen helpottamiseksi. Myöhemmin kehittyi idea satelliittijärjestelmistä, joiden kehitys aloitettiin vuonna 1973 Yhdysvalloissa. Koska erillisiä järjestelmiä oli jo ennestään olemassa, päätettiin aikaisemmat järjestelmät yhdistää yhdeksi Navstar GPS -sotilasjärjestelmäksi. (Henttu & Lehtoranta 1993, 9.)

Navstar GPS -järjestelmästä syntyi vuosien aikana maailmanlaajuinen paikannusjärjestelmä, jonka paikannustietoja hyödynnetään päivittäin siviilikäytössä ja viranomaistahojen puolella eri tehtävissä (Henttu & Lehtoranta 1993, 15). Järjestelmää ylläpidetään vähintään 24:llä Maata kiertävällä satelliitilla. Nykyisin toimintakelpoisia satelliitteja on kaiken kaikkiaan 26 – 30 kappaletta. (National Coordination Office for Space-Based Positioning, Navigation, and Timing 2022.)

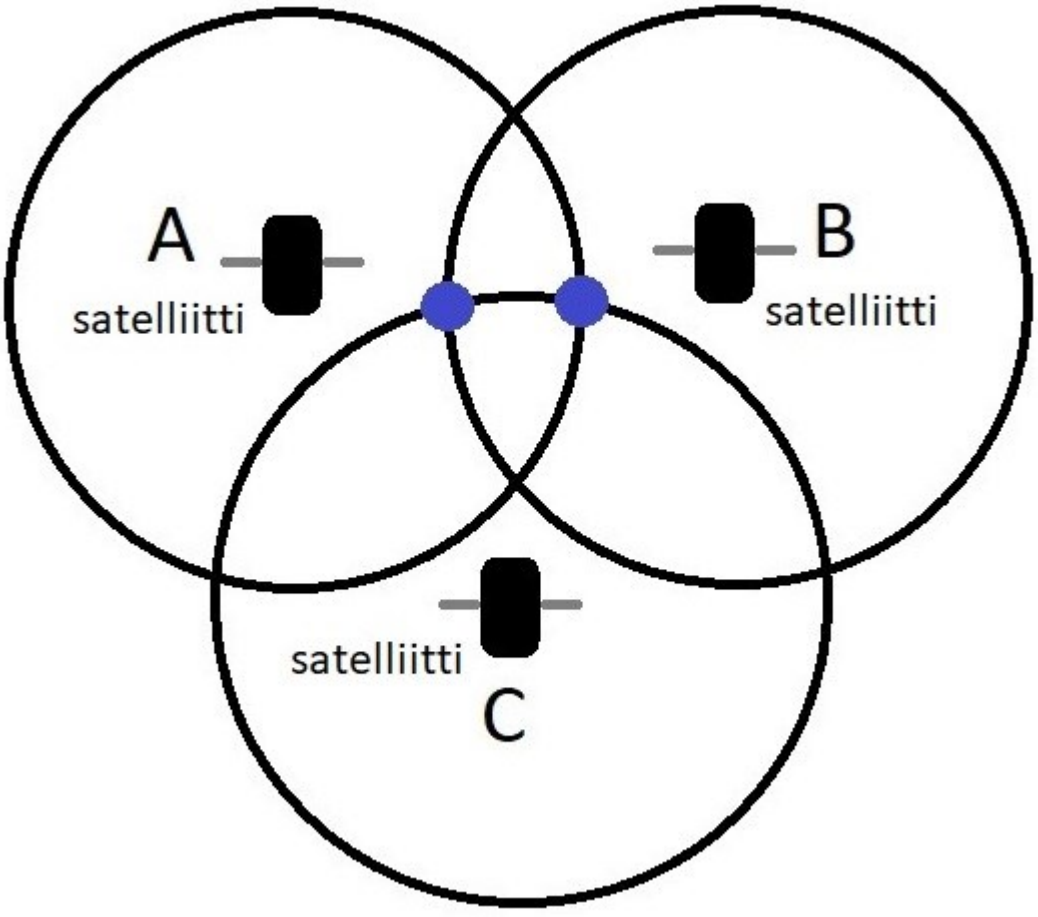
Yhdysvaltain lisäksi Neuvostoliitto kehitti aikoinaan oman Glonass-satelliittipaikannusjärjestelmän, jonka satelliitteja laukaistiin Maan kiertoradoilleen Tjuratamista 12.10.1982. Vuoden 1993 loppupuolella satelliitteja oli käytössä yli 60 kappaletta. Myöhemmin sekä GPS- että Glonass-järjestelmät oli tarkoitus yhdistää yhtenäisjärjestelmäksi, joka tunnetaan nykyisin lyhenteenä GNSS. (Henttu & Lehtoranta 1993, 86.) Tähän järjestelmään kuuluu nykyisin myös Euroopan ja Kiinan satelliittijärjestelmät (Maanmittauslaitos 2022).

Aikaisemmin mainittujen satelliittipaikannusjärjestelmien lisäksi tunnetaan lukuisia muita järjestelmiä alueellisesti ja globaalisti. Toiminnassa olevia järjestelmiä on muun muassa Euroopalla (Galileo), Kiinalla (BeiDou), Intialla (NavIC) ja Japanilla (QZSS). (National Coordination Office for Space-Based Positioning, Navigation, and Timing 2021.)

2.2 GPS-paikantamisen perusteet

GPS-paikantaminen sisältää satelliittien etäisyyden mittausta ja GPS-signaalin ajoitusta. Paikantamista on helpointa ajatella kolmiulotteisesti, jolloin jokainen satelliitti voidaan ajatella oman ympyränsä sisälle. Ympyröiden kohdatessa päästään määrittämään etäisyyksien pisteet. (Henttu & Lehtoranta 1993, 22, 23, 32.)

Etäisyyden mittauksessa voidaan hyödyntää kolmea satelliittia. Kolmas satelliitti rajaa kohteen sijainnin kahden muun satelliitista saadun pisteen välille. Tämänkaltaisesta tilanteesta nähdään esimerkki kuvion 1 mukaisesti, jonka satelliitti C rajaa sijainnin satelliittien A ja B sinisten pisteiden välille. Lisämittaukset ovat kuitenkin tarpeen paikantamisen tarkkuudelle. On myös hyvä ottaa huomioon sijaintipaikan korkeus, sillä korkeuden tietäminen vähentää laskettavia satelliittimittauksia yhdellä. Kohteen etäisyyksien mittauksissa periaatteena on kolmioida paikannettavan kohteen sijainti. (Henttu & Lehtoranta 1993, 22, 24.)



Kuvio 1. Satelliittien etäisyyksien mittaus

Etäisyysmittaukset lasketaan nopeuden ja ajan avulla. Kun molemmat arvot kerrotaan, saadaan tulokseksi kuljettu matka. Koska GPS-järjestelmä toimii radiotekniikalla, käytetään radioaaltojen kulkemaa nopeutta. Radioaallot kulkevat noin 300 000 kilometriä sekunnissa. Matkaan kulunut aika saadaan vertailemalla radiosanoman lähetettyä ja vastaanotettua aikaa. Lopulta saadaan etäisyyden, kun laskettu aikaero kerrotaan radioaaltojen nopeudella. (Henttu & Lehtoranta 1993, 26.)

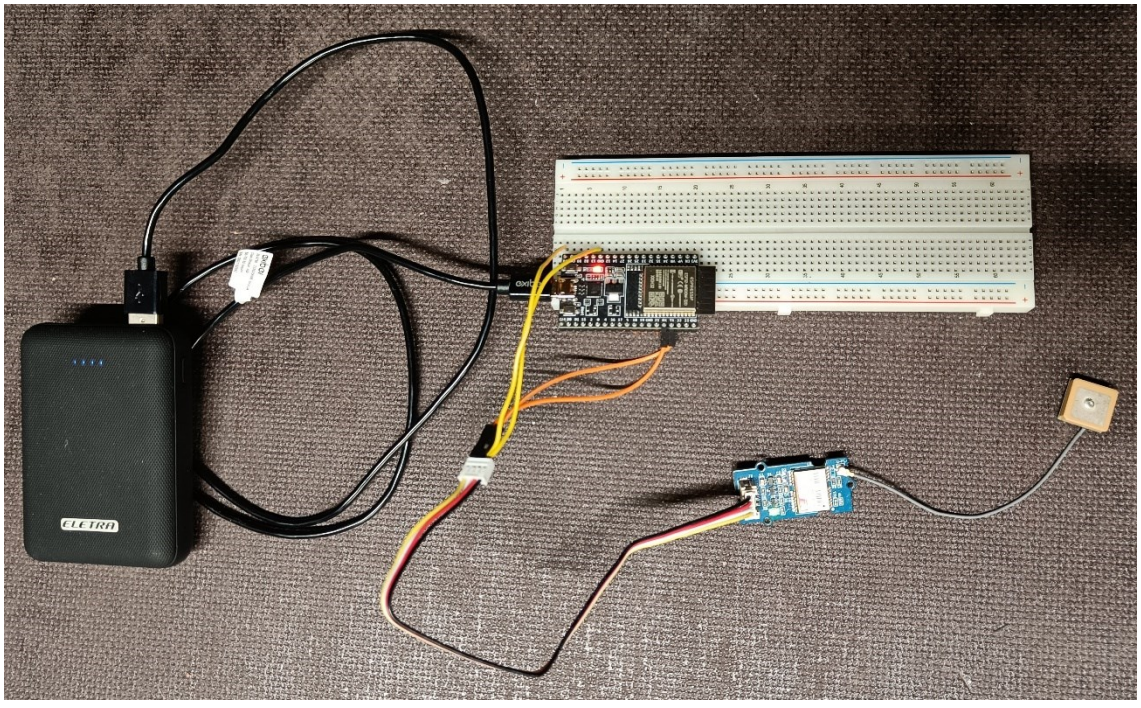
Jotta signaalin kulkema aika saataisiin selvitettyä, on GPS-järjestelmä tahdistettu satelliitissa ja vastaanottimessa niin, että nämä muodostavat samanlaisia merkkijonoja samanaikaisesti. Radiosanomien merkkijonot perustuvat ykkösiin ja nolliin. Viive saadaan ratkaistua, kun edellinen vastaanotettu merkkijonon kohta saadaan uudelleen ja vertaillaan nykyiseen aikaan saatua samaa merkkijonon kohtaa. Merkkijonot eivät siis ole heti synkroniassa toistensa kanssa, vaikka satelliitissa ja vastaanottimessa käytetään samoja merkkijonoja. (Henttu & Lehtoranta 1993, 28.)

3 ULKOPAIKANNUSJÄRJESTELMÄ

3.1 Kokoonpanon ominaisuudet

Testeissä toimi ESP32-WROOM-32D-mikrokontrolleri ja Grove GPS SIM28 -moduuli. Mikrokontrollerin virtalähteenä käytettiin 10 000 mAh Eletra EPB10BK -akkupankkia, jonka toisesta USB-portista ulostulevan virran arvoksi oli merkitty 5 Vdc 1,0 A. Kokoonpanon valintaan vaikuttivat elektroniikan edullinen hinta ja saatavuus. ESP32:n ja Grove GPS -moduulin hinta oli yleisesti noin 50 euroa.

Kuviossa 2 akkupankki sijaitsee kuvan vasemmalla puolella, kytkentäalustaan kytketty ESP32-mikrokontrolleri keskellä ja GPS-moduuli oikealla. GPS-moduulista lähtevän koaksiaalikaapelin päässä on U-FL-liitännän avulla kiinnitetty antenni, jonka kautta GPS-signaaleja otetaan vastaan. Kytkentäalustasta oli hyötyä testien aikana ESP32:n tukemisessa ja kuljettamisessa, mutta kytkentäalusta ei ole kuitenkaan välttämätön.

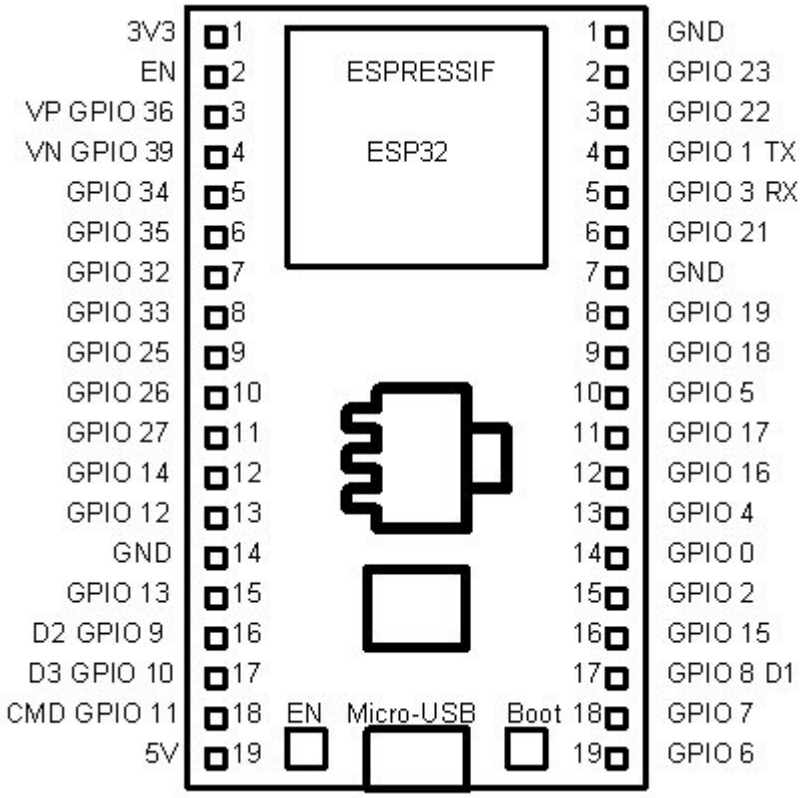


Kuvio 2. Akkupankki, ESP32-mikropiiri ja kytkentäalusta, GPS-moduuli ja moduulin antenni

3.1.1 ESP32-WROOM-32D

ESP32 on Espressif Systemsin valmistama mikrokontrolleri, jota käytetään paljon erilaisissa IoT-ympäristöissä (Espressif Systems 2022, 1). IoT tulee sanoista esineiden internet, jonka ideana on aikaisemmin mainitun nimen mukaisesti yhdistää laitteita Internetiin (Logistiikan Maailma 2022).

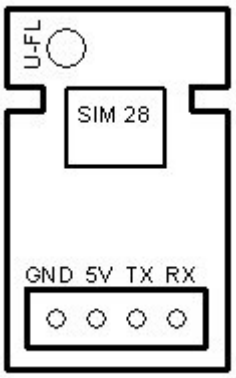
Mikrokontrolleri sisältää tuen WiFi-, Bluetooth- ja Bluetooth LE-yhteyksille (Espressif Systems 2022, 6). Kontrolleri sisältää 38 pinniä, joista yksi pinni toimii 5 voltin jännitteellä, toinen pinni 3,3 voltin jännitteellä, ja kolme muuta pinniä on tarkoitettu maadoitukselle. Kontrolleri sisältää myös opinnäytetyössä tarvittavat RX- ja TX-pinnit, joista kerrotaan myöhemmin lisää. Alapuolelta löytyy kuvio 3, johon on merkattuna mikrokontrollerin kaikki GPIO-pinnit. Näiden pinnien tietämisestä on apua tulevien kytkentöjen toteuttamisessa.



