SAOSTUSKAIVOANTURIN PROTOTYYPPI

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tieto- ja viestintätekniikka, biotalous, Forssa

Kevätlukukausi 2021

Mikael Ketonen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Biotalous, tieto- ja viestintätekniikka

Forssa

Tiivistelmä

Tekijä Mikael Ketonen

Vuosi 2021

Työn nimi Saostuskaivoanturin prototyyppi

Ohjaaja Ari Hietala

TIIVISTELMÄ

Opinnäytetyössä määriteltiin, suunniteltiin ja toteutettiin sulautetun järjestelmän prototyyppi, joka mittaa saostussäiliöiden jätetasoa litroissa tai litratilavuudessa. Opinnäytetyö tehtiin Envor Group Oy:n määrittelemien parametrien mukaisesti. Opinnäytetyön osioina olivat palvelukonseptin määrittely, prototyypin eri työvaiheet, kehitysalusta ja sen logiikka, Arduino IDE -ohjelmointitulkin määrittäminen prototyypin ohjelmointiin, C ++ -ohjelmiston rakenne, mitatun datan oikeellisuus ja sen todentaminen prototyypin osien ja rakenteen valintaperusteet. Lisäksi aihealueista syntyneelle prototyypille rakennettiin visuaalinen demo IoT-alustalle.

Opinnäytetyössä keskityttiin sulautetun järjestelmän prototyypin rakentamiseen, NodeMCU-kehitysalustasta ja ultraäänianturista. Sulautetun järjestelmän ohjelmoinnin seurauksena saatiin arvoja saostussäiliötä minimoioivan testiympäristön täyttötasoista. Mittaustiedot reititettiin ja lähetettiin MQTT-protokollalla ThingsBoardiin visualisointia ja mahdollista jatkokäsittelyä varten.

Opinnäytetyöhön rakennettu sulautettu järjestelmä mahdollisti suoraan jatkokehityksen prototyypille, käytännön kenttätestien muodossa. Tämä voi johtaa tehokkaaseen logistiseen reitin optimointiin ja uusiin hintamalleihin Envor Group Oy:lle.

Avainsanat MQTT, ThingsBoard, C++, ESP-12F, Kehitysalusta, Arduino IDE

Sivut 29 sivua ja liitteitä XX sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in ICT, Bioeconomy

Forssa

Abstract

Author Mikael Ketonen

Year 2021

Subject Septic Tank Sensor Prototype

Supervisor Ari Hietala

ABSTRACT

The aim of this thesis was to design an embedded system that measures septic tanks waste level, under the parameters defined by the commissioner, Envor Group Oy. The thesis included the topics on the definition for the service concept, different work steps of the sensor and the micro-processor platform, setting up the Arduino IDE programming interpreter, the structure of the C ++ software, measuring and verifying the measured data. In addition, the aim was to define an IoT platform demo for the prototype, including data visualization of the measured data.

The study focused on building embedded system prototype from NodeMCU V1 development platform and ultrasonic sensor. After the programming of the embedded system, it was possible to gain values from the septic tank fill levels. The received data was routed and sent with the MQTT protocol to the Thingsboard for visualization and possible further processing.

As a result, the embedded system built in this study enables further research for practical field testing. This can lead to efficient logistic route optimization and new price models for Envor Group Oy.

Keywords MQTT, ThingsBoard, C++, ESP-12F, Development Board, Arduino IDE

Pages 29 pages and appendices XX pages

Sisällys

Liitteet

Liite 1 C++ ohjelmistoratkaisu, pinnankorkeus.

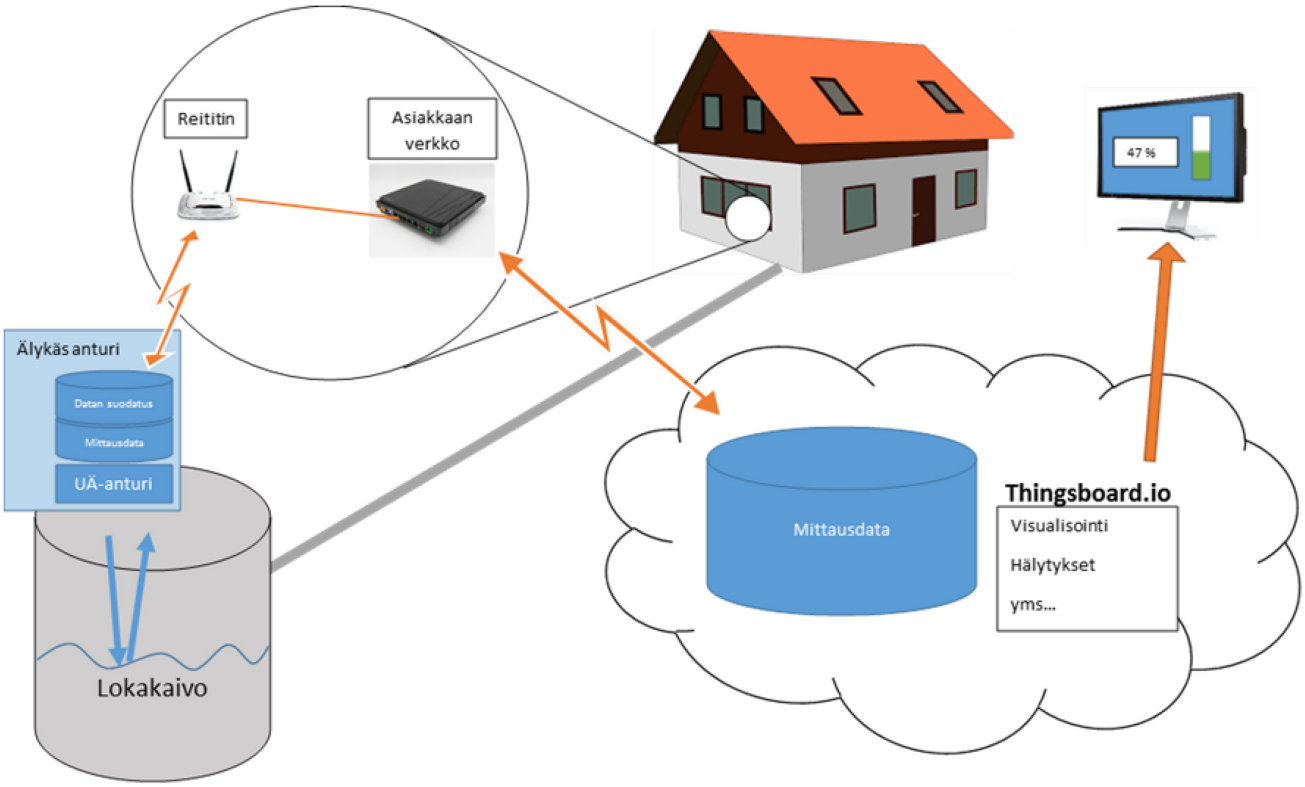
Liite 2 C++ ohjelmistoratkaisu, litratilavuus.

1 Johdanto

Opinnäytetyön tekijänä toimi Hämeen ammattikorkeakoulun biotalouden, tieto- ja viestintätekniikan insinööriopiskelija Mikael Ketonen ja työn toimeksiantajana toimi Envor Group Oy. Opinnäytetyö rakennettiin työn tilaajan asettamasta tutkimuskysymyksestä, "Miten voisimme alentaa loka-autojen logistisia kustannuksia, jotka johtuvat lokakaivojen tyhjennyksestä?". Tutkimuskysymys synnytti kuvassa 1(s.2) olevan idean palvelukonseptista. Konsepti esittää ajatukset siitä, miten opinnäytetyön tilaajan asettama tutkimuskysymys ratkaistaisiin. Palvelukonseptin idea jalostettiin prototyypiksi ja prototyypin testiympäristöksi, joka on lokakaivon tilavuuden seuraamiseen tarkoitettu pinnanmittauksen ohjausjärjestelmä. Rakenteellinen osa tästä työstä demonstroi ohjausjärjestelmän ohjelmoinnin, prototyypin suunnittelun, kokoamisen ja mittaustulosten verifioinnin. Visuaalisessa osassa tätä työtä prototyyppi saa testiympäristöstä dataa, jota käsitellään ja suodatetaan paikallisesti kehitysalustalla, sekä suodatettu data reititetään ja lähetetään MQTT-protokollalla ThingsBoard-alustalle, mahdollista jatkokäsittelyä varten tarkemmin kuvassa 1(s.2).

