SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Johanna Eklund

GPS-paikannus käyttäen ESP32-mikro-ohjainta

Opinnäytetyö
Kevät 2025
Insinööri (AMK), Tietotekniikka

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Tutkinto-ohjelma: Insinööri (AMK), Tietotekniikka

Tekijä: Johanna Eklund

Työn nimi alaotsikoineen: GPS-paikannus käyttäen ESP32-mikro-ohjainta

Ohjaaja: Marko Hietamäki

Vuosi: 2025 Sivumäärä: 26

Tämän opinnäytetyön päätarkoituksena oli luoda GPS-paikannusohjelmisto käyttäen ESP32-mikro-ohjainta. ESP32-mikro-ohjain haluttiin ohjelmoida keräämään ja käsittelemään NMEA-dataa. Tämä data oli tarkoitus saada Neo-6M-GPS-moduulilta UART-portin kautta. Käsiteltyjä tietoja haluttiin käyttää sijaintitietojen piirtämiseen kartalle käyttäen kartta-alustaa. Kartta haluttiin toteuttaa käyttämällä React- ja Leaflet-kirjastoa.

Tämän työn koodaus tehtiin osissa, ja jokaisen muutoksen jälkeen testattiin muutokset toimiviksi. Toimiviksi muutokset voitiin todeta, kun virheilmoitukset oli korjattu. Ensin luotiin ESP32-mikro-ohjain ohjelmisto, joka käsittelee paikannusdataa ja luo HTTP-palvelimen, josta data on saatavilla. Toinen osa työstä oli luoda Reactia käyttäen sovellus, joka hakee sijaintitiedot HTTP-palvelimelta. Tämän jälkeen sovellus piirtää saadun datan kartalle.

Ongelmia tuotti laitteiston toimimattomuus, kun Neo-6M-moduulilta ei tullut haluttua dataa. Tätä ongelmaa ei pystytty korjaamaan. Muut työssä kohdatut ongelmat pystyttiin ratkaisemaan virheenkorjauksella. Näitä ongelmia olivat ESP32-mikro-ohjaimen WiFi-yhteyden aikakatkaisu sekä muistin korruptoituminen ja CORS-pyynnöt. Reactin puolelta täytyi korjata virheellisen datan käsittely sekä ajastukset.

Opinnäytetyön lopputuloksena saatiin kartta, joka esittää ESP32:n lähettämät sijaintitiedot reaaliaikaisesti. ESP32 ohjelmoitiin simuloimaan paikannusdataa, joka lähetettiin React-sovelluksen saataville. React-sovellus käsitteli ja piirsi koordinaatit karttaan käyttäen Leaflet-kirjastoa.

[1] Asiasanat: GPS, paikannus, koordinaatit

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Degree programme: Bachelor of Engineering, Information Technology

Author/s: Johanna Eklund

Title of thesis: GPS positioning using an ESP32 microcontroller

Supervisor(s): Marko Hietamäki

Year: 2025 Number of pages: 26

The main purpose of the thesis was to create a GPS positioning system using an ESP32 microcontroller. The idea was that the ESP32 would be programmed to collect and process NMEA data which was supposed to be provided by a Neo-6M GPS module through an UART port. The processed data was then intended to be used for drawing the position onto a map using a mapping software. It was wanted that the map would be executed using React and the Leaflet library.

The coding for this project was done in parts and after each part the changes were tested. The changes could be seen as functioning once all error messages had been fixed. The first part was the software of ESP32 which was meant to handle the data and create an HTTP-server in which the data would be available. The second part was to create a React app that would fetch the location data from the HTTP-server and draw the received data onto a map after that.

During the project, there were some problems caused by the malfunctioning of the equipment as the desired data was not received from the Neo-6M module. This issue could not be fixed. Other problems encountered were resolved through debugging. These issues included Wi-Fi connection timeouts, memory corruption, and CORS requests on the ESP32 side. On the React side, the handling of incorrect data and timing issues had to be fixed too.

As the result of the thesis there was a map that displays the location data sent by the ESP32 in real-time. The ESP32 was programmed to simulate location data, which was made available to the React application. The React application processed and plotted the coordinates on the map using the Leaflet library.

[1] Keywords: GPS, positioning, coordinates

SISÄLTÖ

Kuvaluettelo

Käytetyt termit ja lyhenteet

GPS	Maailmanlaajuinen paikallistamisjärjestelmä (Global Positioning System).
NMEA	NMEA 0183 -standardin mukainen paikannusdata (National Marine Electronics Association)**.**
UART	Sarjaliikennepiiri (Universal Asynchronous Receiver Transmitter).

1 JOHDANTO

1.1 Työn tausta

GPS-paikannus on tällä hetkellä kuluttajien näkökulmasta hyvin yleistä. Monesta laitteesta löytyy sellainen. Älykello voi tallentaa kuljetun reitin, kuulokkeet voivat ilmoittaa sijaintinsa, ja lähes jokainen älypuhelin voi näyttää käyttäjälleen sijaintinsa kartassa. Suurimmat puhelimen valmistajat ovat alkaneet tuottaa tavaroihin kiinnitettäviä tageja, jotka seuraavat tavaran sijaintia vaikkei tavaralla itsellään olisikaan älyä.

Maailmanlaajuiset GPS-paikantimien markkinat ovat kasvaneet vuodesta 2020 vuoteen 2025 vakaasti (Laricchia, 2024). GPS-paikannusta käyttävien laitteiden markkinat olivat vuonna 2020 1,3 miljardia USA:n dollaria. Vuonna 2025 markkinoiden on arvioitu kasvavan 2,38 miljardiin dollariin. Tämä tarkoittaa siis yli miljardin dollarin kasvua vain viidessä vuodessa.

1.2 Työn tavoite

Tämän työn tavoitteena on ohjelmoida ESP32-mikro-ohjain keräämään ja käsittelemään NMEA 0183 -standardin mukaista GPS-paikannusdataa. Data tullaan keräämään ESP32-mikro-ohjaimeen liitetyn Neo-6m-GPS-moduulin avulla. Paikkatiedoista poimitut koordinaatit tullaan lähettämään karttasovellukselle. Karttasovelluksena toimiva ohjelma tulee piirtämään graafisesti ja reaaliaikaisesti Reactilla toteutettuun sovellukseen, missä kartta on käyttäjän nähtävillä.

1.3 Työn toteutus

Tässä työssä tullaan tekemään kaksi eri ohjelmaa, jotka kommunikoivat keskenään. Ensimmäinen ohjelma on ESP32-mikro-ohjaimelle ladattava ohjelma. Sen tehtävä on kerätä ja käsitellä sijaintitietoa Neo-6M-moduulilta, ja lähettää tietoa eteenpäin karttasovelluksen saataville. Toinen ohjelma on itse karttasovellus, joka hakee sijaintitietoja, ja piirtää ne reaaliaikaisesti kartalle. Ennen ohjelmien luontia, testataan kuitenkin yhteys ESP32-mikro-ohjaimen ja Neo-6M-moduulin välillä.