Kuvio 3. ESP32-WROOM-32D, kaavio pinneistä

3.1.2 Grove GPS SIM28

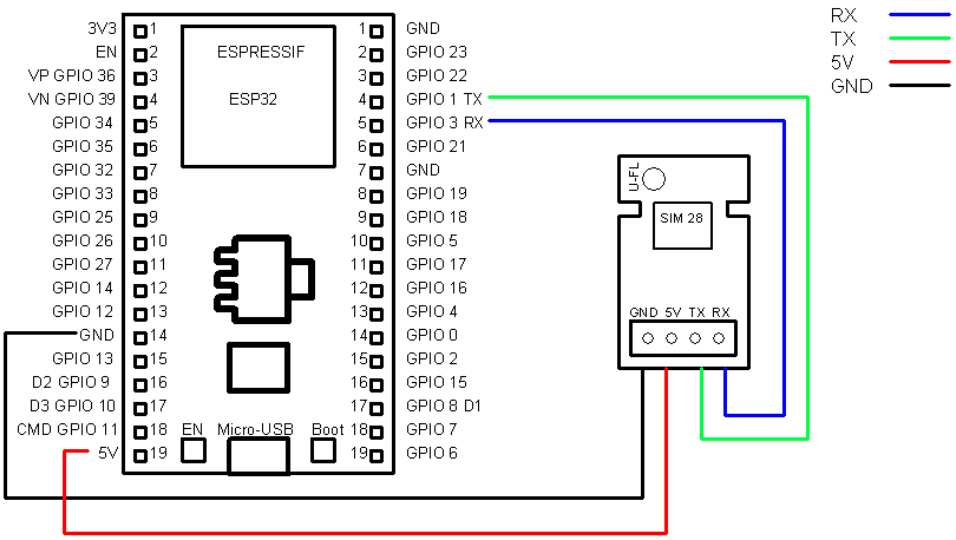
Työssä hyödynnetty Grove GPS -vastaanotin tukee 22 seuranta- ja 66 lisäkanavaa (Seeed Technology 2021). Vastaanottimessa satelliittien lähettämä signaali kulkee yhtä kanavaa pitkin samalla tavalla, kuin televisiot hyödyntävät kanavia ohjelmien lähetyksessä (Henttu & Lehtoranta 1993, 89). Moduuli tukee baudinopeuden arvoja lukujen 9 600 – 115 200 väliltä. Baudinopeus kuvaa tiedonvälityksen nopeutta. Kuviosta 4 nähdään vastaanottimen pinnit.



Kuvio 4. Grove GPS -vastaanottimen pinnit

3.2 Tarvittavat kytkennät

Kytkennät Grove GPS:n ja ESP32:n välillä ovat hyvin yksinkertaisia. Maadoitusjohto yhdistetään GND-pinniin, virtajohto 5V-pinniin, tietoa vastaanottava johto RX-pinniin ja tietoa lähettävä johto TX-pinniin. Koodin lataamisen ajaksi GPS-moduulin maadoitus- ja virtajohdot pitää irrottaa, jotta koodi latautuu ESP32:een onnistuneesti. Kuviosta 5 nähdään mikrokontrollerin ja vastaanottimen kytkennät.



Kuvio 5. ESP32:n ja Grove GPS -moduulin kytkennät

3.3 Kehitysympäristön määritys

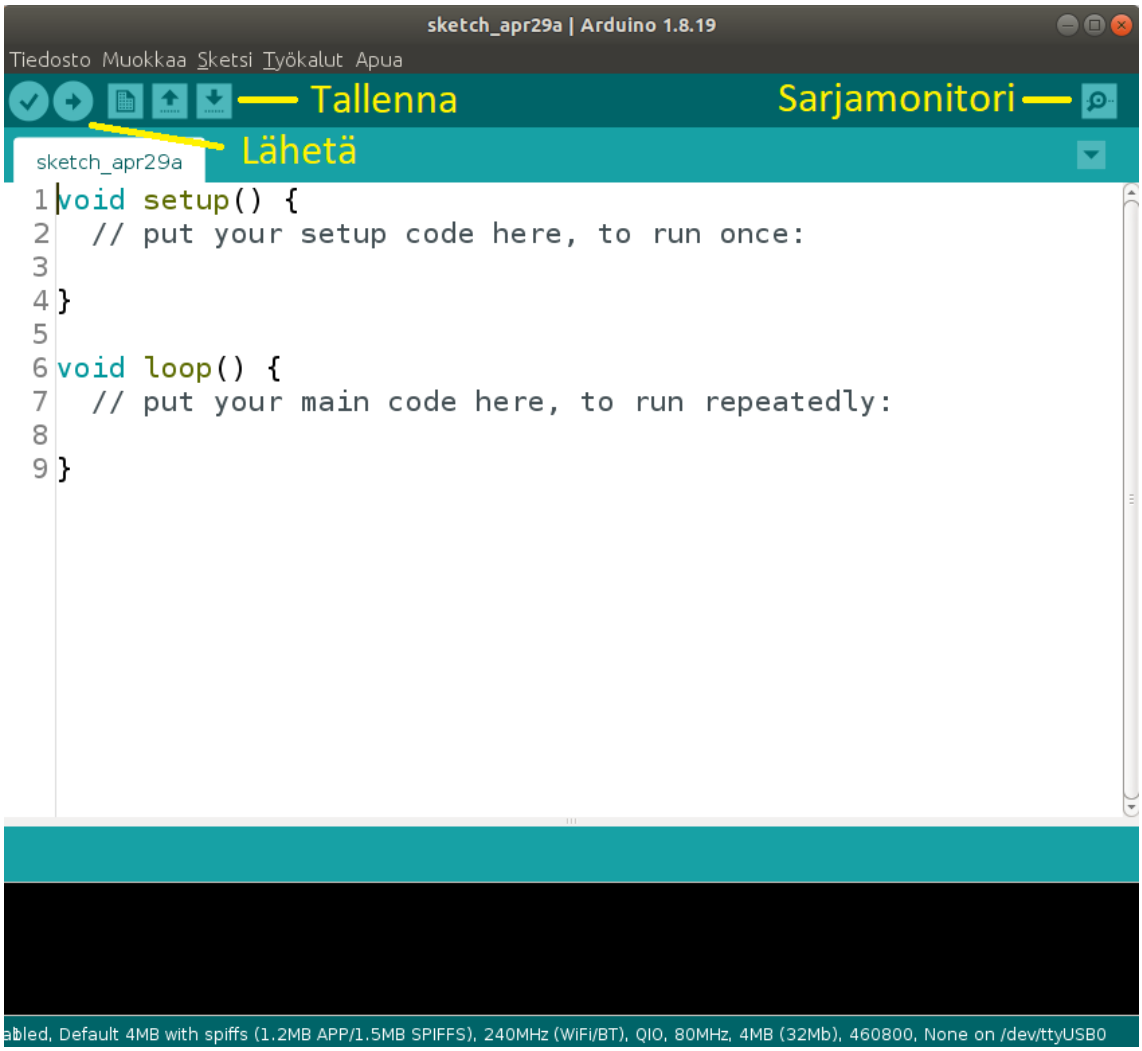
Paikannusjärjestelmän määritys alkoi ROS Melodic -kehitysympäristön asentamisesta. Ympäristö asennettiin robotiikkalaboratorion serverille/työasemalle (Acer Predator G3-710), joka sijaitsi Rantavitikan kampuksen kellarikerroksessa. Serverin käyttöjärjestelmänä toimi Ubuntu Linux 18.04 LTS. Serverille tehtiin aluksi ROS Melodic ja Arduino IDE -asennukset. Melodic-ympäristön asennuksissa käytettiin apuna Lapland Robotics GitHub-repositorion *miniATV* ohjeistusta ja vinkkejä. Melodic tarvitsi kaksi rosserial-pakettia, joista ensimmäinen oli nimeltään ros-melodic-rosserial-arduino ja toinen ros-melodic-rosserial. Lopuksi Arduino IDE:lle määritettiin rosserial-, TinyGPS- ja EspSoftwareSerial-kirjastot sekä ESP32-mikrokontrollerin tuki.

Arduino IDE on vapaasti jaettava ohjelmisto, jonka lähdekoodia jaetaan GPL-lisenssillä versionhallintaan käytetyllä sivustolla GitHubissa (Arduino 2007). Ohjelmistoa käytetään koodin kehitysympäristönä, kun kehityksen kohteena on jonkinlainen mikrokontrolleri. Ohjelmisto tukee muun muassa erilaisia versioita ESP- ja Arduino-mikrokontrollereista.

Jotta työssä käytetty ESP32 toimii Arduino IDE -ohjelmistolla toivotulla tavalla, täytyy ohjelmiston yläpalkista valita ”Työkalut->Kortti->ESP32 Arduino->ESP32

Dev Module”. Tämän jälkeen portin valinta täytyy tehdä manuaalisesti ohjelmiston yläpalkin kautta valitsemalla esimerkiksi ”Työkalut->Portti->/dev/ttyUSB0”.

Kuviossa 6 nähdään Arduino C -kielen käyttämät setup- ja loop-funktiot, painikkeet mikrokontrollerin koodin tallentamiselle ja lähettämiselle sekä sarjamonitorin painike datan näyttämiselle. Setup-funktio suoritetaan ohjelman aloituksessa kerran, ja funktiossa määritetään tavallisesti mikrokontrollerilta luettava sarjaliikenne sekä muut tarvittavat yhteydet. Loop-funktiota puolestaan suoritetaan koodissa toistuvasti. Näitä funktioita hyödynnetään myös lopullisessa toteutuksessa.



Kuvio 6. Näkymä Arduino IDE -ohjelmistosta ja tarvittavien toimintojen painikkeista

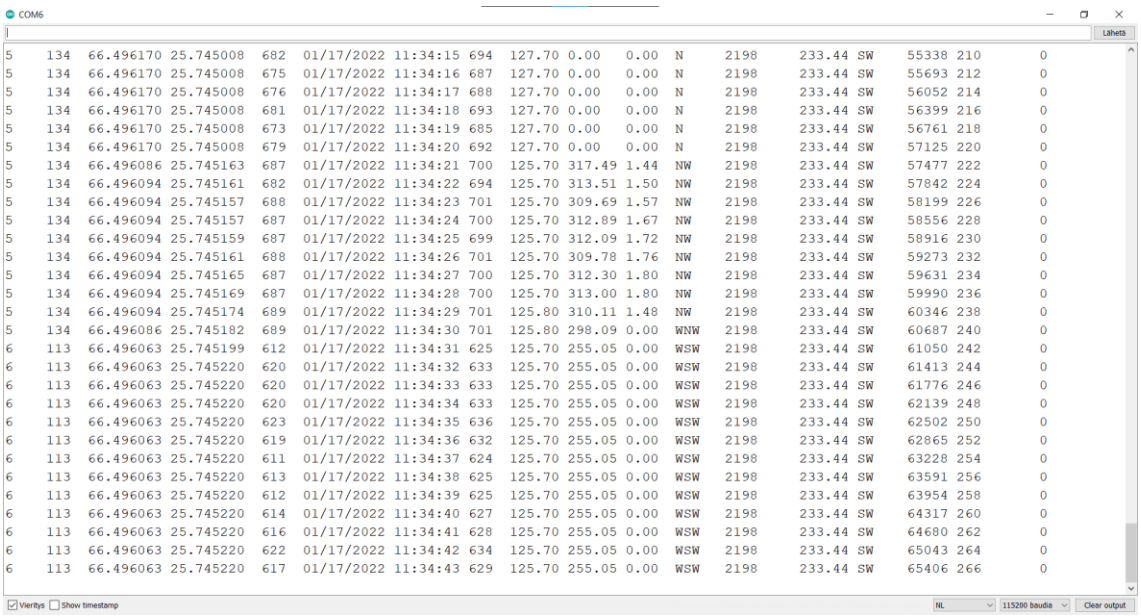
3.4 Mikrokontrollerin koodi

3.4.1 Esimerkkikoodin testaaminen ja ymmärtäminen

Grove GPS -moduulin testaaminen aloitettiin Mikal Hartin kehittämän ohjelman *test_with_gps_device.ino* avulla. Koodissa hyödynnetään TinyGPS-kirjastoa, joka osoittautui yhteensopivaksi Grove GPS -moduulille. Koodilla on mahdollista saada tietoa muun muassa satelliittien lukumäärästä, GPS:n tarkkuuden virheellisyydestä, pituus- ja leveysasteista, korkeudesta, kurssista, nopeudesta ja vastaanotetun tiedon statistiikasta. Jälkimmäisimmät statistiikkaan liittyvät arvot kuvaavat vastaanotettujen merkkien, merkkijonojen ja epäonnistuneiden merkkijonojen lukumäärää. Kun GPS-moduuli oli kytketty ESP32:een oikein, merkkien lukumäärä oli kasvavaa. Moduulin viestintään oli käytetty SoftwareSerial-kirjastoa, jolla saatiin luettua tietoa GPS-moduulista ESP32:n pinnien välityksellä. (Hart 2013.)

Kuviossa 7 alimman rivin paikannustiedot lukevat seuraavalla tavalla: ”6, 113, 66.496036, 25.745220, 617, 01/17/2022, 11:34:43, 629, 125.70, 255.05, 0.00, WSW, 2198, 233.44, SW, 65406, 266, 0”. On huomattava, että tyhjät välit on korvattu pilkuin. Toisin sanoen arvot ovat ilmaistu seuraavassa järjestyksessä: satelliittien lukumäärä, HDOP, leveysaste, pituusaste, age-muuttuja, päivämäärä, kellonaika, satelliitin age-muuttuja, korkeus (metreissä), kurssi GPS:stä katsottuna, nopeus (kilometriä tunnissa), ilmansuunta GPS:stä katsottuna, etäisyys, kurssi GPS:ään, ilmansuunta GPS:ään, vastaanotetut merkit, vastaanotetut lauseet, tarkistussumma.

Arvoista HDOP kuvaa GPS:n tarkkuuden virheellisyyttä, ja age-muuttujat puolestaan kertovat millisekuntien määrän datan koodaamisen jälkeisestä ajasta. Kaikkein viimeisimmäksi ilmoitettu arvo on tarkistussumma, joka kuvaa GPS-yhteyden aikana havaittuja virheitä. Tarkistussummasta on hyötyä muun muassa ohjelman nopeuden tarkkailussa. Jos esimerkiksi ohjelman tai satelliittien välinen yhteys toimii liian hitaasti tai nopeasti, summa kasvaa.