Työn tarkoitus on edistää palvelusuhteiden jatkokehitystä Envor Group Oy:n asiakaskuntaan ja luoda prototyypin tulevaisuuden jatkokehityksen seurauksena lokakaivon tyhjentämiseen uusia hinnoittelumalleja sekä uudistaa heidän asiakaskunnallensa joustavampia palvelumalleja. Edistämisellä tarkoitetaan tässä työssä logististen kustannusten alentumista mahdollisessa jatkokehittelyssä loka-autoille, mikä johtaa automaattiseen logistiikan reittioptimointiin, jolloin lokakaivon tyhjennys perustuisi lokakaivosta saatuun mittadataan. Automatiikka mahdollistaisi yrityksen asiakkaiden hyödyksi hinta-alennuksen ja asiakkaiden ei olisi enää tarpeen tarkistaa fyysisesti lokakaivon täyttöastetta ja tilata puhelimitse tyhjennystä. Automatiikka mahdollistaisi myös Envor Group Oy:lle asiakaskunnan uudistetun palvelusuhteen, joka hyödyttäisi ajallisesti ja rahallisesti molempia osapuolia.

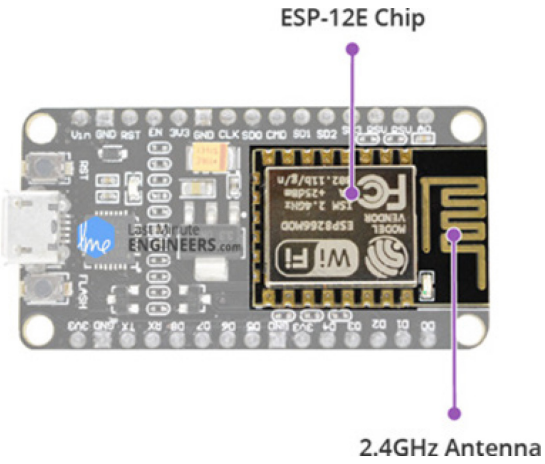
Kuva 1. Palvelukonsepti.



2 Moduuli ja kehitysalusta

Anturia varten tarvittiin mikroprosessorialusta, jolla voitaisiin mahdollistaa ultraäänellä toimiva etäisyyden mittaus. Etäisyysmittauksella haluttiin nimittää arvollisesti kaivon täyttöastetta ja lähettää muodostuva data IoT-alustalle. Moduulin arkkitehtuuriksi valittiin ESP12F-ESP8266 (kuva 2), kehitysalustalla NodeMCU V1 (kuva 3, s. 3).

Kuva 2. ESP-12E (Last Minute ENGINEERS, n.d.).



2.1 Valintaperusteet

Perusteet ESP12F-ESP8266 NodeMCU-V1:n valinnalle olivat halpa hinta ja erittäin laaja harrastajien toimesta syntynyt dokumentaatio erilaisista sulautettujen järjestelmien toteutuksista.

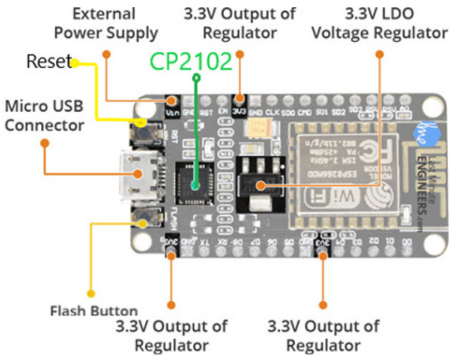
Omakohtaisen kokemukseni mukaan tämänkaltaisella arkkitehtuurilla on rakennettu lukuisia prototyyppejä, jotka käyttivät virtaa erittäin säästeliäästi ja olivat toimintavarmoja. Prototyypistä esimerkkinä on mm. sääasema, jossa lähetysstandardina oli MQTT, alustana ESP01 ja akkuna 3,6 V tasavirralla toimiva ratkaisu. Ratkaisu kesti kenttäolosuhteissa 1,5 vuotta, akun lähtökapasiteetin ollessa 2 450 mAh.

NodeMCU V1 kehitysalustan tärkeimpiä ominaisuuksia tälle työlle olivat, CP2102-usb-to-serial piiri ja jänniteregulaattori, Flash ja reset-kytkin (kuva 3) (NODE MCU TEAM, 2015). Lisäksi työlle olennaisia ominaisuuksia olivat erinäiset toteutukset GPIO-tulo ja lähtötasoissa kehitysalustalla (kuva 4; s. 5).

Kehitysalusta (yllä mainittujen kappaleiden mukaan) teoriassa pystyy käsittelemään anturin lukemaa dataa, tekemään siihen liittyvän laskennan ja lähettämään MQTT-protokollalla käsitellyn datan JSON-formaatissa ThingsBoardiin.

Sulautetun järjestelmän kulutus toiminnallisella ajanjaksolla olisi n. 60 - 100 mAs. Toiminnallinen ajanjakso kestäisi noin 15 s, sisältäen anturin ”aktiivisen” ohjelmallisen osuuden. Sulautetun järjestelmän passiivinen ohjelmaosuus toiminnallisesti kuluttaisi teorian mukaan 0,2 mAs (Ai-Thinker, 2018).

Kuva 3. Kehitysalustan osat (Last Minute ENGINEERS, n.d.).



2.2 Moduuli ja prosessori

ESP-12F, WiFi-moduuli toimii 2,4 GHz:n taajuudella, se tukee protokollia IEEE802.11 b/g/n ja TCP/IP:tä. Moduuli on Ai-thinkerin kehittämää teknologiaa (Ai-Thinker, 2018).

Testitilanteessa havaitsin moduulin pystyvän lähettämään WiFi-signaalia ilmassa ilman esteitä, noin 90 metrin kantaman. Moduuli pystyi lähettämään signaalin näinkin pitkälle, jos piirin oikeassa kulmassa oleva RF-antenni oli asemoitu oikealla tavalla vastaanotinta kohti.

ESP8266:n prosessoriin on sulautettu 32-bittinen MCU, joka on nimeltään Tensilica L106. Prosessorin kello toimii 80 ja 160 megahertsin nopeudella. (Espressif, 2018)

Ydinprosessori tukee RTOS:sia ja integroi sisälleen WiFi/MAC/BB/RF/PA/LNA:n. Kokonaispaketti mahdollistaa GPIO-lähdöllisesti helpon sensorien tai muiden applikaatiopohjaisten laitteiden lisäämisen helposti ja nopeasti. Ominaisuuksien etuna on suunnitteluun kuluvan ajan väheneminen merkittävällä tasolla. Sertifikaatit kokonaisuudella ovat FCC, IC, CE, REACH ja RoHS. (Ai-Thinker, 2018)

Moduuli toimii jännitevälillä 3 - 3,6 V ja käyttää <500 mA virtaa, tukien kolmea eri idle-tilaa. Jokaisessa tilassa on erillinen virrankulutus. Virrankulutus voi olla teoreettisesti alimmillaan 0,02 mA. (Ai-Thinker, 2018)

Huomionarvoista prototyypissä on, että saavuttaakseen 20 µA virrankulutuksen ”Deep Sleep” -tilassa, vaatii ESP-12F-ESP8266 jännitteen 2,5 V. Ai-thinkerin ilmoittama moduulin jänniteväli 3,3 – 5 V mahdollistaa ”Deep Sleep” -tilassa virrankulutuksen <1 mA. (Ai-Thinker, 2018)