Kun edellä mainitut kaksi ohjelmaa on tehty ja todettu toimivaksi, testataan niiden toimintaa yhdessä. Näissä testeissä on tarkoituksena korjata kaikki esiintyvät virheet. Lopuksi tullaan vielä testaamaan kokonaisuus, eli kaksi ohjelmaa, jotka kommunikoivat keskenään sekä data, joka liikkuu niiden välillä. Työ voidaan todeta toimivaksi, kun ESP32:n lähettämä data tulee näkyviin kartassa.

2 GPS-PAIKANNUS

2.1 GPS-paikannuksen historia

GPS (Global Positioning System) on maailmanlaajuinen navigointijärjestelmä, joka hyödyntää tarkkaa ajanmääritystä ja etäisyyksien mittaamista sijainnin määrittämiseen (Miettinen, 2006, s. 23). GPS-järjestelmä kehitettiin alunperin sotilastarkoituksiin. Sen luonnin tavoitteena oli kehittää tekniikka, joka sietää vihollisen häirintää, ja toimii ilmakehästä ja avaruuden luonnonilmiöstä huolimatta.

Maailmanlaajuinen navigointijärjestelmä ideoitiin jo 1940-luvulla (Miettinen, 2006, s. 21). Kuitenkin vasta 1950-luvulla käyty Vietnamin sota johti varsinaisten hankkeiden käynnistykseen. Vuonna 1964 amerikkalaiset ottivat käyttöön satelliittipaikannusjärjestelmän nimeltä Transit (Navy Navigation Satelite System, NNSS). Se päätyi paria vuotta julkaisunsa jälkeen rajoitetusti myös siviilien käyttöön. Transit poistettiin käytöstä 1996, jolloin sen korvattiin GPS-paikannuksella.

Vuonna 1999 USA julkisti aloitteen GPS:n modernisoimiseksi (Kaplan & Hegarty, 2006, s. 5). Aloite lisäsi uusiin GPS-satelliitteihin kaksi siivilisignaalia. Lisätyt signaalit mahdollistivat kaksoistaajuuslaskennan, joka lisäsi merkittävästi tarkkuutta siviilikäyttäjille. Lisätyt signaalit myös paransivat häiriönsietokykyä, sillä jos toinen signaali kokee häirintää, voi vastaanottaja vaihtaa toiseen signaaliin. Tällä muutoksella oli tarkoitus edistää GPS:n käyttöä maailmanlaajuisesti siviilikäytössä, kaupallisessa ja tieteellisessä käytössä.

2.2 GPS sekä muut satelliittijärjestelmät

GPS-paikannukseen kuuluu muutamia peruskäsitteitä (Miettinen, 2006, s. 19–20). Navigointi tarkoittaa prosessia, jossa määritellään kohteen sijainti, suunta ja etäisyys sekä edetään valittua reittiä pitkin kohti kyseistä kohdetta. Sijainti tarkoittaa tarkkaa tietoa paikasta maapallon pinnalla, tämä ilmaistaan koordinaattilukujen avulla. Koordinaattijärjestelmä on vertausjärjestelmä, jossa pisteen sijainti määritetään numeerisesti joko metrisinä lukuina tai kulmina suhteessa ennalta asetettuihin koordinaattiakselien tai nollatasojen arvoihin. Paikan sijainti määritellään koordinaattien eli koordinaattilukuarvojen, niiden yksiköiden ja

käytetyn koordinaattijärjestelmän perusteella. Reittipiste on sijainti, joka on määritelty koordinaateilla ja kuuluu reitin varrella oleviin kulkupisteisiin. Reitti tarkoittaa kulkutietä, joka yhdistää lähtöpaikan ja päätepisteen.

GPS-järjestelmät voidaan jakaa kolmeen eri osaan (Miettinen, 2006, s. 32). Ensimmäinen osa on avaruusosa eli aktiiviset sekä varalla olevat satelliitit. Toinen osa on valvontaosa. Valvontaosaan kuuluu keskusasema, jonka tehtävä on ohjata järjestelmien toimintaa sekä suorittaa tarvittavat huoltotoimenpiteet. Lisäksi valvontaosaan kuuluu maa-asemat, joista voidaan seurata tapahtumia taivaalta sekä kerätä tietoa. Kolmantena valvontaosaan kuuluu maa-antennit, jotka välittävät tiedot satelliiteille. Kolmas ja viimeinen osa on GPS-paikantimet, eli tavallisten käyttäjien sovelluksen ja laitteet.

GPS on kaksikäyttöinen, sillä se tarjoaa palveluita niin sekä siviili- että sotilaskäyttöön (Kaplan & Hegarty, 2006, s. 3–4). Näitä palveluita kutsutaan nimillä Standard Positioning Service (SPS) eli standardi sijaintipalvelu, ja Precise Positioning Service (PPS) eli tarkka sijaintipalvelu. Näistä SPS on tarkoitettu siviilikäytttöön ja PPS on tarkoitettu USA:n valtuutettuun sotilaskäyttöön sekä valikoitujen valtion virastojen käyttäjien käyttöön. PPS-palvelun käyttöä hallinnoidaan kryptografialla. GPS-järjestelmä käyttää yksisuuntaista time of arrival (TOA) -konseptia sijainninmäärittelyyn.

Euroopan unioni halusi vuonna 1998 satelliittijärjestelmän, joka on erillinen GPS-järjestelmästä (Kaplan & Hegarty, 2006, s. 5–6). Satelliittijärjestelmä haluttiin maailmanlaajuiseen siviilikäyttöön. Näin syntyi Galileo-satelliittijärjestelmä. Galileolla on monia eri palvelutasoja. Suoraan käyttäjille se tarjoaa avoimen ja ilmaisen palvelun. Kaupallisena palveluna se tarjoaa tarkempaa paikannusta. Safety-of-life (SOL) -palveluna se tarjoaa turvaa kriittisiin tarpeisiin. Säänneltynä julkisena palveluna se tarjoaa korkeamman turvatason valtion valtuuttamassa käytössä. Se tarjoaa myös tuen pelastus- ja etsintäoperaatioihin.

Venäläinen GLONASS-satelliittijärjestelmä on vastine USA:n GPS:lle (Kaplan & Hegarty, 2006, s. 8). GLONASS koostuu satelliiteista keskikorkealla maapallon kiertoradalla (MEO), maa-asemasta sekä käyttäjien laitteista. Kuten GPS, myös GLONASS on kaksikäyttöinen, se tarjoaa eri palvelut siviileille ja sotilaskäyttöön. GLONASS on siviilikäytössä ilmainen palvelu.