Kuvio 7. Näkymä GPS-moduulilla vastaanotetuista tiedoista

Ohjelman viiveeseen liittyvät ongelmat vältetään kuvion 8 smartdelay-funktion avulla. Funktiosta on huomattavasti hyötyä koodin ja GPS:n välisen toiminnan tauottamisessa, ja funktion käyttäminen on todettu testien aikana välttämättömäksi.

```
static void smartdelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (ss.available())
      gps.encode(ss.read());
  } while (millis() - start < ms);
}
```

Kuvio 8. Smartdelay-funktio ja GPS-arvojen lukeminen moduulista

3.4.2 Esimerkkikoodin muokkaaminen

Seuraavaksi Mikal Hartin kehittämään GPS-ohjelmaan tarvittiin WiFi-yhteys, jonka avulla paikkatiedon lähettäminen ESP32:sta langattomasti ROS-palvelimelle olisi mahdollista. Aikaisemmin yhteyden aikaansaamiseksi oli hyödynnetty micro USB -kaapelia, joka kytkettiin suoraan työasemaan. Toiminnallisuus langattomalle tiedonsiirrolle löytyi Agustin Nunezin kehittämästä ohjelmasta e*spros-wifi.ino*. Olennaisin osa koodista oli erikseen määritetty kuviossa 9 näkyvä WifiHardware-luokka, jota oli hyödynnetty NodeHandlen osalta. Esproswifi-koodi

tuki ROS-palvelimeen yhdistäviä rosserial -ja WiFi-suoritinkannan yhteyksiä. (Nunez 2017.)

```
class WiFiHardware {

  public:
  WiFiHardware() {};

  void init() {
    // do your initialization here. this probably includes TCP server/client setup
    client.connect(server, 11411);
  }

  // read a byte from the serial port. -1 = failure
  int read() {
    // implement this method so that it reads a byte from the TCP connection and returns it
    //  you may return -1 is there is an error; for example if the TCP connection is not open
    return client.read();         //will return -1 when it will works
  }

  // write data to the connection to ROS
  void write(uint8_t* data, int length) {
    // implement this so that it takes the arguments and writes or prints them to the TCP connection
    for(int i=0; i<length; i++)
      client.write(data[i]);
  }

  // returns milliseconds since start of program
  unsigned long time() {
     return millis(); // easy; did this one for you
  }

  protected:
    WiFiClient client;

};
```

Kuvio 9. WiFi-hardware-luokka

Vaikka ohjelma alkoi lupaavasti rakentumaan Esproswifin perustalle, ei tämä ollut vielä kuitenkaan valmis. Koska koodi oli alun perin suunniteltu vastaanottavaksi solmuksi, täytyi koodia muokata toimimaan lähettävänä solmuna. Lähettävään solmuun löytyi ratkaisu rosserialin mukana tulevasta ohjelmasta *TcpHelloWorld.ino*. Koodi muutettiin kuvion 10 tapaisesti *publisheriksi*, jonka avulla ESP32:een ladattu ohjelma muodosti myöhemmin yhteyden paikalliseen ROS-palvelimeen.

```
72 std_msgs::String str_msg;
73 ros::NodeHandle_<WiFiHardware> nh;
74 ros::Publisher chatter("testdata", &str_msg);
```

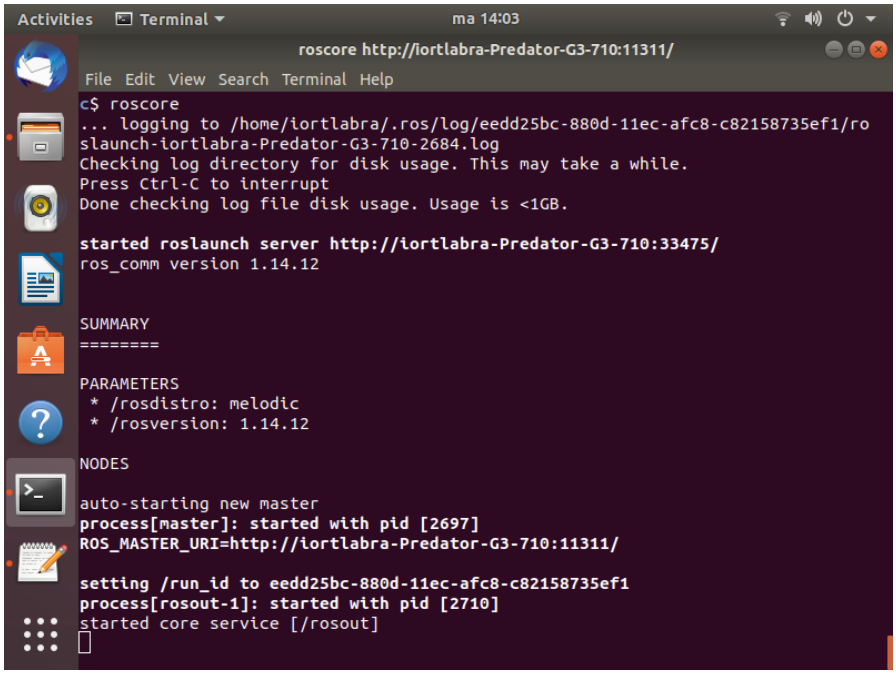
Kuvio 10. Koodin kohta, jossa tietoa julkaiseva toiminnallisuus on määritettynä

Yhteyden muodostamisessa käytettiin kuvion 11 NodeHandle-objektia *nh*. Yhteyden kautta ESP32 julkaisi paikannustietoja aiheeseen nimeltä *testdata*.

```
76 void setup() {
77   Serial.begin(115200);
78   delay(1000);
79   Serial.println("WiFi Setup");
80   setupWiFi();
81   delay(2000);
82   Serial.println("GPS module setup");
83   ss.begin(9600);
84   delay(2000);
85   Serial.println("GPS module setup done");
86   delay(2000);
87   Serial.println("init node");
88   nh.initNode();
89   Serial.println("init node done");
90   delay(10000);
91   nh.advertise(chatter);
92   Serial.println("\nSetup done");
93 }
```

Kuvio 11. Setup-funktio ja NodeHandle-objektin käyttö yhteyden aloittamiseksi

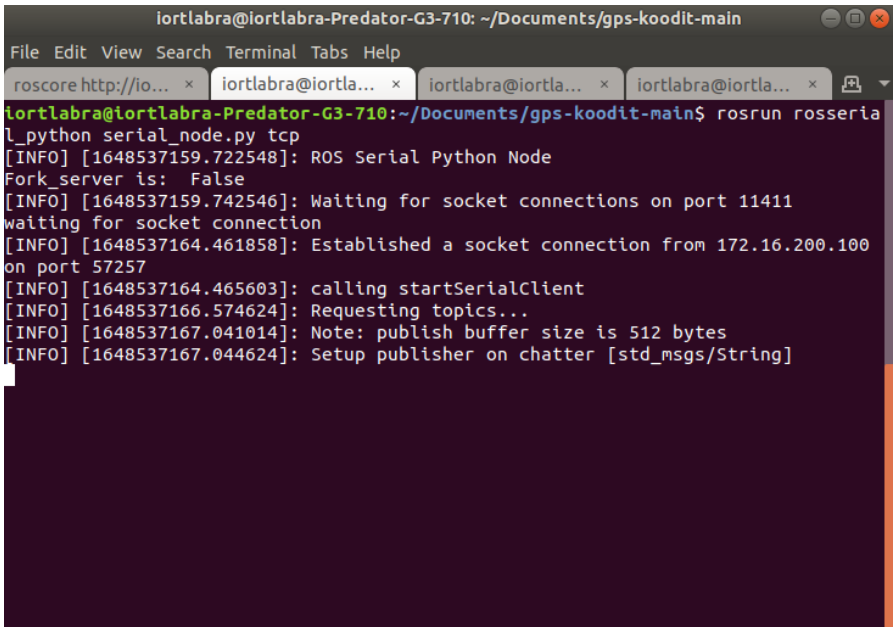
Palvelimessa suoritettiin samanaikaisesti komentoa *roscore*, kun mikrokontrollerin koodia testattiin. Komennon suoritus nähdään kuviossa 12. Roscoren suorituksen näkymässä tärkein tieto on kohta ”ROS_MASTER_URI”, joka toimii palvelimen osoitteena. Sekä palvelimen että ESP32:n täytyi olla yhdistettynä samaan verkkoon, jotta koodi toimisi oikein.



Kuvio 12. Näkymä palvelimella suoritetusta roscore-komennosta

Kuviossa 13 nähdään onnistunut rosserial-yhteys ESP32:n ja ROS-palvelimen välillä. Ensiksi käynnistyy Python-solmu, joka odottaa WiFi-suoritinkannan yh-

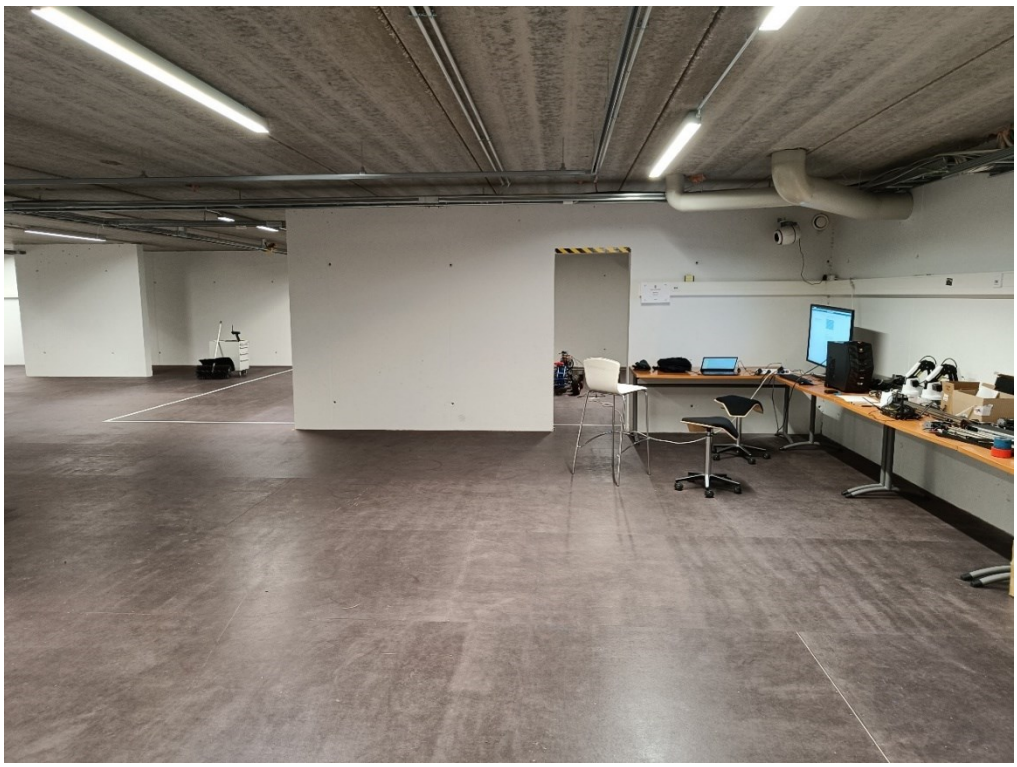
teyttä. Kun mikrokontrollerin yhteys on muodostettu verkon kautta ROS-palvelimelle, käynnistyy ROS-ympäristön *SerialClient*. Tämän jälkeen voidaan vastaanottaa aiheiden lähettämää tietoa.



Kuvio 13. Näkymä ESP32:een muodostetusta rosserial-/TCP-yhteydestä

3.5 Paikantamisen testaaminen

Kehitetyn paikannusjärjestelmän testaaminen tapahtui aluksi Rantavitikan kampuksen robotiikkalaboratoriossa, jotta GPS-signaalin vahvuudelle saataisiin sieltä varmuus. Robotiikkalaboratorio nähdään kuviossa 14. Signaalin kuuluvuus kellarissa oli erittäin huono, eikä yhteyttä satelliitteihin tämän takia saatu. Tulos oli oikeastaan ohjaajan kanssa odotettavissa, sillä signaalin heikentymistä rakenteiden ja muiden esteiden välillä voi tapahtua enemmän tai vähemmän riippuen GPS-laitteesta ja paikannettavasta sijainnista. GPS-signaalin heikon kuuluvuuden vuoksi sisäpaikantamiseen oli ennestään suunniteltu GPS-järjestelmän korvaavaa vaihtoehtoa, johon myöhemmin esitelty Marvelmind-laitteisto kykenee. Laitteiston etuna on hyödyntää GPS-signaalien sijaan ultraääneen perustuvaa teknologiaa.



Kuvio 14. Kampuksen kellarikerroksessa sijaitseva robotiikkalaboratorio

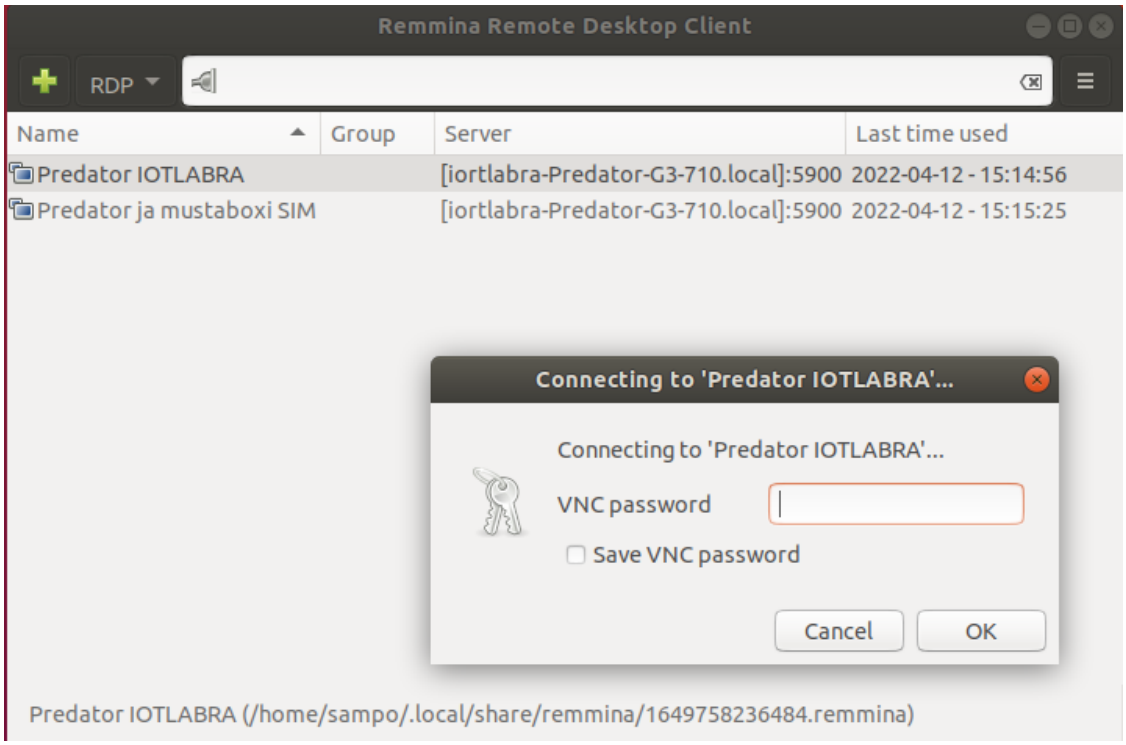
Robotiikkalaboratorion sijaan signaali kuului kuitenkin sisätiloissa, jos paikannin vietiin muualle esimerkiksi ikkunoiden läheisyyteen. Ikkunan luona signaalin saamiseen aikaa kului noin 1 – 5 minuuttia, joskus kuitenkin jopa 30 minuuttia.

Paikantamisen viiveeseen vaikuttavia tekijöitä ovat muun muassa signaalin taittuminen Maan ilmakehässä, satelliittikellot, vastaanottajat ja keskinäisesti epäedullinen satelliittien sijainti. Näiden lisäksi aikaisemmissa testeissä on havaittu, että paikantimen etäisyys muutamalla metrillä rakennuksen ikkunasta voi heikentää huomattavasti GPS-signaalia. (Henttu & Lehtoranta 1993, 59.)

Paikantimen testauksessa havaittiin aikaisemmin mainittujen seikkojen lisäksi, ettei ikkunan yläpuolella voinut olla suurta katosta. Kampuksen takapihalla oli katos, jonka läheltä löytyi paksuja lasi-ikkunoita. Näiden ikkunoiden kohdalla signaalia ei saatu. Paikanninta on testattu katoksen lisäksi muissa rakennuksissa, lähinnä kampuksen ulkopuolella. Esimerkiksi parvekkeen ikkunasta sekä lähellä olevista puista haittaa ei ole ollut GPS-signaalille.

3.5.1 Testaamisen tukena käytetyt ohjelmat

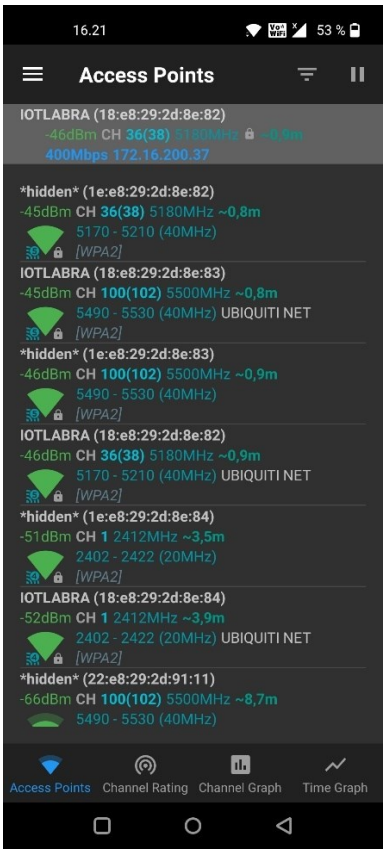
Myöhemmin paikantamisen testauksissa siirryttiin koulun takapihan alueelle. Ensiksi oli kuitenkin selvitettävä, miten voitaisiin aloittaa alusta serverillä suoritettavaa rosserial-yhteyttä mahdollisten katkosten varalta. Tulevissa testeissä käytettiin apuna ylimääräistä kannettavaa tietokonetta, johon oli asennettu sama palvelimen käyttämä Ubuntu-käyttöjärjestelmä ja Remmina-ohjelmisto. Remmina mahdollisti VNC-protokollan avulla etätyöpöytäyhteyden muodostamisen samassa verkossa olevaan palvelimeen. Kuviossa 15 nähdään Remmina-ohjelmiston päänäkymä ja luettelo määritetyistä yhteyksistä.