2.3 Kehitysalusta ja mikrokontrolleri

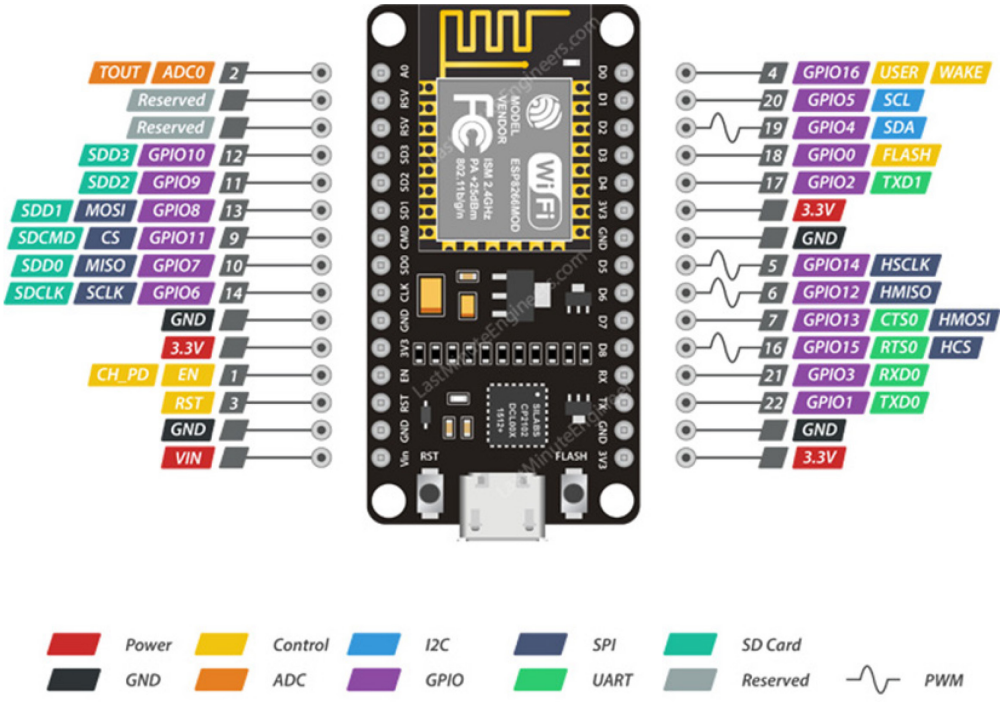
MCU on avoimeen lähdekoodiin perustuva mikrokontrolleri. Mikrokontrollerin ympärillä on V1-kehitysalustan PCB, joka on tarkoitettu prototyypin tekoon. Termi NodeMCU viittaa kirjaimellisesti käytettyyn sulautettuun ohjelmaan alustalla. Termi voidaan kuitenkin purkaa osiin, jossa Node viittaa ohjelmistoon ja MCU mikrokontrolleriyksikköön. Ohjelmisto käyttää

Lua-skriptikieltä ja se on alun perin rakennettu eLua-projektille. Projektin on teettänyt Expressif, nimellä Non-OS SDK. (Ellison, 2017)

2.4 Kehitysalustan GPIO-määrittelyt

NodeMCU antaa pääsyn ESP-12F-8266-V1:n GPIO-lähdöille ja -tuloille, joita on 17. GPIO:sta käyttäjän hyödynnettävissä on vain 11, sillä 6 näistä on varattu yhdistämään Flash-muistisiru (Ai-Thinker, 2018). GPIO:t on määritelty osa-alueittain, mainittakoon ne tässä kategorisesti listana; tulo/lähtö, virta, kontrolli, 12C, SPI, SD kortti, GND, ADC, UART, ei käytössä, varattu Flash, PWM-linja (kuva 4). (NODE MCU TEAM, 2015)

Kuva 4. Kehitysalustan GPIO-kuvaukset (Last Minute ENGINEERS, n.d.).



3 Ultraäänianturi

Projektin yhtenä lähtökohtana oli tarve anturille, joka pystyy mittaamaan lokakaivon täyttöasteen ja maan pinnan välistä etäisyyttä kaivon sisällä, jolloin kaivosta saataisiin mitattua kaivon täyttöaste. Tarkoituksena oli löytää kustannustehokas mutta luotettava ratkaisu, joka olisi veden ja pölynkestävä, kuten V2-anturin tekniset ominaisuudet mahdollistivat (taulukko 1, s. 6). Anturiksi valittiin JSN-SR04T V2.

Taulukko 1. Anturin tekniset ominaisuudet (DFROBOT, n.d.).

Käyttöjännite tasavirralla	3,0 - 5,5 V
Virrankulutus	<8 mA
Resoluutio	0,5 cm
Ultraäänipulssi	40 kHz
Pulssinlähdönkulma	45 - 75 astetta
Käyttölämpötila	-20 - 70 ℃
Suojaluokitus	IP68

3.1 Valintaperusteet

Vertailun kohteeksi valittiin saman valmistajan kaksi eri ultraäänianturia, JSN-SR04T V1 ja V2. JSN-SR04T V2-anturi (kuva 5, s. 7) valittiin, koska sen kyky mitata matkaa kohteeseen oli pidempi, kokonaisuudessaan 1,5 metriä. Anturi mahdollisti järkevän mittausvälin, kun tarkasteltiin yleisesti kotitalouksille myynnissä olevien lokakaivojen tavanomaisia syvyyksiä, kaivoissa pinnan ja pohjan väliset syvyydet asettuvat tyypillisesti 1–6 metrin välille.

V2-Ultraäänianturi pystyy mittaamaan välimatkan 25 - 600 cm. Versio 1 anturista mittaa matkaa samasta aloitus oletusarvosta vain 4,5 metriin asti. V2-anturin virrankäyttö on noin neljäsosa vähemmän kuin V1-anturin. (DFROBOT, n.d.)

Kuva 5. JSN-SR04T V2 (DFROBOT, n.d.).

3.2 Mittaamisen fysiikka

Lyhyt ultraäänipulssi lähetetään ajankohtana 0, joka heijastuu objektista. Anturi vastaanottaa äänisignaalin ja muuntaa sen sähköiseksi signaaliksi. (DFROBOT, n.d.)

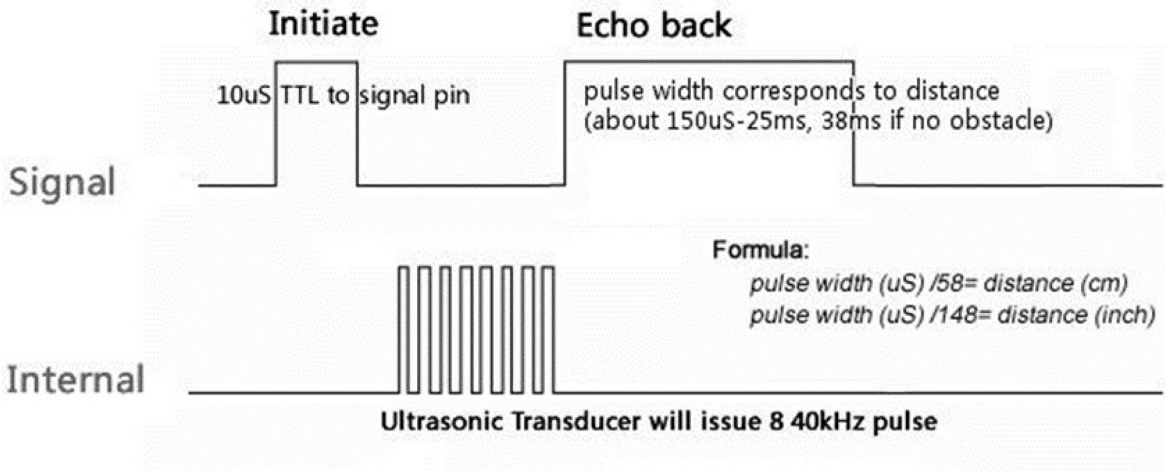
Seuraava pulssi voidaan lähettää, kun kaiku katoaa. Tätä ajanjaksoa kutsutaan syklijaksoksi. Suositusjakson tulisi olla vähintään 50 millisekuntia. (DFROBOT, n.d.)

10 mikrosekunnin levyisen laukaisupulssin lähtiessä signaalipinnille, ultraäänimoduuli tuottaa kahdeksan 40 kHz:n ultraäänisignaalia ja havaitsee mittalukeman kaikupohjasta takaisin lähtöpisteeseen, suurimman palautuvan kaiun perusteella. (DFROBOT, n.d.)

Mitattu etäisyys on verrannollinen kaiun pulssin leveydelle ja voidaan laskea alla olevassa kuvassa 6 esitetyllä kaavalla. (DFROBOT, n.d.)

Ulostulopinni antaa 38 ms: n ylätason signaalin, mikäli fyysistä estettä ei havaita (DFROBOT, n.d.).

Kuva 6. Anturin looginen fysiikka (DFROBOT, n.d.).



4 Prototyyppi

Prototyypin suunnittelemiseen käytettiin seuraavia osia: ESP-12E, JSN-SR04T V2, 3 kpl Ikean HR6 ladda 2450 mAh:n akkuja tasavirralla 1,6 V (kuva 7). Prototyypissä käytettiin seuraavia fyysisiä osia: ESP-12F-8266 NodeMCU-V1, JSN-SR04T V2, ja muuntaja, joka muuttaa 230 V:n vaihtojännitteen 3,3 - 5 V:n tasajännitteeksi (kuva 8, s. 10).