Kiinalainen BeiDou-satelliittipaikannusjärjestelmä on suunniteltu Kiinan armeijan ja siviilien käyttöön (Kaplan & Hegarty, 2006, s. 9). Se tarjoaa palveluita paikannukseen, kaluston hallintaan sekä tarkkuusaikalevitykseen. Toisin kuin aiemmat satelliittijärjestelmät, jotka ovat yksisuuntaisia TOA-mittauksia, BeiDou käyttää kaksisuuntaista determination satellite service (RDSS) -palvelua.

2.3 NMEA-data

NMEA eli National Marine Electronics Association oli olemassa jo ennen GPS:n keksimistä (Gakstatter, 2015). Nykyisin NMEA on standardi datarakenne, jota tukevat kaikki GPS-laitteiden tuottajat. NMEA:n mukainen data helpottaa ohjelmoijien työtä, sillä standardin mukainen data on helppo jäsennellä. NMEA koostuu lausekkeista, kuten esimerkiksi GPGGA. Kirjaimet GP GPGGA-lausekkeessa kertovat käyttäjälle, että kyseessä on GPS-sijainti. GPGGA-lauseke antaa aikaleiman, latituden eli leveysasteen, longituden eli pituusasteen, sekä kirjaimet ilmaisemaan ilmansuuntaa. Tämän lisäksi lauseke kertoo muun muassa, kuinka monta satelliittia käytettiin koordinointiin.

2.4 ESP32

ESP32 on edullinen siru, jonka on kehittänyt Espressif Systems (Kurniawan, 2019, s.6–7). Siihen on integroitu yhdelle sirulle Wi-Fi- ja Bluethooth-yhteydet. Siinä on tuki vanhoille Bluetooth for legacy -yhteyksille sekä Bluetooth Low Energy (BLE) eli matalan virran -yhteyksille. ESP32:lle on saatavilla kahta eri mallia: siru ja moduuli. Ne eroavat toisistaan muun muassa koon ja pinnien määrän perusteella.

ESP32 soveltuu esineiden internet (IoT) -projekteihin (Oner, 2021, s. 3). Sitä voidaan käyttää erilaisiin projekteihin esimerkiksi teollisuuden automaatiossa, älykodeissa, sairaanhoidossa sekä puheen tai kuvan tunnistuksessa (Espressif Systems, 2025, s. 5). ESP32-sirussa on UART-sarjaportti (Universal Asynchronous Receiver Transmitter), joka helpottaa ESP32-sirun ja ulkoisten UART-laitteiden välistä asynkronisen sarjadatan lähetystä ja vastaanottoa (mts. 37).

2.5 Neo-6M

Neo-6M on suoriutuskyvykäs ja kustannustehokas Neo-6-sarjan GPS-moduuli (U-blox, 2011, s. 5). Neo-6M on itsenäinen GPS-vastaanotin. Siinä on teknologiaa, joka vaimentaa häirintälähdettä. Moduuli on integroitavissa ja yhdistettävissä helposti muihin laitteisiin. Neo-6M on mahdollista yhdistää UART-, USB-, SPI-, tai DDC-väylän kautta. Neo-6M tukee vastaanotossa ja lähetyksessä NMEA 0183 -standardia (mts. 9).

Neo-M8 on GNSS-moduuli, joka käyttää GPS:n lisäksi Galileoa, GLONASSia sekä BeiDouta (U-blox, 2022, s. 5). Neo-M8-sarja on luokaltaan ammattitason moduuli. Neo-M8 on taaksepäin yhteensopiva Neo-7-, 6- ja 5-sarjojen kanssa. Neo-M8:ssa on samat yhdistysväylät kuin 6-sarjassakin. Neo-M8 tukee vastaanotossa ja lähetyksessä NMEA 0183 -standardia (mts. 12).

2.6 Kartta-alustat

Yksi suosituimmista kartta-alustoista on Leaflet (Leaflet, i.a.). Leaflet on avoimen lähdekoodin JavaScript-kirjasto. Se on ohjelmistokehittäjälle helppokäyttöinen ja suorituskykyinen kirjasto. Se on alustan kannalta joustava, sillä se toimii sekä tietokoneella että puhelimellakin. Leafletin toimintoja on mahdollista laajentaa laajennuksilla. Leaflet tarjoa kirjastonsa käyttöön valmiita dokumentteja ja tutoriaaleja.

Mapbox on paikannusteknologia, joka hyödyntää AI:ta eli teköälyä (Mapbox, i.a.). Se on tarkoitettu logistiikkapalveluille, autonvalmistajille sekä puhelinsovelluskehittäjille. Mapboxia käytetään ohjelmistokehityspaketin (SDK) tai API-rajapinnan kautta. Siinä on saatavilla valmiita karttatyylejä ja reaaliaikaista tietojen päivitystä, mitkä tekevät Mapboxista helposti räätälöitävän alustan.

OpenLayers on monipuolinen karttakirjasto, joka tarjoaa tehokkuutta ja paljon toimintoja (OpenLayers, i.a.). OpenLayers on yhteensopiva monen kolmannen osapuolen kirjastojen kanssa, mikä tekee siitä hyvin räätälöitävän. OpenLayers tukee myös vektoridatan renderöintiä ja sillä on valmius mobiilitukeen. OpenLayersillä on itsellään tarjolla dokumentteja ja opetustutoriaaleja.

2.7 Ohjelmointiohjelmistot ja -kirjastot

React on Metan omistama avoimen lähdekoodin kirjasto (React, i.a.). Sen avulla on mahdollista tehdä web-käyttöliittymiä ja natiiveja käyttöliittymiä. Käyttöliittymät rakennetaan komponenttien kuten LikeButton avulla. Reactin komponentit ovat ohjelmoinnissa käytettäviä funktioita. Ohjelmaan lisätyt komponentit rakennetaan yhtenäiseksi sivuksi tai sovellukseksi. Reactin funktiot ovat JavaScript-ohjelmointikielellä toteutettuja funktioita. Ohjelmoijalle on Reactilla tarjolla dokumentteja ohjelmoinnissa alkuun pääsyyn.

Visual Studio Code on ohjelmointikäyttöliittymä, joka on muokattavissa käyttäjän omien tarpeiden mukaiseksi (Visual Studio, i.a.). Visual Studio Code on käyttäjän muokattavissa laajennusten ja väriteemojen avulla. Visual Studio Codessa on mahdollista ajaa ja debugata eli etsiä virheitä. Siinä on integroitu terminaali ohjelmien ajoon ja versionhallintaan. Versionhallinta onnistuu myös Visual Studio Codeen rakennetusta versionhallintatyökalusta. Kielituki löytyy kaikille koodikielille. Visual Studio Codeen on mahdollista asentaa myös tekoälytyökalu GitHub Copilot koodauksen avuksi.

GitHub on ohjelmistojen yhteistyöalusta (GitHub, i.a.). Sen avulla on mahdollista rakentaa ohjelmistoja ja toteuttaa jatkuvaa ohjelmistotoimitusta. GitHubilla on tarjolla tekoälytyökaluja ohjelmoinnin nopeuttamiseen sekä turvauhkien etsintään ohjelmisto koodista. Myös itse koodaus on mahdollista GitHubissa siihen integroidun Visual Studio Code Codespacen avulla. GitHub mahdollistaa tehtävänseurannan, koodin katselmoinnin sekä kysymysten esittämisen alustallaan.