Kuvio 15. Remmina-ohjelmiston päänäkymä ja etätyöpöytäyhteyden muodostaminen VNC-protokollan avulla

Kun etätyöpöytäyhteys oli kunnossa, täytyi seuraavaksi aloittaa lähiverkon mittausten tekeminen Wifi Analyzer -mobiilisovelluksella. Sovellus ilmoitti langattoman verkon signaalin vahvuutta desibelimilliwattien avulla. Desibelimilliwatit kuvaavat sähkön tehon suhdetta yhteen milliwattiin (Kosatsky 2013, 10). Verkon kentän lukemat ilmoitetaan negatiivisina lukuina, ja suurempi arvo tarkoittaa aina

parempaa kentänvoimakkuutta (Botfel Estonia 2022, 9). Kuviossa 16 mobiiliso-vellus Wifi Analyzer näyttää luettelon muun muassa lähellä olevista langattomista verkoista ja verkkosignaalien vahvuuksista.



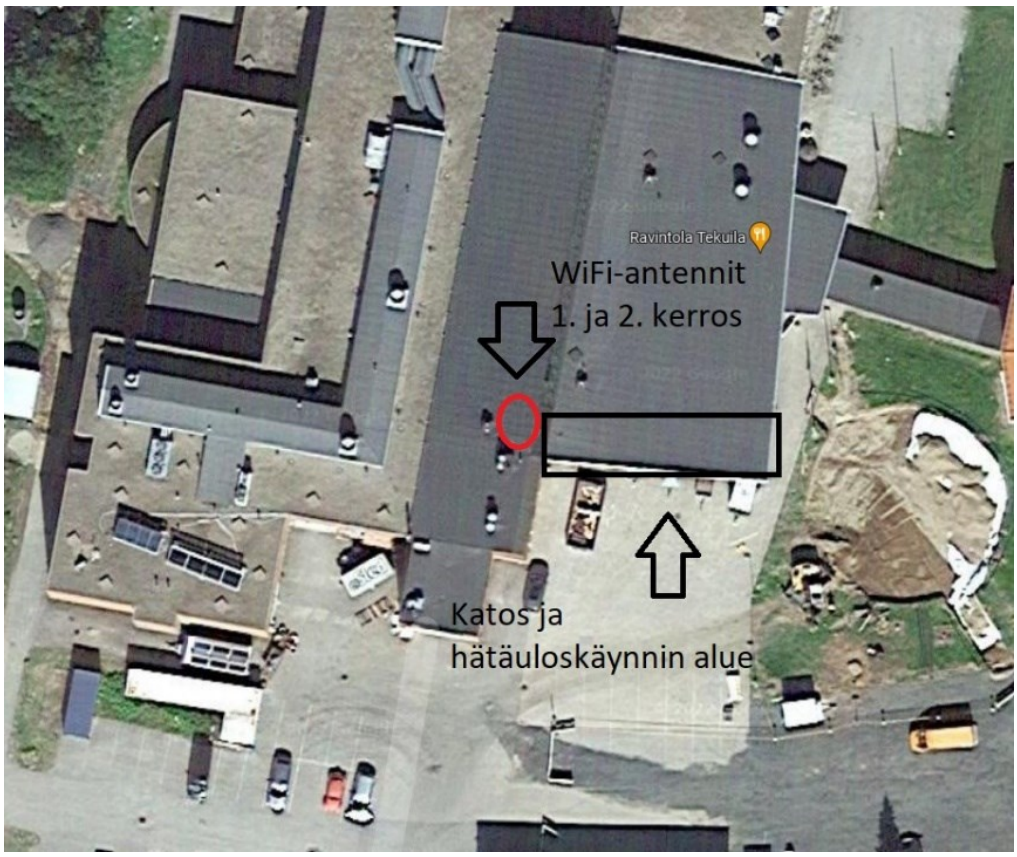
Kuvio 16. Näkymä WiFi Analyzer -mobiilisovelluksen mittauksista

3.5.2 Verkkoyhteyden jatkaminen WiFi-antennien avulla

Verkkosignaalin mittauksissa kävi ilmi, että signaali ei kantanut juuri ollenkaan ulos asti. Tämän vuoksi ohjaajan kanssa päätettiin, että verkon jatkeena kokeil-taisiin kahta Ubiquiti Networksin kehittämää UniFi AP AC PRO -antennia. Anten-nit olivat valmiiksi määriteltyjä IOTLABRA-verkkoon, johon myös ROS-palvelin oli alun perin yhdistettynä. Kuviossa 17 nähdään toinen käytetyistä antenneista.

Kuvio 17. Ubiquiti UniFi AP AC PRO -antenni

Antenni otti virran Power over Ethernet -tekniikalla (PoE) laitteen mukana tulevan muuntajan avulla verkkovirran kautta. Laitteeseen tuleva Ethernet-kaapeli kytkettiin laitteen Main-porttiin. Ensimmäinen antenneista oli kytkettynä ensimmäisen kerroksen hätäuloskäynnin luokse, joka sijaitsee pääaulan lähellä. Toinen antenni oli puolestaan kerrosta ylempänä. Molemmat antennit sijoitettiin kuvion 18 mukaisesti kampuksen ikkunoiden läheisyyteen. Antennien jakama verkkoyhteys toimi mesh-periaatteella, joka jatkaa kunkin antennin kohdalla verkon signaalia. Signaalin jatkaminen ei ollut kuitenkaan tarpeeksi varma vaihtoehto, koska pihalle signaalia ei kuulunut juuri ollenkaan.



Kuvio 18. UniFi-antennien sijainnit B-osan aulassa (Google Maps 2022)

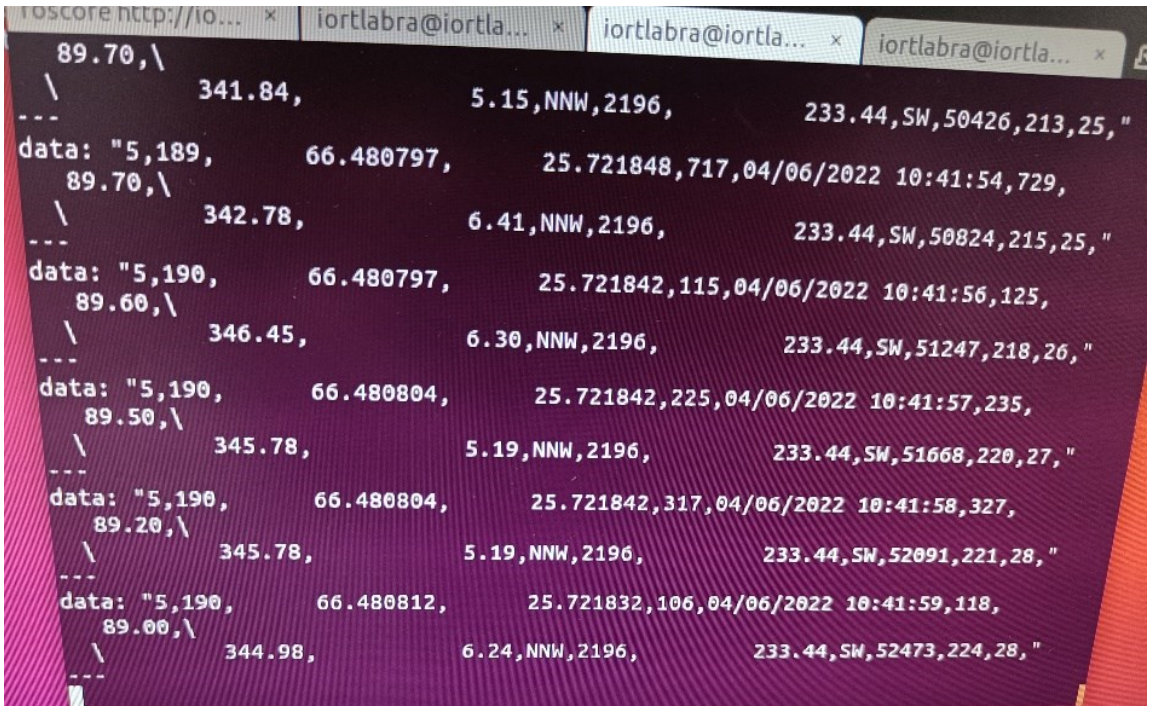
3.5.3 Erillinen verkkoyhteys

UniFi-antennien heikon verkkosignaalin vuoksi kokeilussa oli seuraavaksi ohjaajan mainitsema LR54-reititin. Kuvion 19 reititin jakoi langatonta verkkoyhteyttä SIM-kortin avulla, joten mikrokontrollerin koodia oli päivitettävä käyttämään samaista verkkoa. Myös etätyöpöytäyhteys kellarin koneelle täytyi määrittää uudelleen. Reitittimellä oli tarkoitus tehdä kolme erilaista testausta, joista kerrotaan seuraavaksi tarkemmin.

Kuvio 19. LR54-reititin

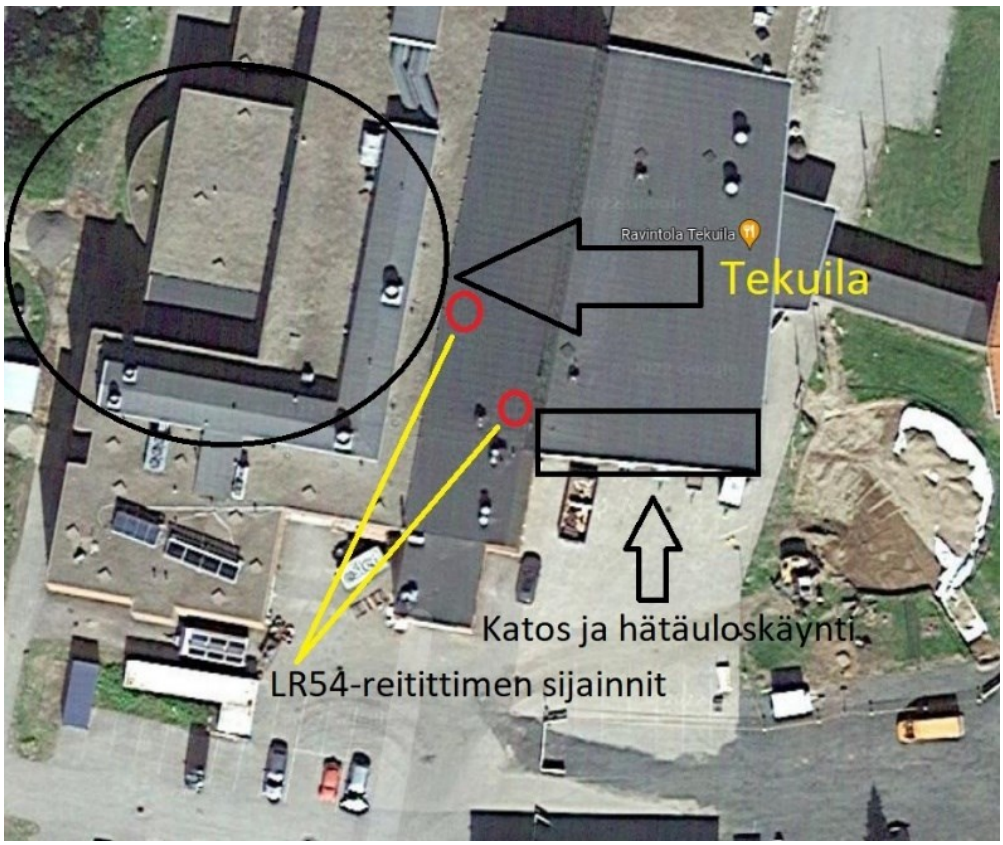
Ensimmäisessä kokeilussa reititin oli hätäuloskäynnin ovien sisäpuolella B-osan aulassa. Verkon signaali oli heikko kellarin palvelimella. Palvelin löysi verkon WiFi-hakuluetteloon, mutta ei pystynyt yhdistämään verkkoon ilman työaseman siirtämistä muutamalla metrillä kellarin tiloissa. Sen sijaan hätäuloskäynnin ovien luona signaali oli vahvuudeltaan -57 – -88 desibelimilliwattia. Signaalin arvona noin -60 desibelimilliwattia oli vielä kohtuullinen GPS-yhteyden testaamiselle, mutta -90 desibelimilliwattia oli vahvuudeltaan jo niin heikko, ettei yhteyttä ole juuri ollenkaan. Koulun takapihalle oli kuitenkin niin heikko signaali, ettei GPS-moduulin kanssa ollut mahdollisuutta kävellä ulko-ovien läheisyydestä montaa metriä kauemmas. Tämä kokeilu oli periaatteessa epäonnistunut, koska datan täytyy päästä kulkemaan kellarin palvelimelle asti.

Toisessa testissä kellarin palvelin oli siirretty hätäuloskäynnin luokse ulko-ovien lähelle. Verkon signaali oli vahva, ja GPS-moduulin sekä ESP32:n vieminen katoksen luokse oli mahdollista korkeintaan viiden metrin etäisyyteen. GPS-signaali löytyi ulkona minuutin aikana, jolloin dataa alkoi muodostua rosserialin kautta palvelimelle. Kuviossa 20 nähdään kampuksen takapihalla sijaitsevasta GPS-moduulista vastaanotettua paikkatietoa ROS-palvelimella.



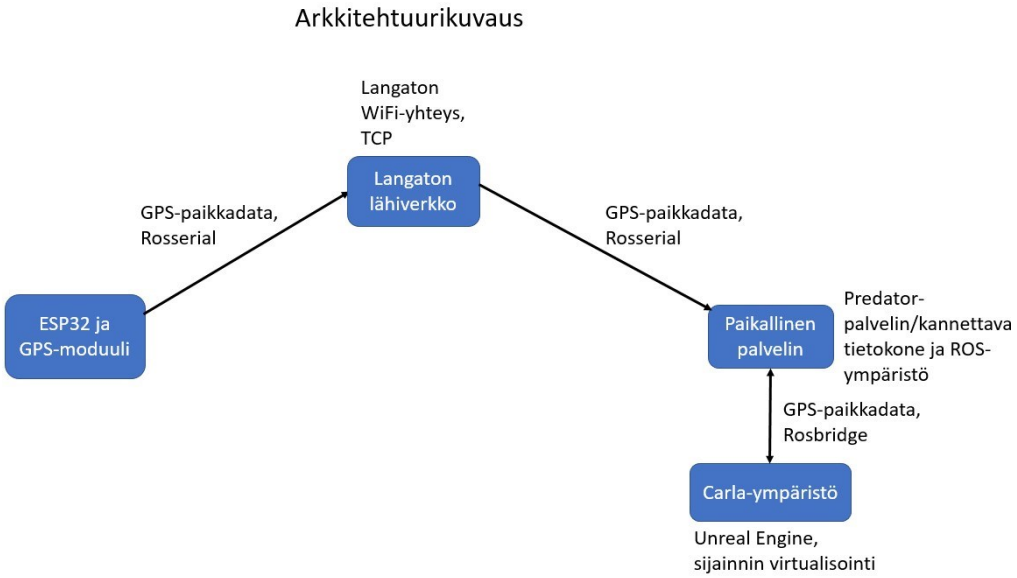
Kuvio 20. GPS-moduulista vastaanotettu data

Kolmannessa kokeessa reititin sijaitsi kuvion 21 mukaisesti Tekuila-ravintolan edustalla pääaulan käytävällä ja palvelin puolestaan kellarin tiloissa. GPS-moduuli ja ESP32 pysyivät ulkona katoksen läheisyydessä. GPS:n luona kannettavan tietokoneen WiFi-kuvake näytti WiFi-signaalille puolta palkkia, mutta WiFi Analyzer antoi signaalille arvon -82 desibelimilliwattia. Yhteys toimi kohtuullisesti kellarin palvelimelle asti. Edelleenkään koulun takapihan alueella ei pystynyt etenemään ulko-ovien luota viittä metriä kauemmaksi tai yhteyden laatu olisi muuten heikentynyt huomattavasti ja yhteys olisi jopa katkennut kokonaan. Kampuksen takana WiFi Analyzer näytti signaalin arvona -90 desibelimilliwattia. Tässä testissä kuitenkin GPS-moduuli löysi satelliitit suurin piirtein minuutin aikana, kuten aikaisemmin mainitussa testissä.