4.1 Suunnittelu

Alustan kytkennät suunnitteluvaiheessa olivat seuraavanlaiset: ESP-12E:n GPIO0 kytketään ESP-12E:n RST-napaan, ESP-12E:n GPIO04 kytketään JSN-SR04T V2:n echo-napaan, ESP-12E:n GPIO5 kytketään JSN-SR04T V2:n trig-napaan, ESP-12E:n GPIO14 kytketään JSN-SR04T V2:n

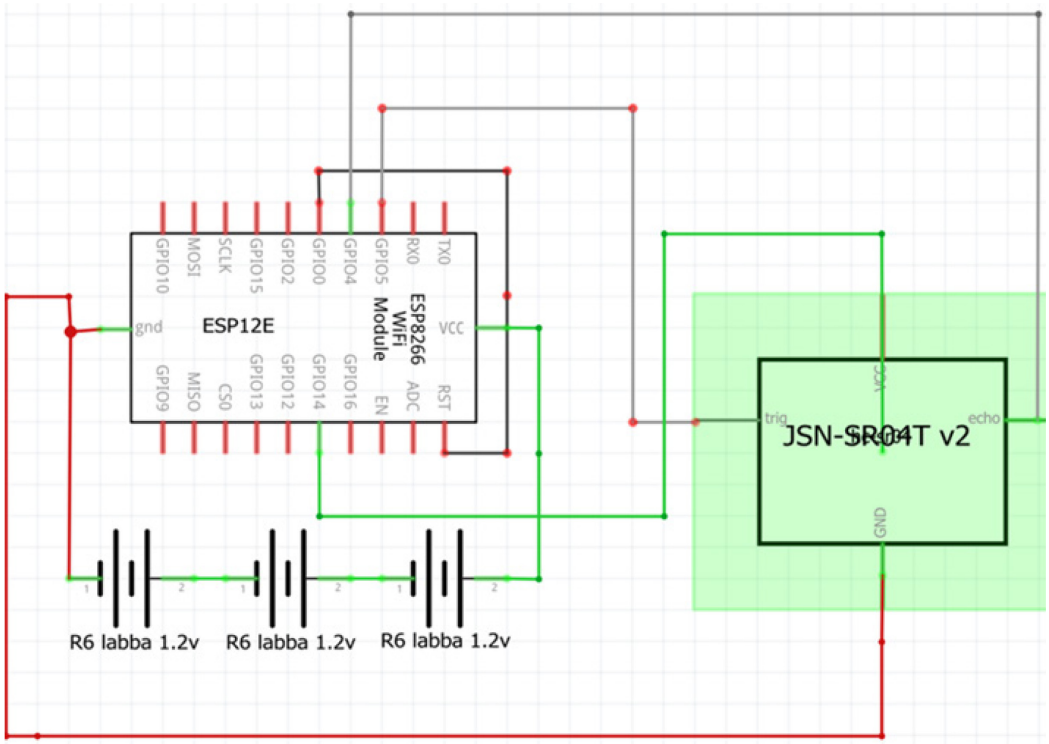
VCC-napaan. ESP-12E:n gnd kytketään JSN-SR04T V2:n gnd-napaan, ESP-12E:n gnd kytketään kolmen HR6-akun sarjaan, sarjasta pluskytkentä ESP-12E:n VCC-napaan (kuva 7).

Kytkentöjen suunnittelu tehtiin tällä tavoin, jotta ultraäänianturin virranohjaus olisi mahdollista ESP-12E:n GPIO-lähtöjen tasolla (Ai-Thinker, 2018). Kytkennät mahdollistaisivat kehitysalustan logiikalle säätää anturin virtaa fyysisesti päälle ja pois.

Kehitysalustan logiikka suunniteltiin ohjelmistossa menemään toiminnallisesti passiiviseen tilaan. Suunnitelman mukaan mittaussuorituksen jälkeen ohjelmisto ohjaisi GPIO14:n virransyötön alas. Mielestäni oli hyvin perusteltua suunnitella kytkennät edellä mainitulla tavalla, jotta arkkitehtuuri olisi järkevä, sillä se ei teoriassa silloin kuluttaisi enempää virtaa kuin opinnäytetyön luvussa 2.2, sivulla neljä mainitaan.

Anturin valmistajan suosittelemalla tavalla tehtynä kytkennät nukkumistilassa kuluttaisivat kytkennässä kokonaisuudessaan anturin osalta virtaa 8 tai yli 8 mA (DFROBOT, n.d.). Valmistajan tavasta poiketen nukkumistilassa anturin virran syötön ollessa kytkettynä GPIO14, anturi ei kuluttaisi tällöin teoriassa mitään. (Espressif, 2018)

Kuva 7. Kytkentäkaavio.



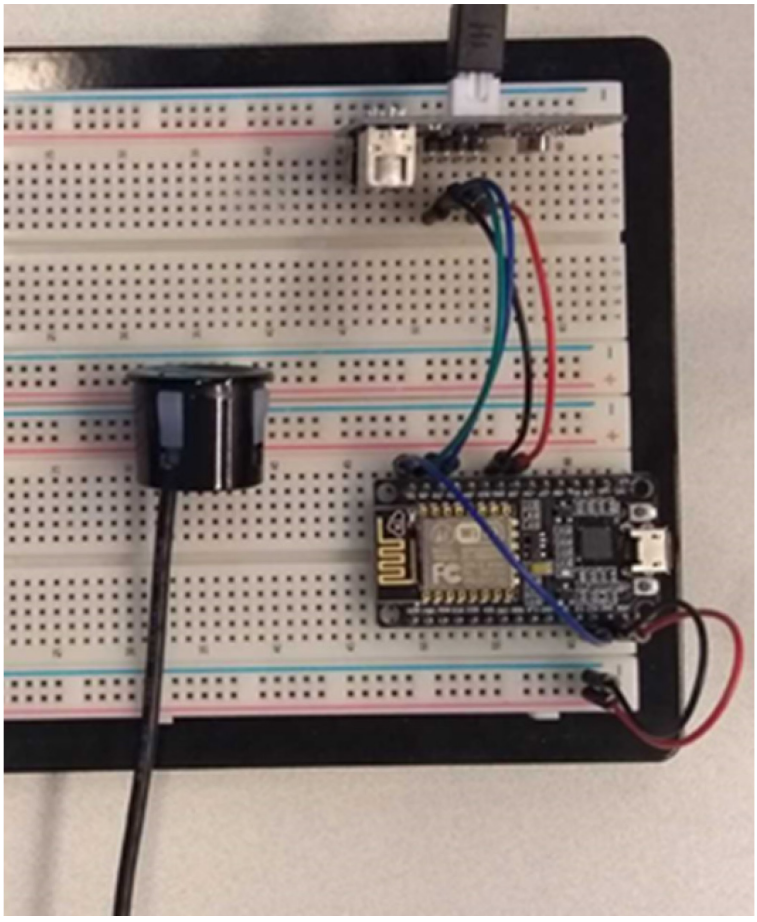
4.2 Tulos

Suunnitelman perusteella rakennettiin prototyyppi, jonka kytkennässä käytettiin seuraavia osia: ESP-12F-8266-NodeMCU-V1, JSN-SR04T V2, AC-verkkovirrasta tasasähköön muuntaja 3,3 - 5 V (kuva 8).