3 PROJEKTIN SUUNNITTELU

3.1 Käytettävät laitteet ja sovellukset

Tässä työssä tullaan käyttämään Espressif Systemsin ESP32-S3-WROOM-1-mikro-ohjainta. Selkeyden vuoksi tullaan käyttämään nimeä ESP32, mutta mainittakoon, että kyseessä on sen S3-malli. Tähän päätökseen päädyttiin, sillä työn tekijällä on eniten kokemusta kyseisen ESP32-mallin ohjelmoinnista. ESP32:n ohjelmointiin tullaan käyttämään sille tarkoitettua Visual Studio Code -ohjelmaan lisättyä ESP-IDF -laajennusta. ESP-IDF:n ohjelmisto on ladattavissa Espressif Systemsin GitHubista.

Paikannuskoordinaattien keräykseen tullaan käyttämään Neo-6M-moduulia, joka välittää datan ESP32:lle UART-sarjaportin kautta NMEA-standardin mukaisena lausekkeena. Neo-6M on edullisin vaihtoehto moduuliperheestä, ja se on tähän työhön riittävä.

Kartta-alustana tullaan käyttämään NMEA 0183 -standardia tukevaa Leafletiä. Leaflet-ohjelma on avointa lähdekoodia, joten se on helposti saatavilla. Siitä on myös tarjolla paljon tutoriaaleja, joten se on käyttäjäystävällinen ohjelmoijalle.

Versionhallintaan tullaan käyttämään GitHubia. Tämän työnosan kuvaus tullaan kuitenkin jättämään pois, sillä se ei ole työn kannalta oleellista tietoa. Lopputulos tulee kuitenkin olemaan saatavilla GitHubissa.

3.2 Lähtökohta

Ennen kuin voidaan aloittaa itse työ, on ensin etsittävä internetistä esimerkkejä GPS-paikannusprojekteista, joissa käytetään ESP32-mikro-ohjainta ja Neo-6M-moduulia. Tarkoituksena on selvittää niiden avulla, miten voidaan tehdä kytkennät ESP32:n ja Neo-6M-moduulin välillä. Esimerkeistä halutaan myös nähdä, millä logiikalla NMEA-datan lukeminen on toteutettu. Onneksi ESP32:lle löytyy internetistä runsaasti tutoriaaleja, joten esimerkkien löytymisen ei pitäisi olla ongelma työn aloitukselle. ESP32:n ja Neo-6M:n väliset kytkennät täytyy kuitenkin tarkistaa ESP32-S3:n Espressiffin omista dokumenteista, sillä

internetistä löytyvät tutoriaalit on usein tarkoitettu nimenomaan ESP32-perusmallille. ESP32-S3-mallissa on vähän erilainen pin-asettelu.

Aiemmasta kokemuksesta on tiedossa, että on yleistä, että tutoriaalit on tehty Arduino IDE -ohjelmointikäyttöliittymälle. Koska tässä työssä ei tulla kyseistä ympäristöä käyttämään, joudutaan todennäköisesti kääntämään mahdolliset koodin osat käyttöliittymän muutoksen takia koodikielestä toiseen.

3.3 Toivottu lopputulos

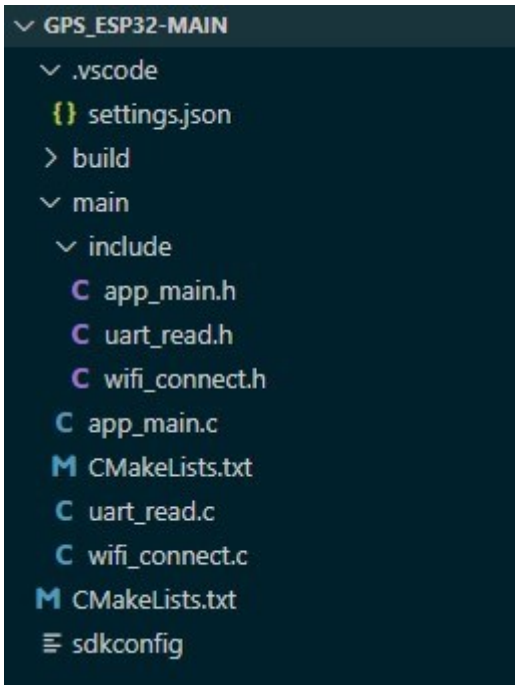
Jotta saadaan piirrettyä sijaintitiedot karttasovellukseen, täytyy eri koodin osien sekä laitteiston toimia yhdessä. Ensinnäkin ESP32:n tulee pystyä lukemaan ja jäsentämään Neo-6M:ltä saadusta NMEA-datasta haluttu GPGGA-lauseke. Neo-6M:n täytyy myös pystyä lähettämään data ESP32:lle. Data täytyy pystyä sen jälkeen siirtämään ohjelman osalle, joka välittää datan karttasovellukseen. Tämä ohjelman osa on mahdollista toteuttaa kahdella eri tavalla. Ensimmäinen vaihtoehto on säilyttää koko toiminnallisuus ESP32:n puolella, jolloin karttasivu on nähtävissä vain erillisellä laitteella, joka on samassa WLAN-verkossa kuin ESP32. Toinen mahdollisuus on, että ESP32 lähettää käsittelemänsä datan ulkoiselle ohjelmalle, joka suorittaa kartan piirtämisen.

Toivottu lopputulos on siis ohjelma, joka kerää sijaintitiedot, ja joko lähettää ja piirtää ne karttaohjelmalle. Valmiin kartan tulisi olla käyttäjän nähtävissä. Parempi lopputulos olisi, jos sovellus pystyy piirtää sijainnin karttaan reaaliaikaisesti Reactia käyttäen. Jos tämä ei onnistu, toinen vaihtoehto on, että kerätyt sijaintitiedot piirretään karttaan kaikki kerralla.

4 TYÖN TOTEUTUS

4.1 Aloitus

Työ aloitettiin kääntämällä internetistä löydettyjen tutoriaalien koodin osia Arduino-ohjelmointikielestä ESP-IDF:n käyttämään C-kieleen. Kääntö kielestä toiseen tapahtui yksinkertaisesti selvittämällä koodista, mitä siinä tapahtuu ja tekemällä saman asian käännetyssä koodissa. Koodi ei kuitenkaan ollut vielä ohjelmointikielikäännöksen jälkeen suoraan käännettävissä ESP32:lle, vaan se vaati lisäyksiä ja rakenteellisia muutoksia. C-kielen kääntöä varten tarvittiin rakentaja. Sitä varten asennettiin MinGW-ohjelma, ja lisättiin se Windowsin ympäristömuuttujiin. Tämän lisäksi projektin rakennusta varten tarvittiin kaksi CmakeLists.txt-tiedostoa. Näiden tarkoitus oli osoittaa projektin rakennusohjelmalle, mistä tiedostopoluista kaikki tarvittavat tiedostot löytyvät. Lopuksi korjattiin vielä projektin kansiorakenne järjestelmällisemmäksi.