Kuvio 21. LR54-reitittimen sijainnit (Google Maps 2022)

Kuviossa 22 on esitetty aikaisemmin mainittujen testien arkkitehtuurikuvaus. Arkkitehtuurikuvausta käytettiin testiympäristön havainnollistamisessa.



Kuvio 22. GPS-paikantimen ja ROS-järjestelmän testaus kampuksen ympäristöissä

3.6 Paikkatiedon lähettäminen Carla-ympäristöön

Sijainnin lähettäminen Carla-ympäristöön oli ajankohtaista, kun GPS-paikkatiedon saaminen oli testattu takapihan alueelta onnistuneesti. Aluksi oli asennettava Carla-ympäristö, joka toimi Unreal Enginen puolella. Carlan ja ROS-ympäristön välillä käytetään tiedonvälitykseen protokollaa, jonka nimi on Rosbridge. Asentamisen ja takapihan mallintamisen oli toteuttanut Lapin AMK:n asiantuntija Tuomas Herranen, joka opasti Carlan käyttöönotossa ja soveltamisessa.

Carla-ympäristöä käytetään yleisesti autonomisten ajojärjestelmien kehittämisessä. Simulaatioympäristössä on mahdollista testata muun muassa autonomisen alustan antureista saatavaa tietoa. (CARLA Team 2022.) Tulevaisuudessa hankkeen alustojen paikkatietoa on tarkoitus käyttää virtuaalisen sijainnin määrittämiseen. Käytännössä tämä merkitsee digitaalisen kaksosen konseptia, jossa alustan todellisia toimintoja simuloidaan virtuaaliympäristössä.

Ympäristön käyttöönotossa oli hyödynnettävä export -ja source-komentoja, jotta Carlaan ajettavat komennot alkaisivat toimimaan mahdollisimman nopeasti ja työtä päästäisiin jatkamaan eteenpäin. Tarvittavat komennot suoritettiin jokaisessa Ubuntun terminaalin välilehdessä, jolle oli pienintäkään tarvetta. Näitä tekniikoita ei kuitenkaan tarvitse käyttää joka kerta ympäristön käyttöönoton yhteydessä, jos ROS-ympäristöstä on luotu työtila tai tiedot saa jotenkin muulla tavalla pysymään terminaalin ulottuvilla. Export-komennoissa määritettiin lähinnä Carlan juurikansio, sen käyttämän Python-jakelun polku sekä kansio Python-rajapinnalle/-koodeille.

Kuviossa 23 nähdään ROS-ympäristön määrityksessä käytetyt komennot. Kohdat 1 ja 2 määrittävät tarvittavien tiedostojen polut, joita komentotulkki tarvitsee Carlan käyttöönottoa varten. Kohta 3 määrittää palvelimen yhteyteen liittyvät osoitteet ja portin. Neljännen kohdan komento käynnistää Carla-ympäristön, ja viidettä kohtaa käytetään solmun *carla_manual_control* käynnistämiselle. Viides kohta perustuu launch-tiedostoon, jonka sisältö on XML-formaatissa. Launch-tiedostoon määritetään käynnistettävät ROS-ympäristön solmut (ROS 2022e). Myös kannettavalla tietokoneella täytyi määrittää terminaalissa ROS-palvelimen asetukset vastaamaan Predator-koneen IP-osoitetta.

```
Export:

1.
export CARLA_ROOT=/home/iortlabra/carla export
PYTHONPATH=$PYTHONPATH:$CARLA_ROOT/PythonAPI/carla/dist/carla-0.9.13-py2.7-linux-
x86_64.egg:$CARLA_ROOT/PythonAPI/carla

2.
export PYTHONPATH=$PYTHONPATH:/home/iortlabra/carla/PythonAPI/carla/dist/carla-0.9.13-
py2.7-linux-x86_64.egg

3.
export ROS_MASTER_URI=http://<YOUR_IP_ADDRESS>:11311
export ROS_HOSTNAME=<YOUR_IP_ADDRESS>


Source:

4. Unreal Engine startup In Carla folder
make launch

5. Launch file for carla_manual_control node
roslaunch carla_ros_bridge carla_ros_bridge_with_example_ego_vehicle.launch
```

Kuvio 23. Carlan käyttöönoton yhteydessä terminaalin välilehtiin ajettavat komennot

Seuraavassa kuviossa nähdään Carla-ympäristö, jossa on virtualisoitu Rantavitikan kampuksen takapihan alue. Kuvion pienemmässä ikkunassa on editorin kameran HUD-arvoja, johon tullaan myöhemmin sijoittamaan paikkatietoa.

Kuvio 24. Näkymä asennetusta Carla-ympäristöstä, jossa suoritetaan Python-solmua *carla_manual_control*

3.6.1 Tiedonvälityksen selvittäminen

Datan lähetystä testattiin aluksi esimerkkikoodeilla, jotka oli kirjoitettu Python-ohjelmointikielellä. Esimerkkikoodeja oli mahdollista testata joko kahden laitteen välillä tai pelkästään Predator-työasemalla. Testeissä kuvion 25 dataa lähettävä publisher-koodi *talker.py* toimi aluksi kannettavalla tietokoneella. Koodissa määritetty *chatter* toimi aiheen nimenä ROS-järjestelmässä. Solmun *talker* uudelleennimeämiselle ei ollut tarvetta.

```
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Kuvio 25. Talker.py-koodi

Kuviossa 26 dataa vastaanottavaa subscriber-koodia *listener.py* suoritettiin Predator-koneen ROS-palvelimella. Aiheen nimenä toimi *chatter*, mutta solmun uudelleennimeämiselle ei ollut tarvetta.

```python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + 'I heard %s', data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber('chatter', String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

Kuvio 26. Listener.py-koodi

Seuraavaksi kokeiltiin alkuperäisiä kooditiedostoja ilman aiheen uudelleennimeämistä. Kun Ubuntun komentotulkkiin saatiin vastaanotettua lähetettävää tietoa, oli selvää, että yhteys laitteiden välillä toimi. Samalla saatiin varmuus siitä, että solmuja ei tarvitse nimetä uudelleen, mutta aiheiden nimet on oltava samat sekä tietoa vastaanottavassa että lähettävässä koodissa. Testit pystyttiin toteuttamaan tämän lisäksi pelkästään Predator-työasemalla, jolloin kannettavalle tietokoneelle ei ollut tarvetta.

Kun datan lähettäminen ja vastaanottaminen Python-esimerkkikoodien avulla oli selvitetty, siirryttiin testeissä hieman aidompaan vaiheeseen. Tässä vaiheessa tiedon lähettäminen tuli onnistua Carla-ympäristöön asti. Tällöin Python-koodissa *talker* oli muutettava tietoa julkaisevan aiheen nimi vastaamaan vastaanottavan aiheen nimeä kuvion 27 mukaisesti, jotta tiedon vastaanottaminen onnistuisi solmussa *carla_manual_control*. Myös solmun uudelleennimeäminen oli tarpeen tämän Python-koodin puolella.

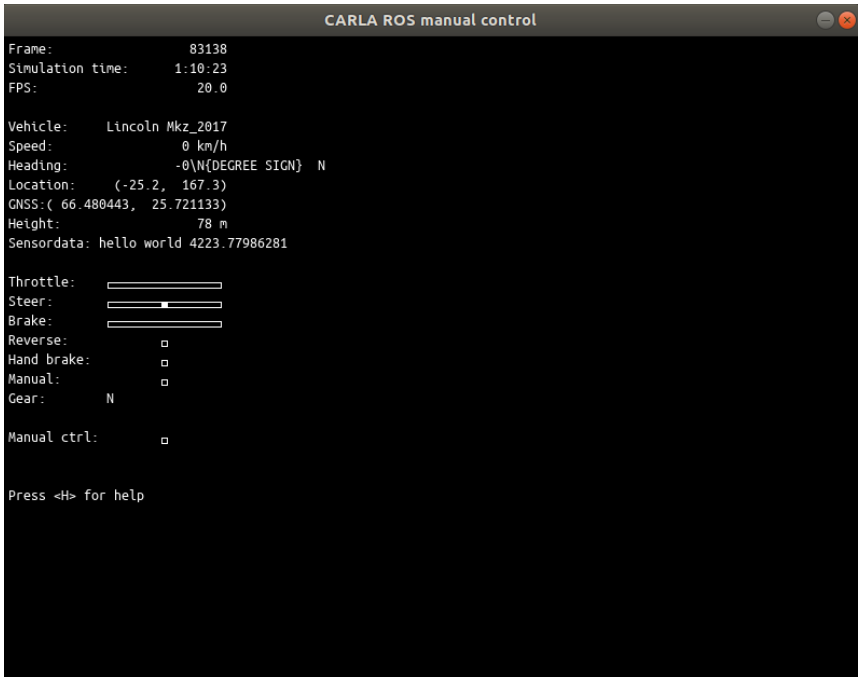
talker.py × original.py

home > sampo > talker.py > ...

```
41
42  import rospy
43  from std_msgs.msg import String
44
45  def talker():
46      pub = rospy.Publisher('testdata', String, queue_size=10)
47      rospy.init_node('carla_manual_control')
48      rate = rospy.Rate(10) # 10hz
49      while not rospy.is_shutdown():
50          testdata = "hello %s" % rospy.get_time()
51          rospy.loginfo(testdata)
52          pub.publish(testdata)
53          rate.sleep()
54
55
56  if __name__ == '__main__':
57      try:
58          talker()
59      except rospy.ROSInterruptException:
60          pass
61
```

Kuvio 27. Kuvassa on muokattu kooditiedosto *talker.py*, joka lähettää dataa solmuun *carla_manual_control* aiheen *testdata* avulla

Carlassa vastaanotettua tietoa käytettiin HUD-arvoissa kuviossa 28 näkyvän *Sensordatan* kohdalla eli arvot näkyivät tekstinä Unreal Editorin kameran näkökentässä.



Kuvio 28. Näkymä kameran HUD-kentästä, jonka kohtaan *Sensordata* on lähetetty testattavaa tietoa

3.6.2 Tiedonvälityksen vianmäärittäminen

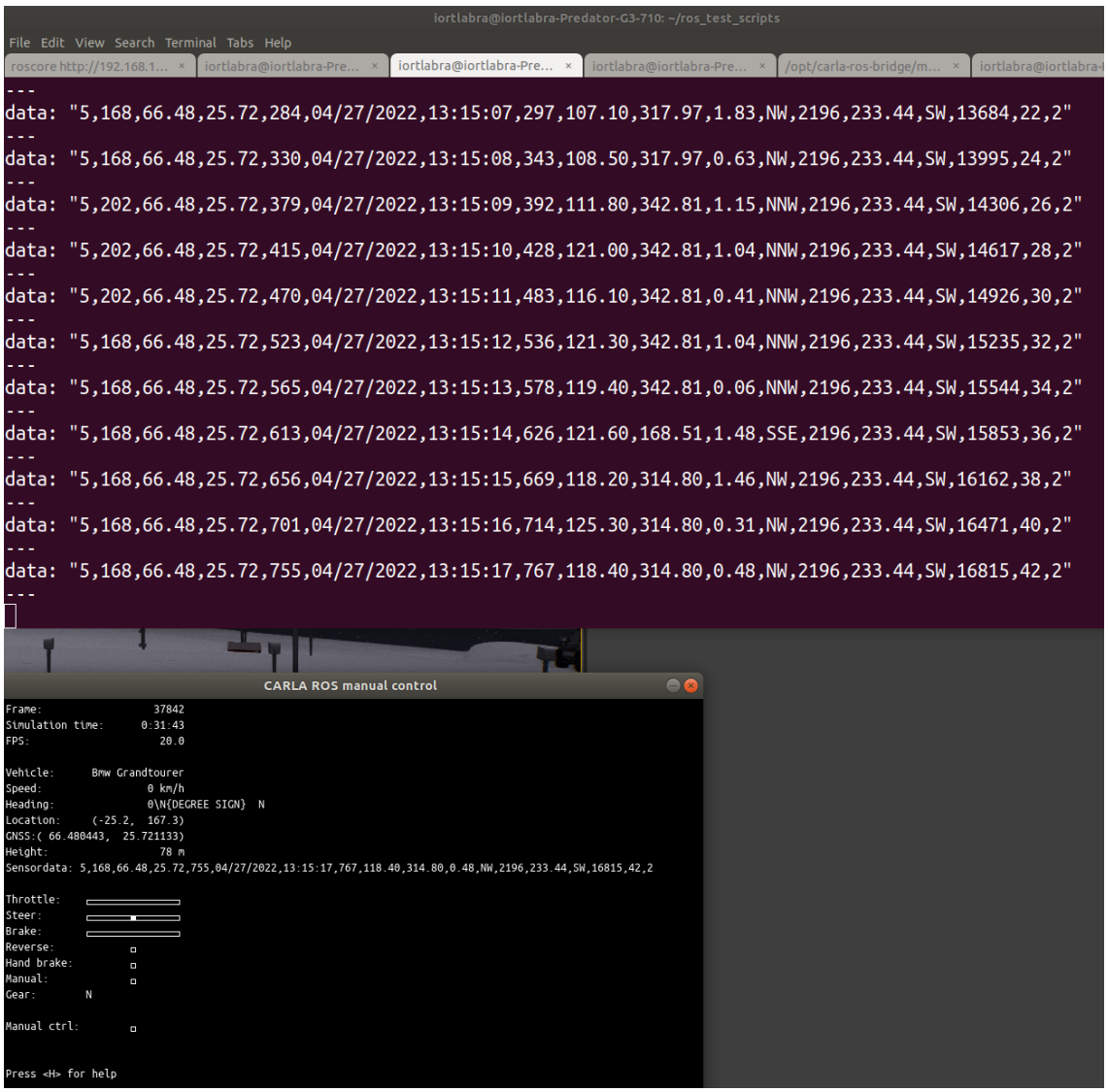
Seuraavaksi datan lähetys oli saatava vastaamaan ESP32:n kautta GPS-moduulista saatavaan tietoon. Esproswifi-koodissa oli aikaisemmin ollut ongelmana NodeHandlen tarjoama IP-osoite ”0.0.0.0”, josta ilmeni häiriöitä. Ongelmaan löytyi ratkaisu kokeilemalla Roscoren uudelleenkäynnistystä, mutta ratkaisuun auttoi myös lukuisat pienet muutokset Esproswifi-koodin puolella. Ongelmaa selvitettiin koululla työskentelevän asiantuntijan kanssa. Koodiin tuli muutoksia muun muassa WiFiHardware-luokkaan sekä WiFi-yhteyden ja NodeHandlen käynnistyksen järjestyksiin Setup-funktiossa.