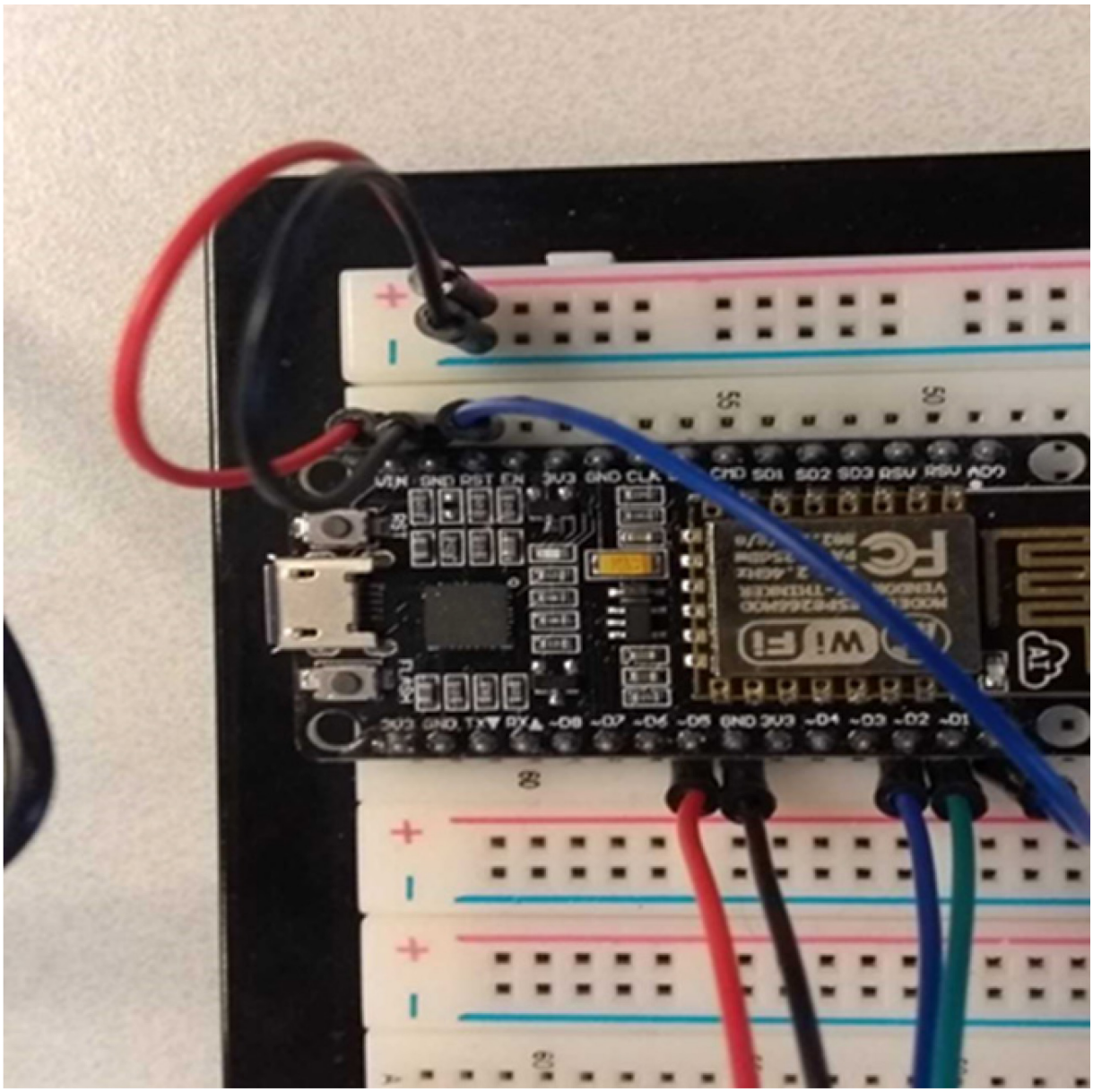
Alustan kytkennät kytkettiin seuraavasti: Kehitysalustan GPIO0-napa kytkettiin kehitysalustan RST-napaan, kehitysalustan GPIO04-napa kytkettiin ultraäänianturin echo-napaan, kehitysalustan GPIO5-napa kytkettiin anturin trig-napaan, kehitysalustan GPIO14-napa kytkettiin anturin VCC-napaan, kehitysalustan gnd-napa kytkettiin anturin gnd-napaan, kehitysalustan gnd-napa kytkettiin muuntajaan, muuntajasta kytkentä tehtiin kehitysalustan VCC-napaan (kuva 8; kuva 9, s. 11).

Kytkennät tehtiin tällä tavoin, koska haluttiin suunnitelman mukaisesti mahdollistaa ultraäänianturille virranohjaus, ESP-12E:n GPIO-lähtöjen tasolla (kuva 8).

Kuva 8. Kytkentäalusta.



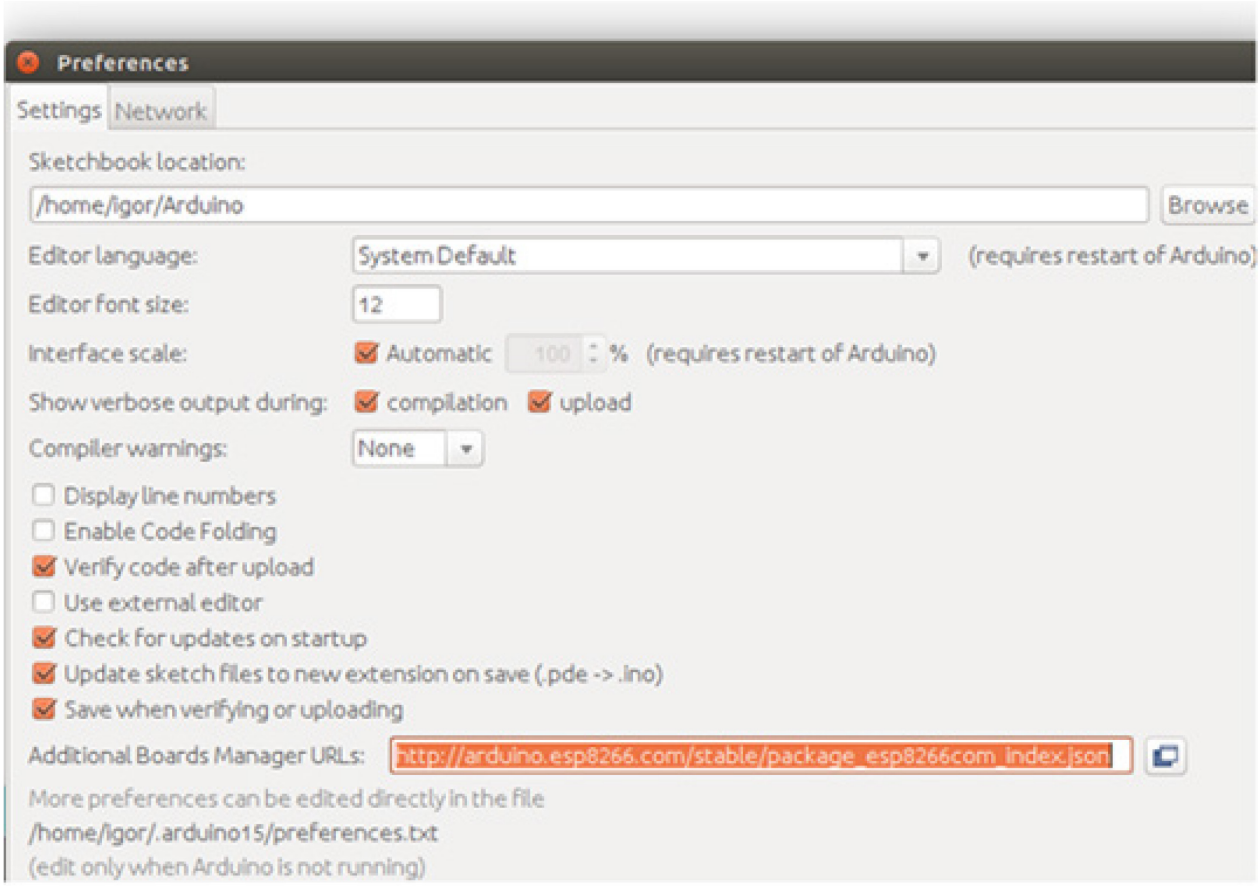
Kuva 9. Kytkentäalusta 2.



5 Ohjelmointitulkin määrittely

Kehitysalustan ohjelmointiin ja alustalta anturin ohjaamiseen tarvittiin Arduino IDE:n asennus. Kehitysalustan ohjelmointia varten ohjelmointitulkki vaati ohjelmistokirjaston. Haku tehtiin ’’Additional Boards Manager URLs’’ syöttövalikkoon kuvassa 10. (Grokhotkov, 2017)