Kuva 1. ESP32-projektin korjattu kansiorakenne.

Seuraavaksi lisättiin projektiin uusi tiedosto, jonka tarkoitus on muodostaa WiFi-yhteys, jotta voidaan lähettää dataa karttaohjelmalle. Tätä varten ohjelmaan tuotiin esp_wifi.h-kirjasto. Sen avulla ESP32:n WiFi-yhteys avattiin STA-tilassa eli wifi-asiakastilassa. Ohjelmalle täytyi määritellä halutun WiFi-yhteyden nimi ja salasana. Saman WiFi-yhteyden on

oltava käytettävissä myös myöhemmin tehtävällä karttasovelluksella, jotta toteutus pysyy yksinkertaisempana. Eri WiFi-yhteyksissä ajettavat sovellukset vaatisivat karttasovelluksen, joka on julkaistu jollekin ulkoiselle palvelinalustalle, joka ylläpitää sovellusta.

4.2 Datan keruu Neo-6M:ltä

Ensimmäinen kehitysvaihe oli testata, tuleeko Neo-6M:ltä haluttua dataa ESP32:lle UART-portin kautta. ESP32 yhdistettiin pinnien kautta Neo-6M-moduuliin. Testaukseen käytettiin aiemmin käännettyä yksinkertaista ohjelmaa, joka lukee dataa UART:n kautta. Koodin testaus ei kuitenkaan tuonut toivottuja tuloksia. Dataa ei näyttänyt tulevan sarjaportin kautta. ESP32 kyllä tulosti yrityksensä lukea dataa, mutta luettu data oli aina tyhjää.

Neo-6M:n datan lähetys voi viivästyä hetken käynnistyksessä, mutta edes 10 minuutin odotuksen jälkeen ei dataa tullut. Seuraavaksi tehtiin pieniä korjauksia ohjelmiston tulostuksiin, datan käsittelyyn sekä ajastuksiin, mutta mikään muutoksista ei korjannut ongelmaa. Kytkennät testattiin muutamaan kertaan, mutta vastaanotettu data pysyi silti tyhjänä. Neo-6M oli kuitenkin kytkettynä päälle. Tämän pystyi päätellä siitä, että moduuli lämpeni, kun ESP32 käynnistettiin. Tässä vaiheessa työtä ei kuitenkaan haluttu jäädä jumiin laitteistojen takia, vaan siirryttiin ohjelmiston kehitykseen.

4.3 Sijaintitietojen välitys karttaohjelmalle

Seuraava vaihe oli selvittää, miten sijaintidata tulisi välittää kartta käyttöliittymänä käytettävälle ohjelmalle. Ensimmäisenä kehitettiin ohjelmistoa, jossa ESP32 toimii datan kerääjänä ja backendinä eli palvelinpuolena. Karttaohjelmisto taas toimisi frontendinä eli käyttöliittymänä. Jotta frontend ja backend toimisivat yhdessä, täytyy ESP32:n pystyä lähettämään NMEA-data palvelimelle saataville. Frontendin taas on pystyttävä hakemaan data fetch-toiminnoilla, jotta se voidaan piirtää Leaflet-karttaan. Backend ja frontend tullaan ajamaan samassa WiFi-yhteydessä, jolloin websivujen pitäisi helposti olla näkyvissä toisilleen.

ESP32:n palvelinpuolena toimii HTTP-websivu. Tätä varten on ESP32:lle saatavilla valmis kirjasto nimeltä esp_http_server, jonka funktioita voitiin tässä työssä hyödyntää. ESP32:n aiemmin toteutettua koodia muunnettiin niin, että aiemmin luotu WiFi-yhdistys lukee ja

tulostaa myös ESP32:n IP-osoitteen. Tätä IP-osoitetta tullaan käyttämään HTTP-palvelimen verkko-osoitteena. Tämän lisäksi täytyi lisätä osoitteen oletusportti. Lisäksi tehtiin muutoksia UART-portista lukutoimintoon niin, että saadut latitude- ja longitude-tiedot välitettiin vielä eteenpäin HTTP-palvelimelle. HTTP-palvelin asetettiin päivittämään saadut latitude- ja longitude-tiedot karttaohjelman saataville.

4.4 Karttaohjelman toteutus

Itse karttan toteuttava sovellus tehtiin Javascript-kielellä käyttäen Reactia ja Leaflet-kirjastoa. Ennen kuin aloitettiin tekemään itse ohjelmaa, täytyi ensin tarkistaa, että JavaScript-kieli on käännettävissä Visual Studio Codella. Jotta se olisi mahdollista, täytyi ensin asentaa Node.js-ohjelmisto. Tämän jälkeen Node.js:n kotikirjasto lisättiin Windowsin ympäristömuuttujiin. Karttaohjelman pohja luotiin Reactin avulla. Sitä varten täytyi ensin asentaa npm-kirjasto komennolla ’npm install’. Tämä täytyi tehdä Visual Studio Coden komentoterminaalista, sillä oletuksena avautuva Powershell-terminaali ei suoritusoikeuksien puutteen takia pystynyt komentoa ajamaan.

React-projekti luotiin ’npx create-react-app’-komennolla Visual Studio Coden komentoterminaalista. Tämä komento loi valmiiksi kaikki tarvittavat tukitiedostot sekä päätiedoston, App.js, johon tullaan tekemään kaikki tarvittavat muutokset. Lisäksi täytyi asentaa kaikki kirjastot, joita tullaan käyttämään. Näitä olivat React, Leaflet sekä Web-vitals.

Itse kartta oli melko yksinkertainen toteuttaa ottaen mallia Leafletin verkkosivuilta löytyvistä valmiista tutoriaaleista. Leaflet asetettiin käyttämään OpenStreetMap-karttaa. Haetut koordinaatit tullaan piirtämään kartalle käyttäen Leafletin Markeria ja Polylinea. Ojelma asetettiin hakemaan fetch-toiminnolla ESP32:n palvelimen verkko-osoitteesta sinne tallennetut koordinaatit.

4.5 Verkkosivujen testaus

Kun sekä ESP32:n että React-sovelluksen verkkosivut oli testattu toimivaksi ilman virheilmoituksia, oli aika testata niitä yhdessä. Testaus tapahtui yksinkertaisesti niin, että

molemmat ohjelmat ajettiin samaan aikaan, ja seuraamalla, saavatko ne yhteyden toisiinsa. Tästä tuloksena oli virheilmoitus Reactin puolelta. Virhe liittyi CORS-kyselyihin eli ongelma koski Reactin tekemää fetch-pyyntöä. Vaikka kyseessä olikin virheilmoitus, oli se tämän työn kannalta kuitenkin hyvä asia. Se nimittäin osoitti, että React-sovellus oli tosiaankin löytänyt ja yrittänyt ottaa yhteyttä ESP32-palvelimeen.