Setup-funktion rakennetta nähdään tarkemmin kuviossa 29. Ensiksi kuviossa määritetään ESP32:n baudinopeus, sitten kutsutaan WiFi-yhteyden määrittämiseen käytettyä funktiota. Kun WiFi-yhteys on määritetty, siirrytään GPS-moduulin baudinopeuden määrittämiseen. GPS-moduulin jälkeen käynnistetään ROS-järjestelmään tarvittava solmu. Lopuksi määritetään aihe, jonka lähetyksestä vastaa NodeHandle-objekti. Eri vaiheet sisältävät muutaman sekunnin taukoja, mutta viimeiselle vaiheelle on varattu varmuuden vuoksi 10 sekuntia aikaa. Tauot on ilmoitettu millisekunneissa.

```
76 void setup() {
77   Serial.begin(115200);
78   delay(1000);
79   Serial.println("WiFi Setup");
80   setupWiFi();
81   delay(2000);
82   Serial.println("GPS module setup");
83   ss.begin(9600);
84   delay(2000);
85   Serial.println("GPS module setup done");
86   delay(2000);
87   Serial.println("init node");
88   nh.initNode();
89   Serial.println("init node done");
90   delay(10000);
91   nh.advertise(chatter);
92   Serial.println("\nSetup done");
93 }
```

Kuvio 29. Setup-funktion yhteyksien järjestys ohjelman käynnistyksessä

NodeHandlen yhteysongelman ratkettua oli varmistettava vielä kampuksen takapiha-alueelta, miltä GPS-moduulin data näyttäisi oikeassa tilanteessa. Kuviossa 30 nähdään takapihalta tulevan GPS-moduulin tietoa Ubuntun terminaalissa ja Carla-ympäristössä.



Kuvio 30. Terminaalissa varmistettu piha-alueelta vastaanotettu GPS-paikkatieto

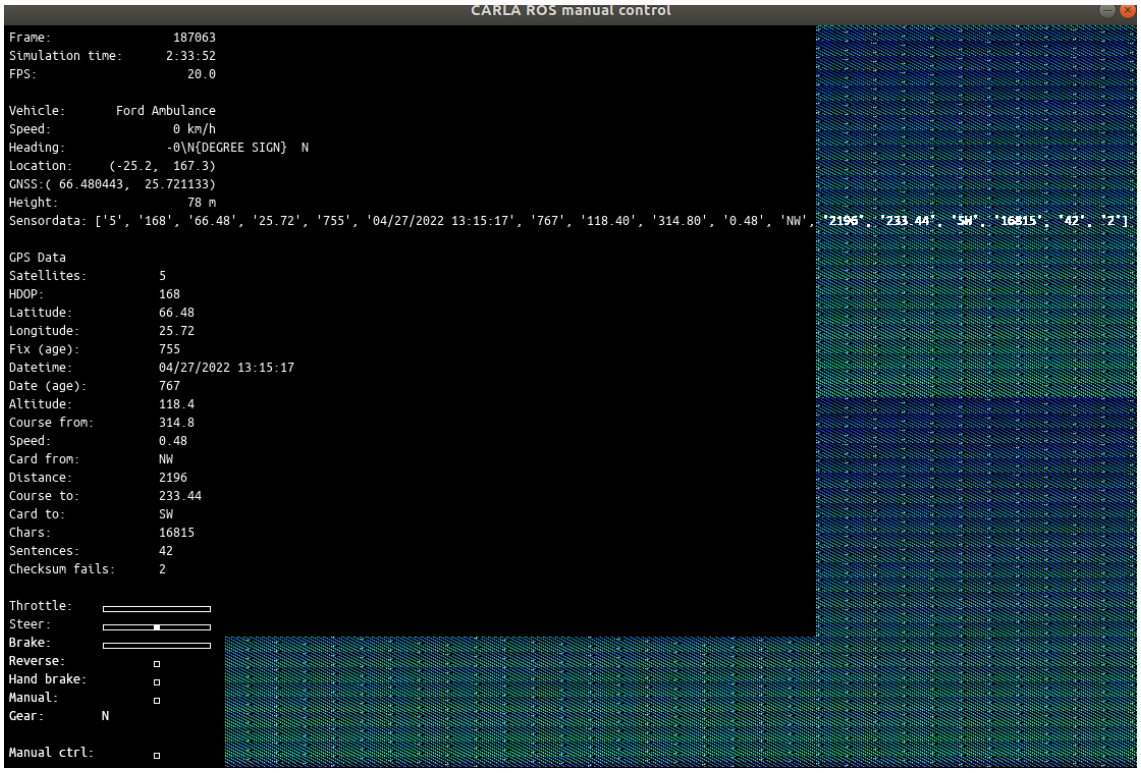
3.6.3 Tiedonvälityksen testaaminen

Kun Esproswifi-ohjelma tuki sekä rosserial -, WiFi -ja GPS-moduulin yhteyksiä, oli koodiin lisättävä lähetettävää staattista tietoa ja toiminnallisuus datan erottamiselle pilkuilla. Kuvion 31 staattisella tiedolla oli mahdollista työstää helposti datan muotoa Carla-ympäristössä, koska GPS-moduulia ei hyödynnetty koodin työstämisen aikana. Tieto oli käytännössä heti saatavilla keinotekoisesti. Solmun *carla_manual_control* muokkaus onnistui lopulta hyvin, ja testattavan datan avulla saatiin eroteltua GPS-moduulista saatavaa tietoa Carlan HUD-arvoihin.

```
124  strcat(gpsData, "5,168,66.48,25.72,755,04/27/2022 13:15:17,767,118.40,314.80,0.48,NW,2196,233.44,SW,16815,42,2");
125  delay(2500);
126  Serial.println(gpsData);
```

Kuvio 31. Strcat-komennolla liitetty kokeiltava data

Kuviossa 32 nähdään Carla-ympäristössä vastaanotettu keinotekoinen paikkatieto. Paikkatieto näkyy kuvassa Python-listana *Sensordatan* kohdalla ja lajiteltuna tietona *GPS Datan* kohdalla.



Kuvio 32. Testattavan tiedon muotoilu kohdan *GPS Data* alapuolella

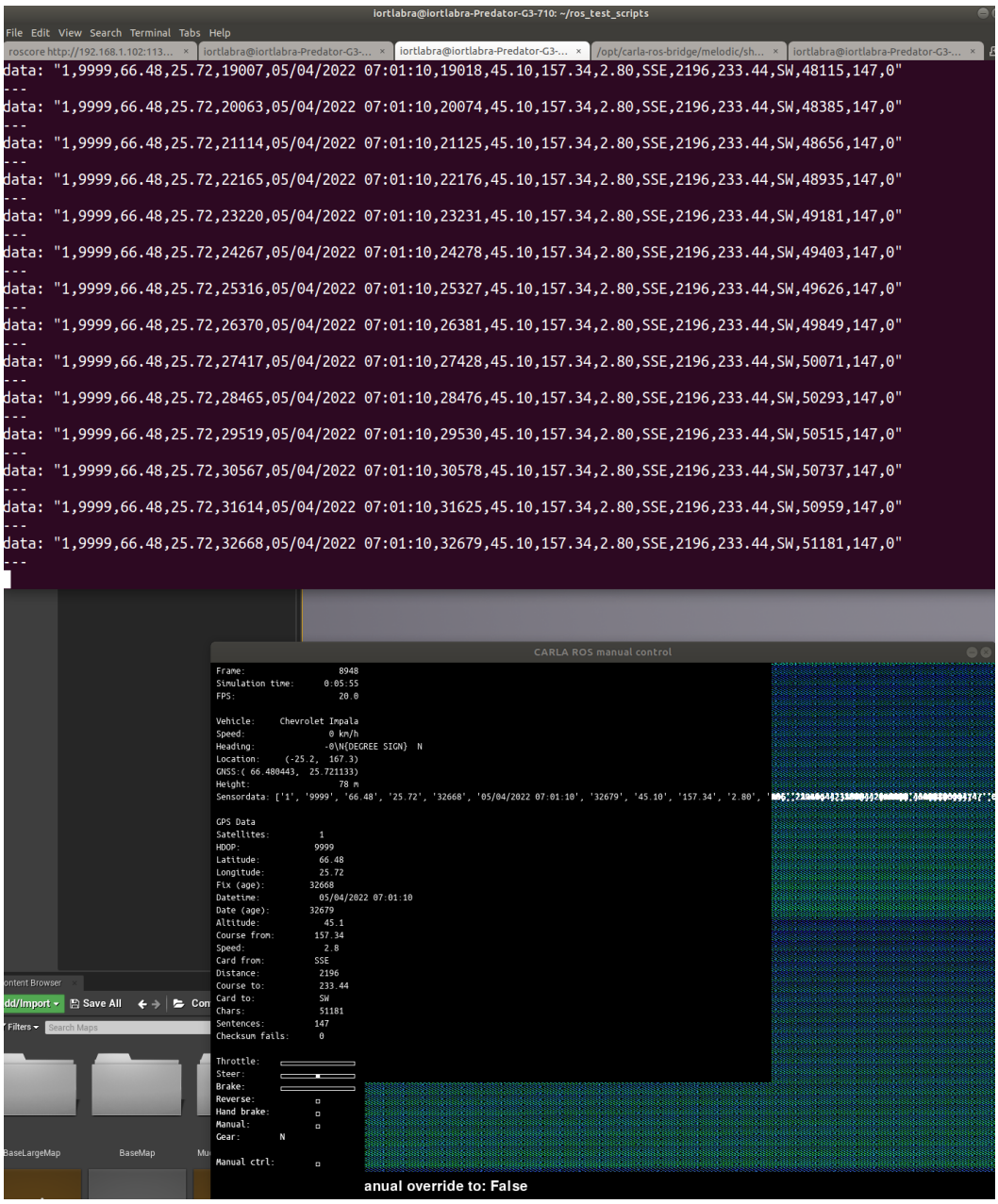
Pilkuilla erotettu data saatiin aikaan muokkaamalla Esproswifin koodia. Koodin loppuun lisättiin funktio print_separator kuvion 33 mukaisesti, jota kutsuttiin aina muiden print-funktioiden yhteydessä. Muuttujan *separatorVal* arvoksi asetetaan luku 17, koska erityyppisiä print-funktioita kutsutaan loop-funktion aikana 16 kertaa, joista print_date -funktion sisällä käytetään vielä kerran print_int -funktiota. Muut print-funktiot tallensivat erilaisilla tyypeillä ilmoitettua paikkatietoa tekstinä array-muuttujaan *gpsData*, joka toimi tiedon keräämisessä. Myös turhia välejä poistettiin koodista, jotta GPS-moduulista tuleva data näkyisi jatkossa selkeämmin ja tietoa olisi helpompi käsitellä Carla-ympäristössä.

```
void print_separator() {
  if (separatorVal > 0) {
    strcat(gpsData, ",");
  }
  else if (separatorVal <= 0)
  {
    separatorVal = 17;
  }
}
```

Kuvio 33. Print_separator-funktio

3.6.4 Lopullisen tiedonvälityksen testaaminen

Kun tiedon muotoileminen oli valmista keinotekoisen datan avulla, tehtiin viimeiset testit vielä aidolla paikkatiedolla. Kuvion 34 paikkatiedot olivat testaushetkellä hieman heitteleviä, joka johtuu todennäköisesti vähäisestä satelliittien lukumäärästä. Muotoiltu tieto olisi näkynyt paljon siistimmin HUD-arvojen näkymässä, jos heittelevät arvot eivät olisi olleet kovin suuria.



Kuvio 34. Esproswifi-ohjelman ja Carlan testaaminen aidolla paikkatiedolla

4 SISÄPAIKANNUSJÄRJESTELMÄ

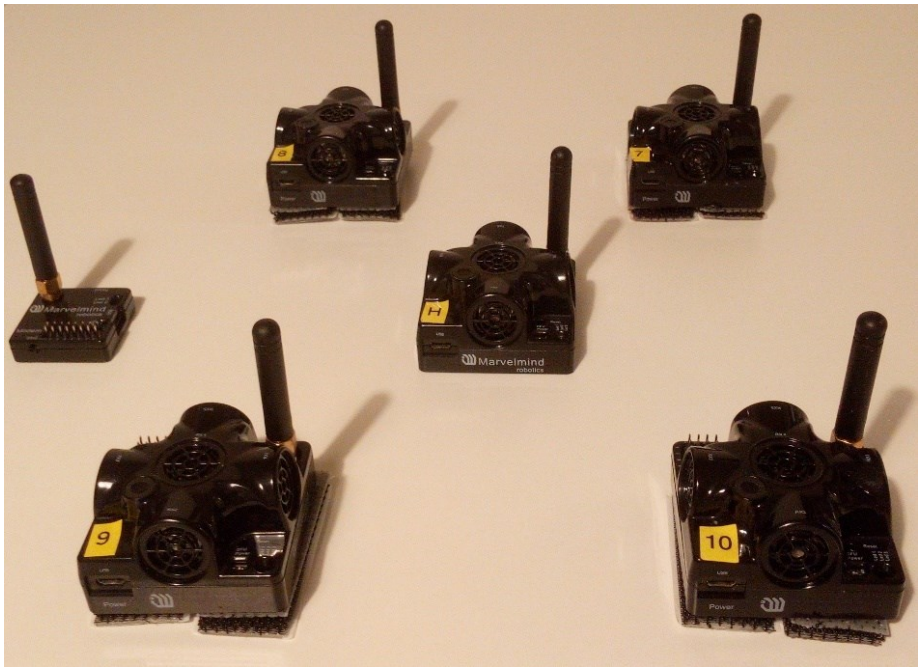
Marvelmind-laitteiston käyttämä langaton radioteknologia perustuu ultraääneen. Ultraäänen lyhenteenä käytetään kirjaimia *UWB*. Ultraääni on ääntä, jonka taajuuden alarajaksi on määritetty noin 20 kilohertsiä. Alarajasta korkeampia ääniä pystyvät kuulemaan enimmäkseen eläimet. Ultraääntä hyödynnetään muun muassa kaikuluotauksessa, hammaskiven poistossa ja lääketieteen suorittamassa ihmisen kehon kuvantamisessa. (Oulun Ultra Oy 2022.)

Alun perin Marvelmind-laitteisto oli ollut ennestään kampuksella kokeilussa. Koska järjestelmän liikkeen tarkkailussa oli havaittu häiriöitä, päätettiin järjestelmää kokeilla vielä uudelleen. Majakoiden testialue robotiikkalaboratoriossa oli merkitty valkoisella viivalla, joka muodosti neliön muotoisen alueen. Paikallaan olevat majakat asetettiin alueen jokaiseen kulmaan, ja liikkuvaa majakkaa liikutettiin alueen sisäpuolella. Kuviossa 35 nähdään kannettava tietokone, jonka lähellä sinisessä ympyrässä oleva modeemi saa micro USB -kaapelin kautta virtaa. Liikuteltava majakka Hedge löytyy keltaisen ympyrän kohdalta kannettavan tietokoneen vierestä.

Kuvio 35. Majakoiden testialue robotiikkalaboratoriossa (808 x 631 cm)

4.1 Kokoonpanon ominaisuudet

Aloituspaketti Marvelmind Starter Set Super-MP-3D, jonka Marvelmind Robotics on suunnitellut, sisältää viisi majakkaa ja yhden modeemin. Majakoista neljä kappaletta on paikallaan olevia majakoita, joista viides majakka Hedge toimii kannettavana majakkana. Kuviossa 36 vasemmalla oleva pienin laite on reititin/modeemi, kuvan keskellä on liikuteltava Hedge ja muut Hedgen ympärillä olevat majakat ovat paikallaan olevia majakoita.