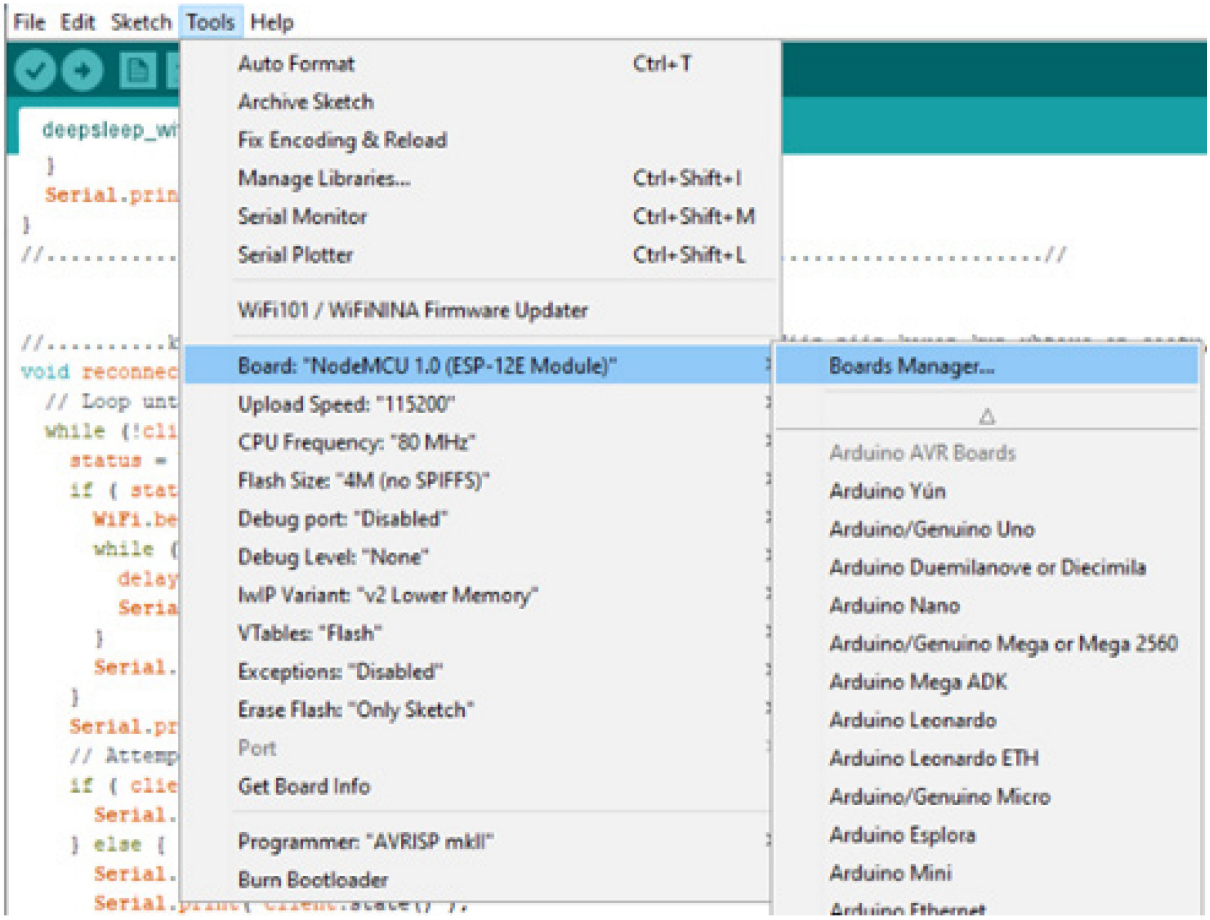
Kuva 10. Asetukset.



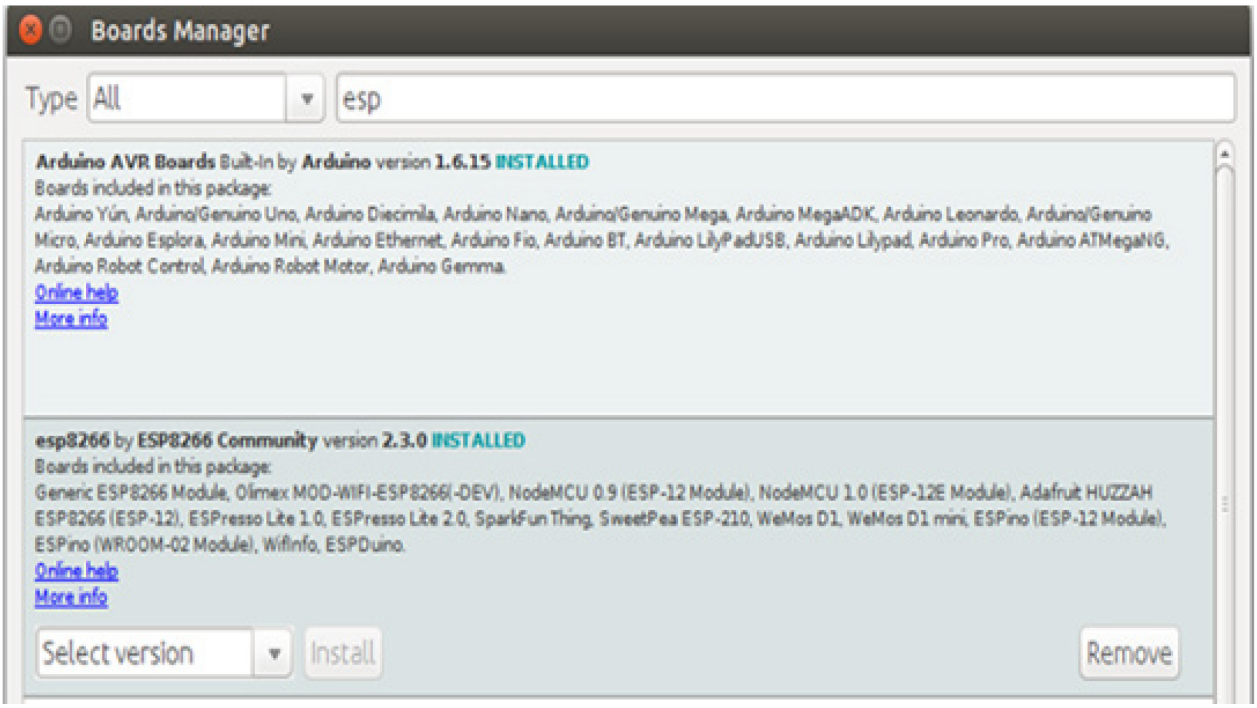
Board manager:in kautta haettiin ESP:n kirjastot. Board manager valinta näkyy kuvassa 11, lisäksi valikkotyökaluissa valittiin oikea käytössä oleva sarjaportti. Latausnopeus asetettiin 115200 baudiin. (kuva 11, s. 13). (Grokhotkov, 2017)

Etsintävalikkoon kirjoitettiin ESP, tämän jälkeen valittiin kirjasto nimeltä ”ESP8266 By ESP8266 community” ja kirjasto ladattiin ohjelmistotulkille (kuva 12, s. 13). (Grokhotkov, 2017)

Kuva 11. Arduino IDE:n boardmanager.

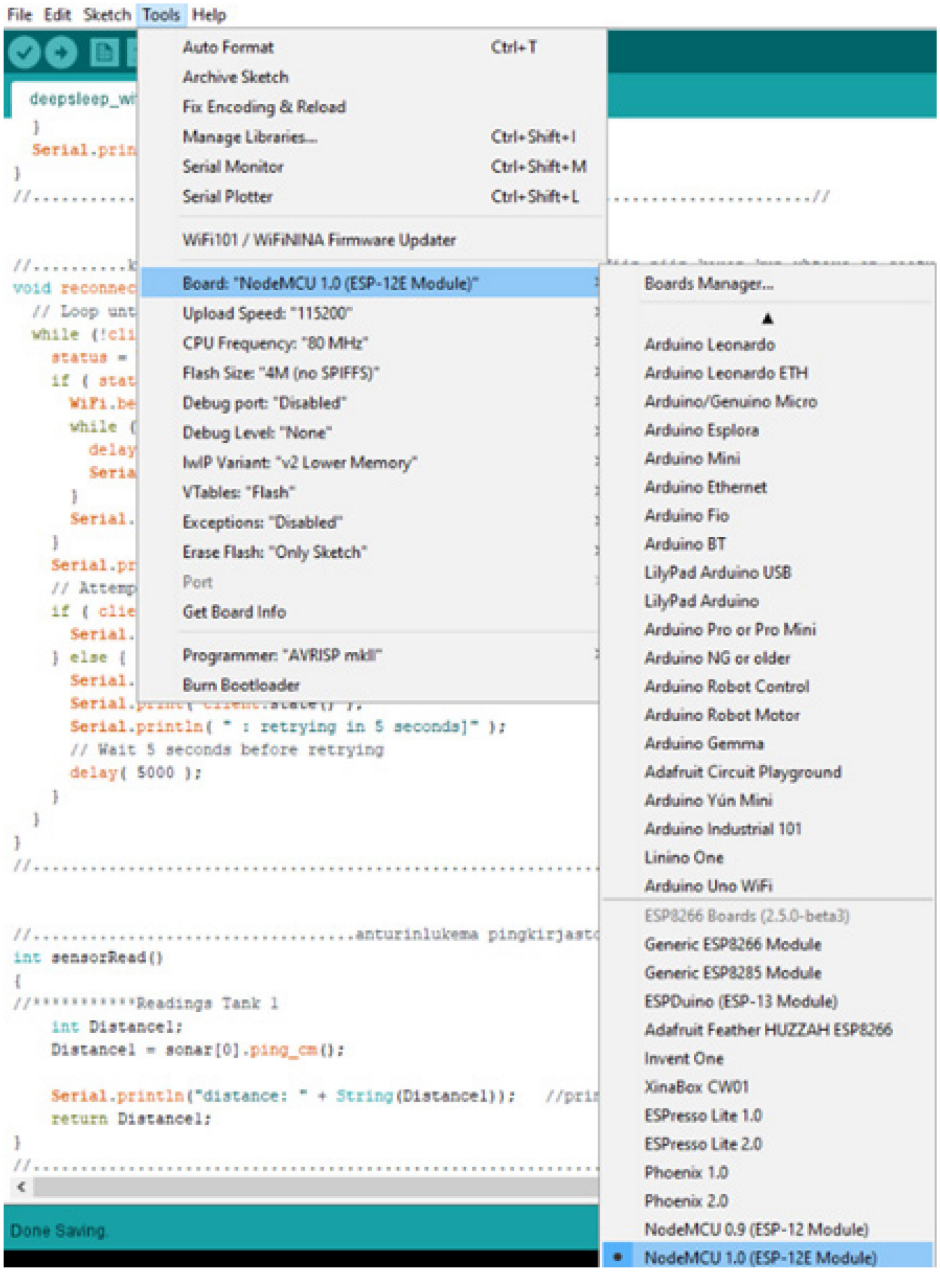


Kuva 12.Boards manager.