CORS-virheilmoitus oli helposti korjattavissa lisäämällä ESP32:n http-palvelimen koodeihin CORS-käsittelijät. Tämän avulla ESP32 pystyi käsittelemään Reactin lähettämän fetch-pyynnön. Seuraava testaus osoitti, että virheilmoitus oli poistunut.

Seuraavaksi Reactin puolelta tuli virheilmoitus, että palvelimelta haettu data ei ollut sitä, mitä sen oletettiin olevan. Tämä johtui siitä, että ESP32: puolella koordinaateista tuli vain yhdet alkuarvot, kun React oletti niitä olevan monta. Tämä korjautui lisäämällä Reactin puolelle virheellisen datan suodatus. Tehtiin uusi testaus ja ongelma oli korjaantunut.

4.6 Testaus simuloidulla datalla

Tässä kohtaa työtä testattiin uudelleen Neo-6M-moduulia, mutta dataa ei edelleenkään tullut sarjaliikenneportista. Kytkennät testattiin jälleen kerran ja lisäksi testattiin muun muassa vaihtaa baund rate eli sarjaportin liikenteen nopeus, mutta parannuksia ei saatu. UART-portista pystyttiin satunnaisesti lukemaan heksadesimaalimuodossa ’FF 00 FF 00’. Tämä viittaa kuitenkin virheeseen, eikä ole oikeanlaista dataa. Neo-6M-moduuliin voitiin siis olettaa olevan toiminnassa, mutta jostain syystä se ei joko kerännyt dataa tai lähettänyt sitä eteenpäin.

Tästä syystä päädyttiin käyttämään stimuloitua dataa koko ohjelmiston testaukseksi. Simuloitujen koordinaattien sijainniksi valittiin Helsinki. Dataan tehtiin vaihtuvuutta valmiilla rand-funktiolla, joka satunnaisti koordinaattien viimeisiä desimaaleja. Nämä koordinaatit asetettiin tallennettavaksi listaan, josta ne muunnetaan JSON-dataksi, ja välitetään palvelimelle.

Ohjelmiston toimivuus testattiin taas ajamalla molemmat verkkosivut samaan aikaan. Tässä testissä huomattiin, että React-sovelluksesta puuttui tärkeä osa. Funktioon

useEffect täytyi lisätä vielä automaattinen päivitys, eli interval, joka asetettiin hakemaan uudet koordinaatit kahden sekunnin välein. Tämän muutoksen jälkeen kartta alkoi toimia. ESP32 lähetti datan palvelimelleen ja React kävi hakemassa ne sieltä, jonka jälkeen Leaflet piirsi ne kartalle.

4.7 Aikatestaus

Kun karttaohjelmisto näytti toimivan niin kuin oli tarkoituskin, tehtiin vielä viimeinen testaus. Testissä oli tarkoituksena antaa ohjelmien olla päällä noin puoli tuntia ja seurata, tuleeko vielä virheilmoituksia. Testi oli siinä mielessä onnistunut, että korjattavia virheitä ilmaantui. Ensimmäinen virhe koski ESP32:n WiFi-yhteyttä. Yhteydessä tapahtui pienen ajan päästä ESP32:n käynnistyksestä aikakatkaisu. Sen seurauksena yhteys palvelimeen katkesi. Tämä korjattiin lisäämällä EPS32:lle task eli tehtävä, jonka tarkoitus oli tarkistaa, ettei WiFi-yhteys ollut sammunut.

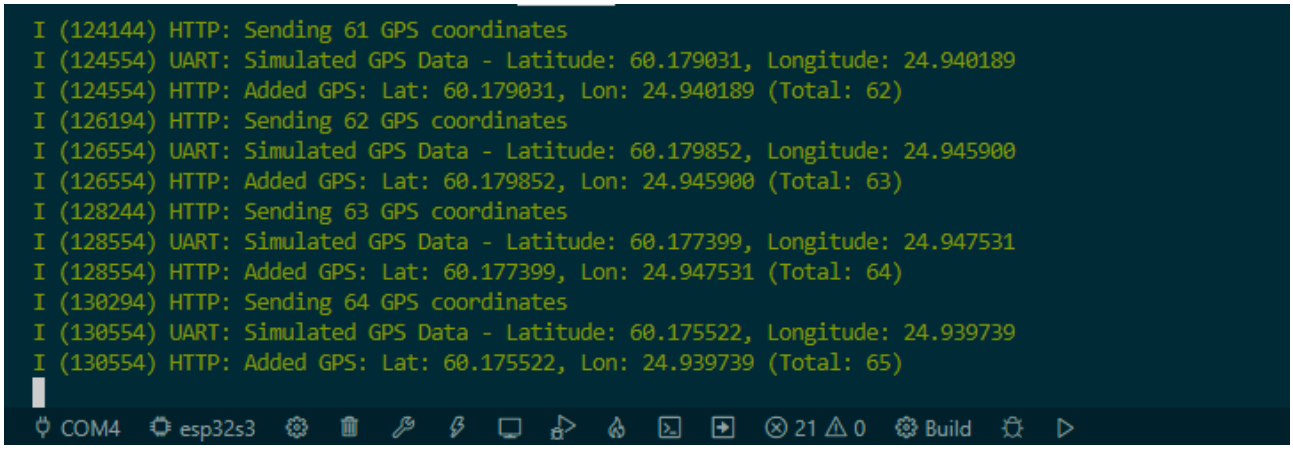
Testaus toistettiin ja WiFi-yhteys pysyi yllä. Hetken päästä tuli kuitenkin ilmi seuraava ongelma: ESP32 aktivoi paniikkikäsittelijän aina 43:n lähetetyn koordinaatin jälkeen. EPS32:n virhetulostuksia tutkittua kävi ilmi, että ESP32:n Heap-muisti korruptoitui. Tämä tarkoitti siis sitä, että jossain kohtaa koodia käytettiin liikaa muistia, jolloin lisämuistia yritettiin hakea paikasta, josta sitä ei ollut lupaa ottaa. Tällöin ESP32 aktivoi paniikkikäsittelijän estääkseen väärän muistin käytön.

Tämän seurauksena tehtiin muutoksia koordinaattidatan käsittelyyn HTTP-palvelimen puolella. Käytettävästä koordinaattilistasta asetettiin poistettavaksi listan viimeiset vanhimmat koordinaatit. Myös HTTP-palvelimen käyttämän JSON-taulukon käyttämä puskurin koko asetettiin dynaamiseksi eli olemaan riippuvainen koordinaattilistan koosta. Viimeisenä pienennettiin vielä WiFi-yhteyttä ylläpitävän tehtävän pinomuistia noin tuhannella. Näillä muutoksilla muistin ei pitäisi ylittyä.