Kuvio 36. Marvelmind-laitteisto, 5 majakkaa ja modeemi

Majakat lähettävät tietoa tavallisesti suoraan modeemille. Tietojen lähettäminen majakoiden osalta vaihtelee sen mukaan, onko majakoiden sijainteja määritetty ennestään vai määritetäänkö sijainteja parhaillaan. Sijainnit määritetään *submapissa* eli alikartassa joko manuaalisesti tai automaattisesti. (Marvelmind Robotics 2020.)

Laitteiston tarkkuudeksi on ilmoitettu keskiarvollisesti noin ± 2 senttimetriä. Alikarttojen avulla on mahdollista kattaa majakoita niin suurelle alueelle kuin tarvitaan. Yksittäisen alikartan avulla laitteiston kantamaksi voidaan saada jopa 30 metriä. (Marvelmind 2020, 4.)

Monilla ultraääneen perustuvilla järjestelmillä on monesti korkea hinta, ja järjestelmät voivat maksaa jopa tuhansia euroja. Marvelmind-laitteiston hinta oli 500 euroa, joka oli kuitenkin todettu laitteiston hankinnan yhteydessä kohtuulliseksi.

4.2 Arkkitehtuurien väliset erot

Molemmissa arkkitehtuureissa tuetaan kaiken kaikkiaan 250:tä yhtäaikaista majakkaa. Lukumäärä sisältää sekä kannettavat että paikallaan olevat majakat. (Marvelmind Robotics 2020.)

4.2.1 Ei-käänteinen arkkitehtuuri

NIA-arkkitehtuurissa Hedge-majakoita suositellaan käytettäväksi yhtäaikaisesti 1–4 kappaletta, jotka voidaan jakaa autonomisiin alustoihin käytettäväksi. Kannettava majakka asetetaan alustaan mukaan. Paikallaan olevien majakoiden on oltava hiljaisemmissa paikoissa, ettei ultraäänelle aiheudu häiriöitä paikantamisen aikana. (Marvelmind 2020, 4.)

Kun majakoiden kartta on rakennettu, kannettava majakka lähettää ultraääntä ja paikallaan olevat majakat vastaanottavat ultraääntä. Jos karttaa rakennetaan parhaillaan, modeemi hallitsee jokaista majakkaa. Tällöin majakat lähettävät ja vastaanottavat ultraääntä ja modeemi kerää jokaisen majakan tiedot etäisyyksistä. Ultraäänen lähetys ja vastaanotto riippuu kuitenkin laitteiston firmwaren versiosta ja itse laitteistosta. On muistettava, että tässä arkkitehtuurissa jokainen majakka toimii samalla taajuudella. Tavallisesti majakat käyttävät 31 kilohertsiä. (Marvelmind Robotics 2020.)

4.2.2 Käänteinen arkkitehtuuri

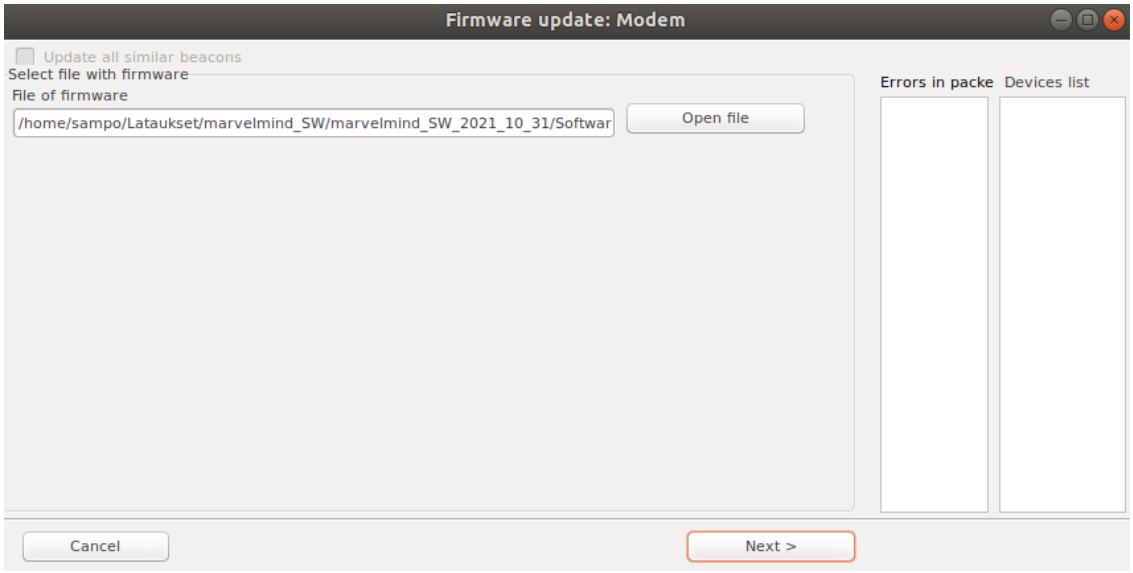
IA soveltuu tilanteisiin, joissa kannettavat majakat ovat hiljaisemmissa paikoissa. Käyttötilanteen alustaksi soveltuu myös drone, vaikka kannettavat majakat vastaanottavat tietoa. Kannettavan majakan kantama dronella on kuitenkin melko rajallinen, noin 2–5 metriä. (Marvelmind 2020, 4.)

Aikaisempaan arkkitehtuuriin verrattuna IA kykenee toimimaan monimutkaisemmissa struktuureissa. Tässä arkkitehtuurissa paikallaan olevat majakat lähettävät

ultraääntä ja kannettavat majakat vastaanottavat ultraääntä. Toisin sanoen arkkitehtuuri tukee monta samanaikaista kannettavaa majakkaa. Arkkitehtuurin jokainen majakka käyttää eri taajuutta 19–45 kilohertsin väliltä. (Marvelmind Robotics 2020.)

4.3 Määritys

Laitteiston määrityksessä käytettiin ohjelmistoa nimeltä Dashboard, jonka oli kehittänyt Marvelmind Robotics. Alussa laitteita kytkettiin micro USB -kaapelin kautta yksitellen tietokoneelle, jossa Dashboard pyysi päivittämään laitteistopohjaisen ohjelmiston eli firmwaren kunkin laitteen kohdalla ajan tasalle. Tähän tarvittava firmware-päivittäjä nähdään kuviossa 37. Laitteiden päivitysten yhteydessä valittiin ensiksi NIA-arkkitehtuuriin perustuva firmware, koska NIA on helposti käyttöönotettavissa johtuen majakoiden samoista taajuusalueista ja radiokanavista.

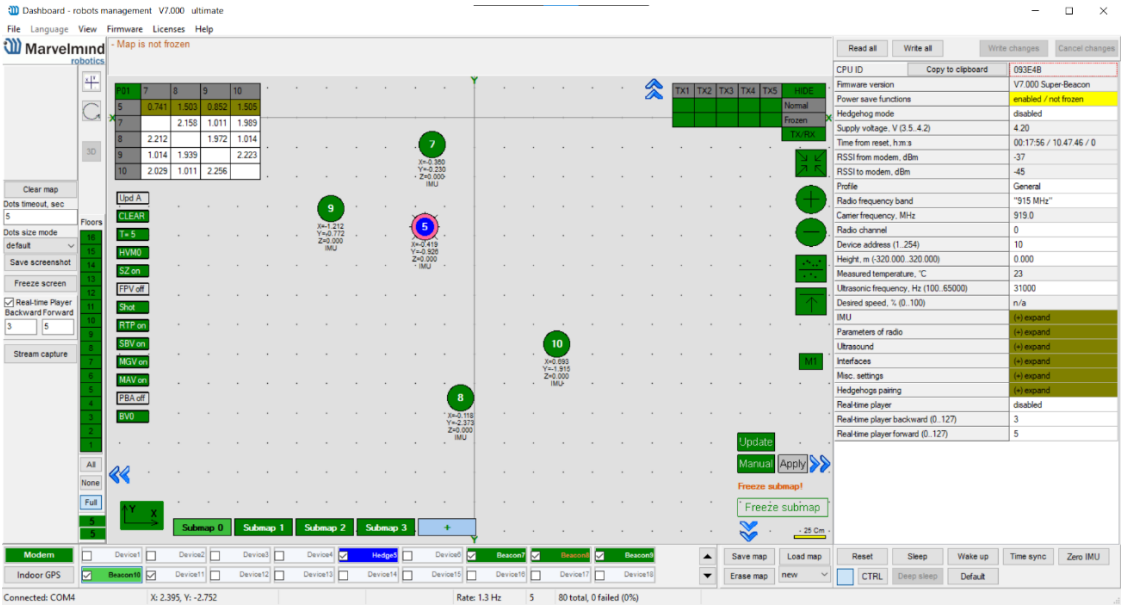


Kuvio 37. Laitteiston firmware-päivittäjän näkymä

Kun päivitykset oli tehty laitteisiin, valittiin Dashboard-ohjelmassa *Submap 0* ja päällä olevat majakat herätettiin ohjelman alapalkista numeroinnin mukaan. Majakat olivat numeroituina 7–10, ja Hedge-majakka löytyi numerona 5. Majakoiden sijainnit määritettiin ohjelman toiminnolla *update*. Sijainnin määrityksessä pystyi hyödyntämään auto ja manual moodeja ohjelman yläkentän ohjeiden mukaisesti.

Sijainnin määrittämisen pystyi jäädyttämään ohjelman toiminnolla *freeze submap*. Alun perin kaikkiin majakoihin oli määritettynä ultraäänentaajuus 31 kilohertsiä ja radiotaajuusalueeksi 915 megahertsiä. Radiokanava oli määritetty puolestaan arvoksi 0 jokaisen majakan kohdalla.

Seuraavassa kuvassa nähdään majakoiden määritetty tila. Ohjelmiston oikeanpuoleisesta palkista pystytään määrittämään tai tarkistamaan majakoiden arvoja.

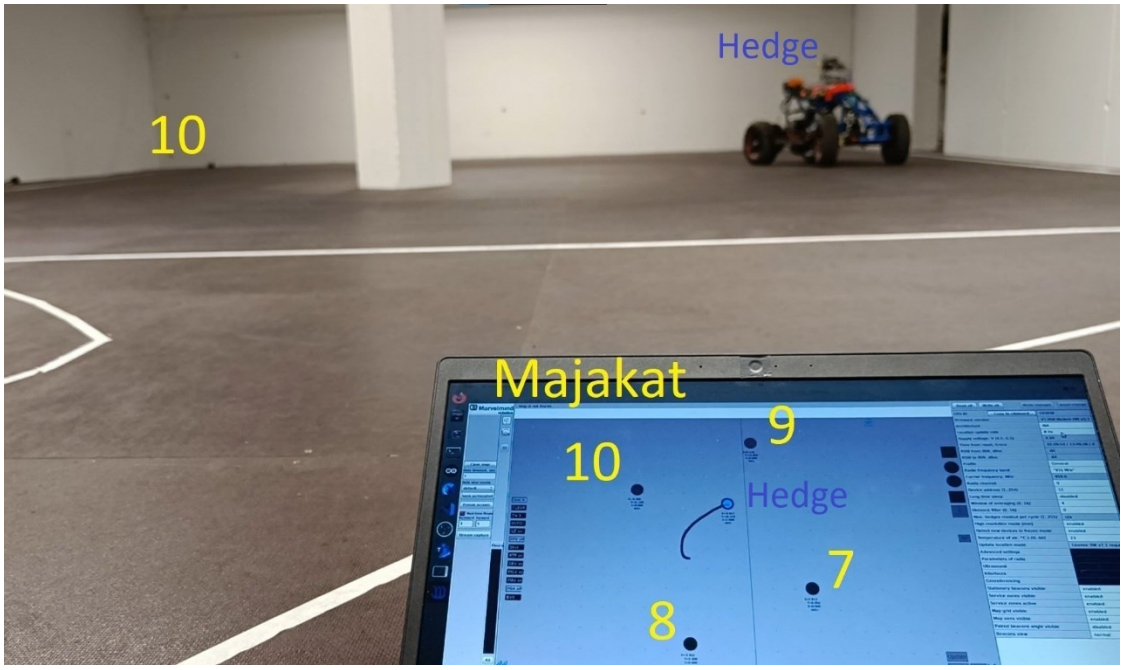


Kuvio 38. Näkymä Dashboard-ohjelmistosta ja NIA-arkkitehtuuriin määritetyistä majakoista

Kun varmuus majakoiden toiminnasta oli saavutettu, siirryttiin majakoita testaamaan robotiikkalaboratorion testiympäristöön. Kaikki neljä majakkaa löytyivät Dashboardista, ja Hedge-majakka liikkui ohjelman näkymässä muutaman sekunnin viiveellä. Erityisempiä ongelmia kokeilun aikana ei tullut vastaan. Kuviossa 39 nähdään tilanne, jossa Hedge-majakkaa liikutettiin kävelemällä. Majakka on liikkunut kuvion mukaan rajatun testialueen rajoilla.

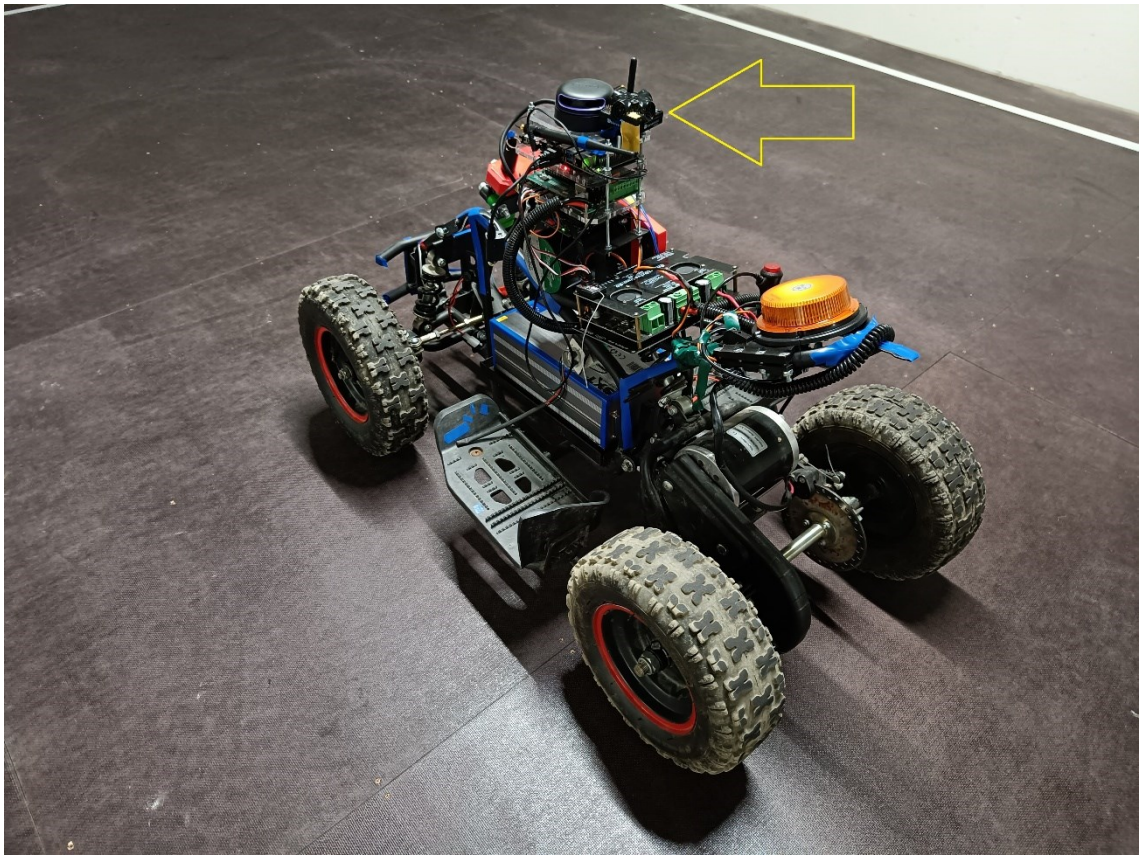
Kuvio 39. Robotiikkalaboratoriossa suoritettu NIA-arkkitehtuurin testaus

Myöhemmin Hedge-majakan liikuttamista kokeiltiin miniATV-alustalla. Alapuolella sijaitsevasta kuviosta selviää majakoiden numerot ja sijainnit. Kuviosta voidaan todeta Hedgen liikkuvan Dashboard-ohjelman näkymässä pienellä viiveellä. Hedgen takaa löytyy paikallaan olevista majakoista numero 9. Tätä majakkaa vastapäätä on puolestaan numero 7. Kuvaajasta vasemmanpuoleisin majakka on numero 8.