Asennuksen jälkeen Arduino IDE:n valikkotyökaluihin tuli uusia mikroprosessorialustojen vaihtoehtoja, joista valittiin ”ESP8266, ESP-12E Module” (kuva 13). (Grokhotkov, 2017)

Kuva 13. Arduino IDE, mikroprosessorialustan valinta.



Silabs:in piirin CP2102, USB-serial-ajuri, ESP12 USB-UART CH341SER täytyi asentaa, jotta kehitysalustan tunnistus tapahtuisi ohjelmistotulkille USB-portin kautta (SILICON LABS, 3.9.2020). Kehitysalustan Flash-muistipiirille koodattu oma ohjelmisto voidaan asentaa vain, jos CP2102-ajuri on asennettu.

6 Ohjelmisto

Ennen varsinaista ohjelmistokoodia laskemisoperaatiolle piti ohjelmistotulkkiin sisällyttää ohjelmiston toimintoihin käytettävät ohjelmistokirjastot, tarkemmin kuvassa 14. (Arduino.cc, n.d; Eckel, 2016; katso myös O'Leary, 2020).

Kuva 14. Ohjelmistokirjastot.

```
#include <math.h>  //matematiikka kirjasto
#include <PubSubClient.h> //Mqtt kirjasto
#include <ESP8266WiFi.h> // wifi määritys kirjasto
#include <NewPing.h> // Ajoituksen yksinkertaistaminen
```

Ohjelmistossa määriteltiin pysyvät muuttujat, anturien lukumäärä, mittauksen enimmäisvälimatka, muuttujan arvo merkittiin yksikössä "cm". Määriteltiin muuttuja "nMeasures" merkitsemään yhden mittauskerran mittauksien lukumäärä. Määritys sisälsi myös WiFi-muuttujat (Grokhotkov, 2017). (kuva 15)

Kuva 15. Muuttujat.

```
#define SONAR_NUM 1

const int MAX_DISTANCE = 600;
const int nMeasures = 20;

WiFiClient wifiClient;
PubSubClient client(wifiClient);

int status = WL_IDLE_STATUS;
unsigned long lastSend;
```

Tämän vaiheen jälkeen määriteltiin pysyvät muuttujat. Muuttujat määriteltiin vastaamaan anturin pinnien gnd-, echo-, trigger- sekä jännitenavan kytkentää mikroprosessorin GPIO-lähtöihin (kuva 16).

Kuva 16. Sensorin muuttujat, GPIO-lähtö ja -tulo.

```
//.....................sensorin pinnien määritys....................//

const int PingPin1 = 5;                          //     GPIO5,  D1
const int EchoPin1 = 4;                         //     GPIO4,  D2
const int SensorPower1 = 14;                   //GPIO14, D5
```

Muuttujien kuvassa 16 kytkentämäärittelyjen jälkeen, muuttujanimet kerrottiin NewPing.h kirjastolle (kuva 17).

Kuva 17. NewPing-muuttujat.

```
NewPing sonar[SONAR_NUM] = {
  NewPing(PingPin1, EchoPin1, MAX_DISTANCE)

};
```

Seuraavaksi suoritettiin itse mittaus, joka suoritettiin 20 kertaa. Mittauskertojen väliin ohjelmoitiin 80 millisekunnin viive, varmistamaan mittaussignaalin häiriötön lähtö ja tulo. (kuva 18)

Kuva 18. Mittaus anturilta.

```
.........mittaukset anturilta.............//

int val[nMeasures];
int filteredVal = 0;
for(int i = 0; i < nMeasures; i++) {  //mittaa
  val[i]= sensorRead();  //aliohjelma
  delay(80);}
```

Mittauksen prosessissa kuvassa 19, aliohjelma ”int SensorRead”, palautti n. 80 litran testiastian pohjan ja pinnan etäisyyden senttimetreinä. NewPing.h kirjastoa hyödynnettiin aliohjelmassa (Eckel, 2016).

Kuva 19. SensorRead-aliohjelma.

```
//.................................anturinlukema pingkirjastofunktiota centtimetreiksihyödyntän, aliohjelma.....
int sensorRead()
{
//***********Readings Tank 1
    int Distance1;
    Distance1 = sonar[0].ping_cm();                                    //Hakee pohjan ja pinnan välimatkan cm yksiköss

    Serial.println("distance: " + String(Distance1));   //printtaa syvyyden sarjaportiin
    return Distance1;
}
```

Sensorpower-muuttuja määritettiin kuvassa 20 ”OUTPUT” tilaan, joka mahdollistaa anturille sähkövirran, kehitysalustan GPIO14-navan kautta. Anturin ja GPIO14-lähdön kytkentä näkyy kytkentäkaaviosta kuvassa 7.(s.9).

Kuva 20. Anturin virransyötön määritys.

```
  while(!Serial) { }
  pinMode(SensorPower1, OUTPUT);
  digitalWrite(SensorPower1, HIGH);
  Serial.println("I'm awake.");
```

Aliohjelma ”Float SensorRead”, kertoi puolikartion mallisen säiliön litratilavuuden (kuva 21).

Kuva 21. Litratilavuus puolikartiosäiliössä.

```
float sensorRead()
{
//***********litrat demo säiliöstä
    float Distance1, Litrat, Level, xR, keskiR;
    float pHalk = 39.5;
    float yHalk = 49.0;
    float pii = 3.14159;

    Distance1 = sonar[0].ping_cm();    //pinnan korkeus
    Level = MAX_DISTANCE - Distance1;
    xR = (((yHalk-pHalk)/2)/MAX_DISTANCE * Level) + (pHalk/2);  // säde korkeudessa
    keskiR = (xR + (pHalk / 2))/2;
    Litrat = (pii * keskiR* keskiR * Level)/1000;


    Serial.println("distance: " + String(Distance1));   //tulostasyvyys
    Serial.println("litrat: " + String(Litrat));        //tulosta litrat

    return Litrat;
}
```

6.1 Suodatus järjestykseen

Mittaustulokset järjestettiin pienimmästä suurimpaan, väliaikaisen muuttujan avulla (kuva 22).

Kuva 22. Luokittelu.

```
//.............järjestä arvot järjestykseen jotta voidaan suodattaa.....................//
   for(int i = 0; i < nMeasures - 1; i++) {
    for(int j = 0; j < nMeasures - i - 1; j++) {
      if (val[j] > val[j+1]) {
        int temp = val[j];
        val[j] = val[j+1];
        val[j+1] = temp;
      }
    }
   }
//.....................................................................................//
```

6.2 Keskiarvo ja persentiili

Mittaustulosten järjestelemisen jälkeen, mittaustuloksista vähennettiin kolme lukuarvoa, pienemmästä päästä ja suurimmasta. Vähennysoperaatio on nimeltään persentiili. Vähennysoperaation jälkeen voitiin suorittaa keskiarvon laskenta. (kuva 23)

Tämä menettelytapa valittiin, koska lokakaivoissa saattaa olla pienimuotoisia epätasaisuuksia. Suodattamalla mittaustulokset saatiin lukuarvolliset pinnankorkeudet senttimetreissä todenmukaisemmin.

Kuva 23. Keskiarvo ja persentiili.

```
//.....................keskiarvo ja presentiili laskenta.................................//
     for(int i = 3; i < nMeasures-3; i++) {     // laske yhteen arvot 3...nMeasures-3
    filteredVal += val[i];
   }
   filteredVal = filteredVal/(nMeasures-6);   //laske keskiarvo

  Serial.println(filteredVal); // printtaa tulos
//.....................................................................................//
```

6.3 Passiivinen- ja aktiivinen tila

Kehitysalustalla mittausprosessin jälkeen ei ollut enää tarvetta suorittaa mikroprosessoria ja muita kytkettyjä elektroniikkoja aktiivisesti päällä, tällöin voitiin kytkeä virrat pois ja asettaa kehitysalusta passiiviseen tilaan. Passiivinen tila tarkoittaa ohjelmallisesti kehitysalustan asettamista ”nukkumaan”, eli ”deep sleep” -tilaan kuvassa 24 (Espressif, 2018).