Viimeinen testi olikin onnistunut. Yhteys pysyi yllä ja koordinaatit piirtyivät karttaan. Uusia virheilmoituksia ei ilmaantunut enää kummankaan ohjelman puolelta. Työ voitiin siis tässä kohtaa todeta valmiiksi.

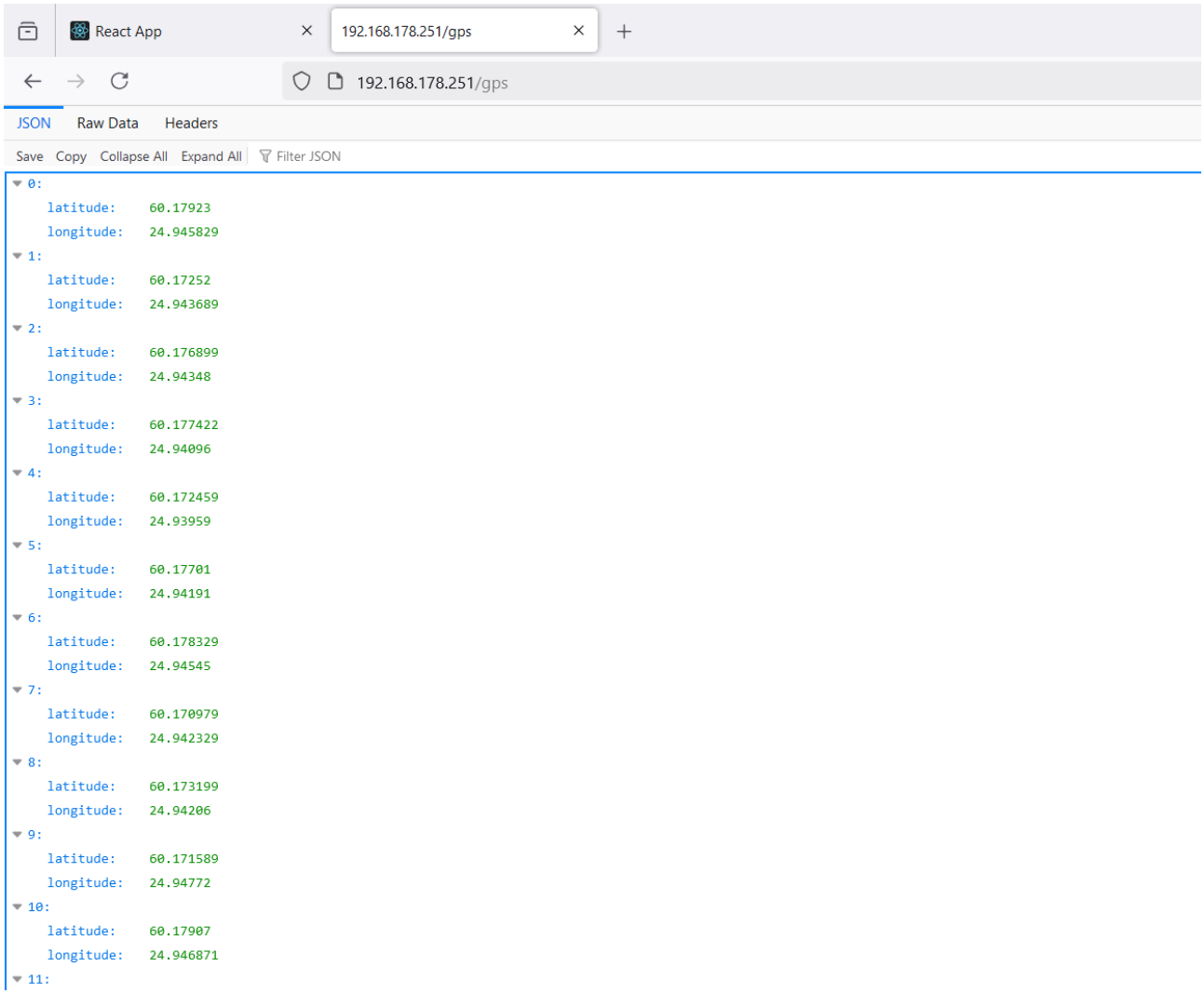
5 TULOKSET

Tämän työn tuloksena saatiin kaksi yhdessä toimivaa ohjelmaa. Yksi on ESP32:n palvelinpuoli, joka kerää, käsittelee ja tallentaa dataa. Tämän jälkeen se lähettää ne saataville luomalleen verkkosivulle. Toinen on Reactilla luotu Leaflet-karttasovellus, joka hakee dataa automaattisesti. Kun se on löytänyt oikeanlaista dataa, se alkaa piirtää sitä kartalle.



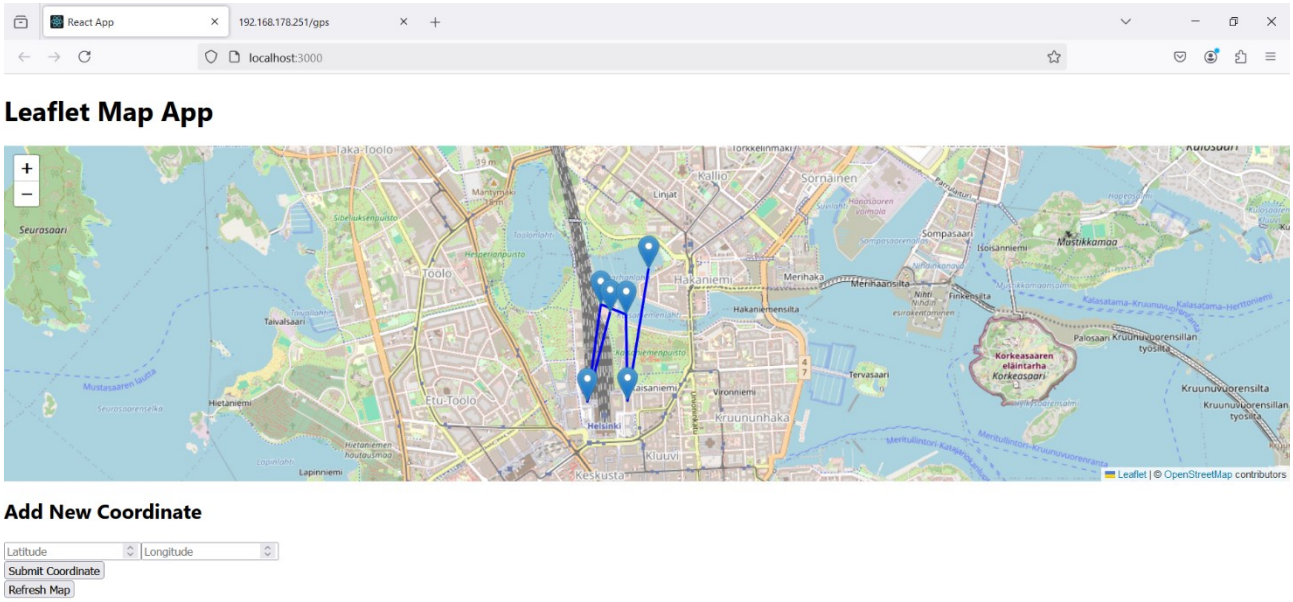
Kuva 2. ESP32:n tulostama simuloitu paikannusdata.