Kuvio 40. Kauko-ohjattavan miniATV-alustan seuranta Hedge-majakan avulla

Kuviossa 41 nähdään miniATV-alustaan asetettu Hedge-majakka. Majakka oli tilapäisesti kiinnitetty alustan korkeimpaan kohtaan.



Kuvio 41. Kiinnitetyn Hedge-majakan paikka alustassa

Myöhemmin kokeilussa oli IA-arkkitehtuuri. Kaikkiin majakoihin ladattiin IA-arkkitehtuuriin perustuvat firmware-ohjelmat, mutta majakoiden löytämisessä ilmeni ongelmia. Suurin osa majakoista katosi Dashboardin näkymästä. Ongelmia kartoitettiin vaihtamalla majakoiden radiotaajuutta ja radiokanavia sekä ultraääneen käytettyjä taajuuksia. Ultraääneen käytetyt taajuudet asetettiin Marvelmindin julkaiseman arkkitehtuurien vertailuun tarkoitetun PDF-tiedoston tietojen avulla. Testissä vain yksi majakoista näkyi Hedgen lisäksi.

5 POHDINTA

Opinnäytetyön aikana Ubuntun tukemat ympäristöt ja komentotulkissa suoritettavat komennot tulivat tutuiksi. Kehittämisen näkökulmasta ROS-ympäristö toimi uutena kokemuksena robotiikan osalta. GPS-moduulin toiminnan testaaminen oli alussa haastavaa, koska yhteyden saaminen satelliitteihin saattoi kestää yllättävän kauan. Ylimääräiset vastaanottimen testaukset olivat tarpeen, jotta vastaanottimen toimivuudelle saatiin varmuus.

Projektissa eniten aikaa kului GPS-vastaanottimen koodin kehittämiselle. Vaikka Internetissä oli valmiita koodin pohjia saatavilla, oli niiden soveltaminen ja toimivuuden testaaminen aikaa vievää ja tarkkuutta vaativaa. ESP32:n käyttämä Arduino C -kieli eroaa perinteisestä C-kielestä monella eri tavalla. Tähän syynä on lähinnä se, että Arduino C -kieli on suunniteltu sulautetuille järjestelmille ja mikrokontrollereille perinteisten tietokoneiden sijaan.

Projektiin ei ollut saatavilla Pozyxin valmistamia laitteita. Aikataulun vuoksi Novatel-paikannusjärjestelmä oli jätettävä pois testattavista paikannusjärjestelmistä. Kaiken kaikkiaan projektissa saatiin selvyydeksi Marvelmindin ja mikrokontrollerin hyödyntämän GPS-paikantimen toimivuudet, joista seuraavilla kehittäjillä on helpompi jatkaa paikannusjärjestelmien kehittämistä.

Ulkopaikannuksen parannusehdotuksena on käyttää ulkokäyttöön soveltuvia verkkoantenneja, jotta langattoman verkon signaalia saataisiin vahvemmaksi. Paremmin kuuluvasta verkkosignaalista on huomattavasti hyötyä alustojen paikannusjärjestelmän jatkokehityksessä.

LÄHTEET

Arduino 2007. GitHub. license.txt. Viitattu 29.4.2022 https://github.com/arduino/Arduino/blob/master/license.txt.

Botfel Estonia 2021. Operaattorivertailu. Selvitys Suomen 5G verkkojen kuuluvuudesta. Viitattu 18.4.2022 https://static.elisa.com/v2/image/2tqybbhjs47b/Im50m7H4Mdm0xJ08RHRh1/Boftel_Vertailumittaus_12-2021.pdf.

CARLA Team 2022. Introduction. Viitattu 3.5.2022 https://carla.org/.

Espressif Systems 2022. ESP32-WROOM-32D & ESP32-WROOM-32U Datasheet. Viitattu 28.4.2022 https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf.

GeeksforGeeks 2021. Source command in Linux with Examples. Viitattu 24.4.2022 https://www.geeksforgeeks.org/source-command-in-linux-with-examples/.

Google Maps 2022. Satelliittikuva Rantavitikalta. Viitattu 18.4.2022 https://www.google.com/maps/@66.4808355,25.7217383,79m/data=!3m1!1e3.

Hart, M. 2013. GitHub. Test_with_gps_device. Viitattu 23.4.2022 https://github.com/mikalhart/TinyGPS/blob/master/examples/test_with_gps_device/test_with_gps_device.ino.

Henttu, P. & Lehtoranta, V. K. 1993. GPS. Maailmanlaajuinen satelliittinavigointijärjestelmä. Opas uuteen aikaan.

Kosatsky, T. 2013. British Columbia Centre for Disease Control. Viitattu 26.4.2022 http://www.bccdc.ca/resource-gallery/Documents/Educational%20Materials/EH/Radiofrequency-Toolkit.pdf#page=14.

Logistiikan Maailma 2022. Esineiden Internet. Viitattu 26.5.2022 https://www.logistiikanmaailma.fi/logistiikka/digitalisaatio/esineiden-internet/.

Maanmittauslaitos 2022. Häiriöitä satelliittipaikannuksessa? Yleisimmät syyt GNSS-häiriöihin. Viitattu 25.5.2022 https://www.maanmittauslaitos.fi/ajankohtaista/hairioita-satelliittipaikannuksessa-yleisimmat-syyt-gnss-hairioihin.

Marvelmind 2020. Architectures comparison. Viitattu 23.4.2022 https://marvelmind.com/pics/architectures_comparision.pdf.

Marvelmind Robotics 2020. Help: Inverse Architecture (IA) vs. Non-Inverse Architecture (NIA). Viitattu 24.4.2022 https://www.youtube.com/watch?v=dwEj1koHAdM.

National Coordination Office for Space-Based Positioning, Navigation, and Timing 2021. Other Global Navigation Satellite Systems (GNSS). Viitattu 25.4.2022

https://www.gps.gov/systems/gnss/#:~:text=GLONASS%20(Global-naya%20Navigazionnaya%20Sputnikovaya%20Sistema,system%20con-sists%20of%2024%2B%20satellites.

National Coordination Office for Space-Based Positioning, Navigation, and Tim-ing 2022. Space Segment. Viitattu 10.4.2022 https://www.gps.gov/sys-tems/gps/space/.

Nunez, A. 2017. GitHub. Esproswifi. Viitattu 23.4.2022 https://github.com/agnu-nez/espros/blob/master/examples/esproswifi/esproswifi.ino.

Oulun Ultra Oy 2022. Ultraääni: mitä se on ja mihin sitä käytetään. Viitattu 17.4.2022 https://oulunultra.fi/ultraaani/.

ROS 2022a. ROS - Robot Operating System. Viitattu 18.4.2022 https://www.ros.org/.

ROS 2022b. Nodes. Viitattu 23.4.2022 https://wiki.ros.org/Nodes.

ROS 2022c. Rosbridge. Viitattu 3.5.2022 http://wiki.ros.org/rosbridge_suite.

ROS 2022d. Roscore. Viitattu 23.4.2022 https://wiki.ros.org/roscore.

ROS 2022e. Roslaunch. Viitattu 3.5.2022 http://wiki.ros.org/roslaunch.

ROS 2022f. Rosserial. Viitattu 23.4.2022 https://wiki.ros.org/rosserial.

ROS 2022g. Topics. Viitattu 23.4.2022 https://wiki.ros.org/Topics.

Seeed Technology 2021. Grove GPS. Viitattu 28.4.2022 https://wiki.seeedstu-dio.com/Grove-GPS/.

LIITTEET

Liite 1. GPS-vastaanottimen lähdekoodi (esproswifi.ino)

Liite 2. Muokattu Python-koodi (talker.py)

GPS-vastaanottimen lähdekoodi (esproswifi.ino) Liite 1 (1/7)

```
#include "WiFi.h"
#include <SoftwareSerial.h>
#include <TinyGPS.h>
#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Int16.h>
#include <std_msgs/Float64.h>

//////////////////////
// WiFi Definitions //
//////////////////////
const char* ssid = ""; // Name of the WiFi network
const char* password = ""; // WiFi network password

TinyGPS gps;
SoftwareSerial ss(3, 1); // RX, TX
IPAddress server(192, 168, 1, 102); // IP of ROS server
IPAddress ip_address;
int status = WL_IDLE_STATUS;
char gpsData[100] = ""; // Read GPS module data to this char array
int separatorVal = 17; // Operates data separation on gpsData, value includes
amount of used print functions in loop plus one extra time on print date function


class WiFiHardware {

  public:
  WiFiHardware() {};

  void init() {
    // do your initialization here. this probably includes TCP server/client setup
    client.connect(server, 11411);
  }
```

GPS-vastaanottimen lähdekoodi (esproswifi.ino) Liite 1 (2/7)

```
  // read a byte from the serial port. -1 = failure
  int read() {
    // implement this method so that it reads a byte from the TCP connection and
returns it
    //  you may return -1 is there is an error; for example if the TCP connection is
not open
    return client.read();         //will return -1 when it will works
  }

  // write data to the connection to ROS
  void write(uint8_t* data, int length) {
    // implement this so that it takes the arguments and writes or prints them to
the TCP connection
    for(int i=0; i<length; i++)
      client.write(data[i]);
  }

  // returns milliseconds since start of program
  unsigned long time() {
     return millis(); // easy; did this one for you
  }

  protected:
    WiFiClient client;

};

void setupWiFi()
{
  WiFi.begin(ssid, password);
  Serial.print("\nConnecting to "); Serial.println(ssid);
  uint8_t i = 0;
  while (WiFi.status() != WL_CONNECTED && i++ < 20) delay(500);
```

GPS-vastaanottimen lähdekoodi (esproswifi.ino) Liite 1 (3/7)

```
if(i == 21){
    Serial.print("Could not connect to"); Serial.println(ssid);
    while(1) delay(500);
  }
  Serial.print("Ready! Address is ");
  Serial.print(WiFi.localIP());
}

std_msgs::String str_msg;
ros::NodeHandle_<WiFiHardware> nh;
ros::Publisher chatter("testdata", &str_msg);

void setup() {
  Serial.begin(115200);
  delay(1000);
  Serial.println("WiFi Setup");
  setupWiFi();
  delay(2000);
  Serial.println("GPS module setup");
  ss.begin(9600);
  delay(2000);
  Serial.println("GPS module setup done");
  delay(2000);
  Serial.println("init node");
  nh.initNode();
  Serial.println("init node done");
  delay(10000);
  nh.advertise(chatter);
  Serial.println("\nSetup done");
}

void loop() {
  float flat, flon;
```

GPS-vastaanottimen lähdekoodi (esproswifi.ino) Liite 1 (4/7)

```
  unsigned long age, date, time, chars = 0;
  unsigned short sentences = 0, failed = 0;
  static const double LONDON_LAT = 51.508131, LONDON_LON = -0.128002;

  print_int(gps.satellites(), TinyGPS::GPS_INVALID_SATELLITES, 5);
  print_int(gps.hdop(), TinyGPS::GPS_INVALID_HDOP, 5);
  gps.f_get_position(&flat, &flon, &age);
  print_float(flat, TinyGPS::GPS_INVALID_F_ANGLE, 10, 2); // 6
  print_float(flon, TinyGPS::GPS_INVALID_F_ANGLE, 11, 2); // 6
  print_int(age, TinyGPS::GPS_INVALID_AGE, 5);
  print_date(gps);

  print_float(gps.f_altitude(), TinyGPS::GPS_INVALID_F_ALTITUDE, 7, 2);
  print_float(gps.f_course(), TinyGPS::GPS_INVALID_F_ANGLE, 7, 2);
  print_float(gps.f_speed_kmph(), TinyGPS::GPS_INVALID_F_SPEED, 6, 2);
  print_str(gps.f_course() == TinyGPS::GPS_INVALID_F_ANGLE ? "****" : Ti-
nyGPS::cardinal(gps.f_course()), 6);
  print_int(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0xFFFFFFFF : (un-
signed long)TinyGPS::distance_between(flat, flon, LONDON_LAT, LON-
DON_LON) / 1000, 0xFFFFFFFF, 9);
  print_float(flat == TinyGPS::GPS_INVALID_F_ANGLE ? TinyGPS::GPS_IN-
VALID_F_ANGLE : TinyGPS::course_to(flat, flon, LONDON_LAT, LON-
DON_LON), TinyGPS::GPS_INVALID_F_ANGLE, 7, 2);
  print_str(flat == TinyGPS::GPS_INVALID_F_ANGLE ? "****" : TinyGPS::cardi-
nal(TinyGPS::course_to(flat, flon, LONDON_LAT, LONDON_LON)), 6);

  gps.stats(&chars, &sentences, &failed);
  print_int(chars, 0xFFFFFFFF, 6);
  print_int(sentences, 0xFFFFFFFF, 10);
  print_int(failed, 0xFFFFFFFF, 9);

  str_msg.data = gpsData;
  chatter.publish( &str_msg );
```

GPS-vastaanottimen lähdekoodi (esproswifi.ino) Liite 1 (5/7)

```
 strcpy(gpsData, ""); // Empty the char array for a new round in this loop

  nh.spinOnce();
  smartdelay(1000); // Very important for getting GPS data

}

static void smartdelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (ss.available())
      gps.encode(ss.read());
  } while (millis() - start < ms);
}

static void print_int(unsigned long val, unsigned long invalid, int len)
{
  separatorVal--;
  char sz[32];

  if (val == invalid)
    strcpy(sz, "*****");
  else
    sprintf(sz, "%ld", val);

  strcat(gpsData, sz);
  print_separator();
  smartdelay(0);
}

static void print_float(float val, float invalid, int len, int prec)
```

GPS-vastaanottimen lähdekoodi (esproswifi.ino) Liite 1 (6/7)

```
{
  separatorVal--;

  if (val != invalid)
  {
    char temp[32];
    dtostrf(val, 0, prec, temp);
    strcat(gpsData, temp);
    strcpy(temp, "");
  }
  else if (val == invalid)
  {
    strcat(gpsData, "****");
  }
  print_separator();
  smartdelay(0);
}

static void print_date(TinyGPS &gps)
{
  separatorVal--;

  int year;
  byte month, day, hour, minute, second, hundredths;
  unsigned long age;
  gps.crack_datetime(&year, &month, &day, &hour, &minute, &second, &hun-
dredths, &age);
  if (age == TinyGPS::GPS_INVALID_AGE)
    strcat(gpsData, "****");
  else
  {
    char sz[32];
    sprintf(sz, "%02d/%02d/%02d %02d:%02d:%02d",
```

GPS-vastaanottimen lähdekoodi (esproswifi.ino) Liite 1 (7/7)

```
        month, day, year, hour, minute, second);
    strcat(gpsData, sz);
  }
  print_separator();
  print_int(age, TinyGPS::GPS_INVALID_AGE, 5);
  smartdelay(0);
}

static void print_str(const char *str, int len)
{
  separatorVal--;

  strcat(gpsData, str);
  print_separator();
  smartdelay(0);
}

void print_separator() {
  if (separatorVal > 0) {
    strcat(gpsData, ",");
  }
  else if (separatorVal <= 0)
  {
    separatorVal = 17;
  }
}
```

Muokattu Python-koodi (talker.py) Liite 2

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('testdata', String, queue_size=10)
    rospy.init_node('carla_manual_control', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```