Kuvassa 2 näkyy ESP32:n tuottamaa simuloitua paikannusdataa. ESP32:n tulostuksista voi seurata datan keruuta UART-portilta sekä datan lähetystä eteenpäin. Ensimmäinen http-tuloste kertoo, mitkä koordinaatit lisättiin puskuriin. Se pitää kirjaa siitä, kuinka monta koordinaattia on tallennettu. Toinen http-tuloste pitää kirjaa siitä, kuinka monta koordinaattia on lähetetty. Myös tämä tuloste laskee, kuinka monta koordinaattia se on käsitellyt. Näistä tulosteista voidaan siis tarkastaa, että kaikki koordinaatit tulevat lähetetyksi. UART-tuloste taas kertoo, minkälaiset koordinaatit simuloitiin. UART-portin tulostamien koordinaattien pitäisi näkyä seuraavassa http-tulostuksessa, jotta yhtään koordinaattia ei ole jäänyt välistä.



Kuva 3. ESP32:n HTTP-palvelimen verkkosivu.

Kuvassa 3 näkyy ESP32:n ylläpitämä verkkosivu. Sinne on tulostunut JSON-muodossa latitude- ja longitude-arvot. Arvot on lajiteltu vanhimmasta uusimpaan alkaen koordinaatista numero nolla. Verkko-osoitteena toimii ESP32:n IP-osoite, ja se on löydettävissä samassa WiFi-verkossa olevilla laitteilla.



Kuva 4. Valmis kartta, johon on piirretty koordinaatteja.

Kuvassa 4 näkyy tämän työn lopputulos eli karttaohjelma, johon piirtyy reaaliaikaisesti saadut sijaintitiedot. Sen pohjalla toimii React-sovellus, joka käyttää Leaflet-kirjastoa. Sovellus on localhostina, eli se on näkyvissä ainoastaan laitteella, jolla React-sovellus on ajettu.

6 POHDINTA

Projektia aloitettaessa oli oletuksena, että React-sovelluksen toiminnallisuus saattaisi tuottaa eniten haasteita. Se osoittautui kuitenkin olemaan kaikista helpoin osuus koko projektissa. Kun React-sovellukseen lisättiin toiminnallisuuksia, ne toimivat yleensä ensimmäisellä yrityskerralla. Kun taas ESP32:n koodit vaativat monta yrityskertaa ja testiä, ennen kuin löytyi oikea pätkä korjaavaa koodia. Helpoksi osuudeksi työssä oletettiin olevan lukea dataa Neo-6M-moduulilta, mutta se ei onnistunutkaan. Loppujen lopuksi ohjelmisto saatiin toimivaksi, mutta laitteistoa ei.

Vaikka haluttu lopputulos jäi vajaaksi halutusta lopputuloksesta, on tulos silti toimiva. Haluttu tulos olisi ollut kartta, johon piirtyy oikeat sijaintitiedot reaaliaikaisesti. Lopputuloksena saatiin toki kartta, joka päivittyy reaaliaikaisesti, mutta se päivittyy synteettisellä datalla. Ohjelma voitiin luodulla datalla todeta toimivaksi, mutta osaa mahdollisista toimintatesteistä ei voitu tehdä, koska laitteisto ei toiminut. Olisi ollut hyvä testata muun muassa yhteyden pysyvyyttä ESP32:n ja Neo-6M-moduulin välillä.

Jatkokehityksen kannalta olisi mahdollista yrittää saada GPS-paikannus toimivaksi myös laitetasolla. Tämä voisi onnistua esimerkiksi hankkimalla uusi Neo-6M-moduuli, olettaen että tässä työssä käytetty moduuli oli viallinen. Vaihtoehtoisesti voitaisiin myös testata vaihtaa Neo-6M:n mukana tullut pieni antenni, erilliseen suurempaan antenniin. Tällä voitaisiin poissulkea itse moduulin viallisuus ja syyttää sen sijaan riittämätöntä antennia. Myös Neo-6M:n kytkösten juotoksia voidaan yrittää parantaa. Viimeisenä voitaisiin testata uudempaa Neo-M8-moduulia, jos se toimisi paremmin.

Tästä työstä opittiin paljon sovelluskehityksestä sekä virheenkorjauksesta. Ohjelmiin lisättiin vähän kerrallaan toiminnallisuuksia. Jokaisen lisäyksen jälkeen testattiin ne toimiviksi. Tällä tavalla virheiden korjaus oli helpompaa, kun tiedettiin virheen todennäköisin lähde. Opittiin myös, mitä kaikkia osia ohjelmistoon tarvitaan, jotta kaksi eri laitteilla pyörivää ohjelmaa voivat kommunikoida keskenään. Nämä taidot ovat ohjelmistokehityksen kannalta tärkeitä.

LÄHTEET

Espressif Systems. (2025). *ESP32 Series: Datasheet Version 4.8.* https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Gakstatter, E. (4.2.2015). *What exactly is GPS NMEA data?* GPS World. https://www.gpsworld.com/what-exactly-is-gps-nmea-data/

GitHub. (i.a.). *Build and ship software on a single, collaborative platform*. https://github.com/

Kaplan, E., & Hegarty, C. (2006). *Understanding GPS Principles And Applications* (2. p.). Artech House.

Kurniawan, A. (2019). *Internet of things projects with esp32 : build exciting and powerful iot projects using the all-new Espressif ESP32.* Packt Publishing.

Laricchia, F. (21.10.2024). *GPS tracking device market size worldwide 2020-2025.* Statista. https://www.statista.com/statistics/1385909/gps-tracking-device-market-size-worldwide/

Leaflet. (i.a.) *Overview*. https://leafletjs.com/index.html

Mapbox. (i.a.). *Location intelligence for business*. https://www.mapbox.com/

Miettinen, S. (2006). *GPS Käsikirja* (3. p.). Genimap.

Oner, V. (2021). *Developing IoT Projects with ESP32*. Packt Publishing.

OpenLayers. (i.a.). *A high-performance, feature-packed library for all your mapping needs*. https://openlayers.org/

React. (i.a.). *React: The library for web and native interfaces.* https://react.dev/

U-blox. (2011). *NEO-6: u-blox 6 GPS Modules*. https://content.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf

U-blox. (2022). *NEO-M8: u-blox M8 concurrent GNSS modules.* https://content.u-blox.com/sites/default/files/NEO-M8-FW3_DataSheet_UBX-15031086.pdf

Visual Studio. (i.a.). *Your code editor. Redefined with AI*. https://code.visualstudio.com/