Kehitysalustan aktiivinen tila voidaan käynnistää ohjelmallisesti, kun alustan fyysinen kytkentä tehtiin GPIO0-navasta, GPIORST-napaan (kuva 7, s. 9). Ohjelmistossa aktiivinen tila käynnistyy passiivisen tilan jälkeen, kuvassa 24 ilmenee passiivinen tila ”ESP.deepSleep”.

Testauksessa kehitysalustan ohjelmistossa määriteltiin kehitysalustan mikroprosessorin sisäinen kello laskemaan 20 sekuntia, jonka jälkeen alusta heräsi aktiiviseen tilaan. Nukkumisaika asetetaan yksikössä µS. (kuva 24)

Kuva 24. Deep sleep.

```
/.....................deepsleep......................................./
 Serial.println("Menen 20sekunnin unille, 2.5 sekunnin päästä");
 delay(1900);
 ESP.deepSleep(20e6); // 20e6 is 20 seconds
/.....................................................................//
```

6.4 Tulos ja verifiointi

Mittaustulos varmistettiin täyttämällä tietty litramäärä hiekkaa testiastiaan. Määrä lisättiin manuaalisesti ja se osoittautui samaksi anturilla mitattuna anturin resoluution tarkkuuden mukaisesti.

Pinnankorkeus mitattiin manuaalisesti puisella tikulla, jonka mitta oli mittanauhalla mitattu ja kirjattu fyysisesti 1 cm:n välein mittatikun kylkeen. Mittatikun mittojen tasalle muodostunut sannan pinnankorkeus vastasi ultraäänianturin ilmoittamaa lukemaa. Anturin antama tulos oli manuaaliseen mittaan verrattaessa, anturin valmistajan ilmoittaman +-1 cm:n virhemarginaalin sisällä.

Testisäiliön tilavuus oli 86,2 litraa, mittaustulokset litrallisesti olivat lähes samat, kuin mitä astian valmistaja ilmoitti tilavuudeksi. Tilavuuden arvollinen heitto 7 % syntyi testiastian ollessa täynnä, se oli enemmän mitä testiastian valmistaja ilmoittaa astian tilavuudeksi. Tämä johtui siitä, että tämän tyyppisissä muovisissa roska-astioissa on lähes aina noin valmistajan +5–10 % ”vara” tilavuudessa, kyseessä on mitä ilmeisemmin astian valmistustekniikkaan liittyvistä seikoista.

Matematiikalla laskettaessa manuaalisesti, ei testiastian tilavuudessa synny heittoa ja anturilla mitattaessa tulos on anturinvalmistajan ilmoittaman virhemarginaalin sisällä. Mitat matematiikkaan mitattiin manuaalisesti, sisältäen testiastian pinta- ja pohjahalkaisijan, sekä niiden välisen korkeuden.

7 IoT-alusta

Tässä opinnäytetyössä kehitysalustan IoT-ratkaisu toteutettiin ThingsBoard-palvelussa. Valintaperusteita IoT-alustalle olivat: nopea sulautetusta järjestelmästä saadun datan demoaminen, yksinkertainen ja nopea sisäistettävä kehitysalustan datan integroimiseen, yleisien lähetysprotokollien tuki kuten http ja MQTT, lisäksi datan käsittelymuotona olisi hyvä olla JSON.

Biotalouden insinöörikoulutuksessa on aikaisemmin käytetty yleisesti ThingsBoard-alustaa. Etuna luettakoon, että demoympäristön toteuttaminen on ilmaista ja se oli työn suunnitteluvaiheessa yleisesti suositeltu, hyväksi todettu korkeakoulun sisällä ja se tuki silloista yleistä IoT-mallia.

ThingsBoard tuki sisältää SQL:n, PostgreSQL:n ja No SQL:n, lisäksi alustalla voi käyttää Cassandra-tietokantoja. Lähetysprotokollat ja muut tiedon käsittelytavat ovat myös valintaperusteiden mukaisia. (ThingsBoard, 2021a)

Työn tekovaiheessa muut saatavilla olevat alustat olivat huomattavasti kalliimpia ja monimutkaisempia ratkaisuja, kuin ThingsBoard.

7.1 MQTT

MQTT on kevyt julkaisu- ja tilausviestintäprotokolla, joten sen oppiminen oli nopeatempoista, prototyypin datansiirto verkossa käytti tätä lähetysstandardia.

7.2 Kehitysalustan määrittely

Kehitysalusta määriteltiin ohjelmallisesti ottamaan yhteyttä WLAN-verkossa olevaan yhdyskäytävään (kuva 25; kuva 26; kuva 27).

Kuva 25. Paikallisen verkon määritys.

```
//................................wifikäytävän määritys(ssid, salasana).........
#define WIFI_AP "HAMKvisitor"
#define WIFI_PASSWORD "hamkvisitor"
//.....................................................................//
```

Kuva 26. Julkaisutavan määritys yhdyskäytävän kautta.

```
char thingsboardServer[] = "demo.thingsboard.io"; //thingsboardin dns osoite tai ip

//.................wificlientin märittelyä.........................//
WiFiClient wifiClient;
PubSubClient client(wifiClient);
int status = WL_IDLE_STATUS;
unsigned long lastSend;
//.....................................................................//
```

Kuva 27. Yhteydenmuodostus paikalliseen yhdyskäytävään.

```
//.............wifin avulla yhdyskäytävään ja yhteydenmuodostus aliohjelma.
void InitWiFi()
{
  Serial.println("Connecting to AP ...");
  // attempt to connect to WiFi network

  WiFi.begin(WIFI_AP, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected to AP");
}
```

WLAN-Reititin välittää tiedon internetin yli, Thingsboard-palvelimelle ja palvelimen Nodet toimivat MQTT-välittäjänä, liittyvä koodi kuvassa 28. Nodet tukevat QoS-tasoja 0 ”korkeintaan kerran”, tasoja 1 ”vähintään kerran” ja joukon ennalta määritettyjä aiheita (ThingsBoard, 2021b). Internetin yli tapahtuvaa yhteyttä yritetään muodostaa kehitysalustalta ohjelmallisesti IoT-alustalle, niin monesti kunnes yhteys on muodostunut (kuva 29).

Kuva 28. Client.setServer.

```
//..käynnistellään initwifi aliohjelma jonka jälkeen avataan yhteys MQTT thingsboariin.
InitWiFi();  //aliohjelma
client.setServer( thingsboardServer, 1883 );
lastSend = 0;
// Wait for serial to initialize.
  if ( !client.connected() ) {
  reconnect();
}
while(!Serial) { }
pinMode(SensorPower1, OUTPUT);
digitalWrite(SensorPower1, HIGH);
Serial.println("I'm awake.");
//.....................................................//
```

Kuva 29. Yhteyden muodostus, kunnes ThingsBoard vastaa.

```
void reconnect() {
  // Loop until we're reconnected
  // **********************   tähän pitää laittaa rajattu määrä yrityskertoja!
  while (!client.connected()) {
    status = WiFi.status();
    if ( status != WL_CONNECTED) {
      WiFi.begin(WIFI_AP, WIFI_PASSWORD);
      while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
      }
      Serial.println("Connected to AP");
    }
    Serial.print("Connecting to ThingsBoard node ...");
    // Attempt to connect (clientId, username, password)
    if ( client.connect("ESP8266 Device", TOKEN, NULL) ) {
      Serial.println( "[DONE]" );
    } else {
      Serial.print( "[FAILED] [ rc = " );
      Serial.print( client.state() );
      Serial.println( " : retrying in 5 seconds]" );
      // Wait 5 seconds before retrying
      delay( 5000 );
    }
  }
}
```

Tieto siirretään JSON-formaatissa IoT-alustalle ja se tallentaa paikallisesti tiedon ohjelmistopolun juureen, "v1/devices/me/telemetry" (kuva 31). Kehitysalustan ohjelmistossa määritettiin alustalle uniikki laitekohtainen ”Access token”, jotta lähetetyt datat tallentuisivat paikalliseen IoT-alustan juureen myös laitteistokohtaisesti (kuva 30). (ThingsBoard, 2021b)

Jokaisen kehitysalustan laitteistokohtainen ”Access Key”, generoidaan ThingsBoard:in käyttöliittymässä. Laitteistokohtaista tunnusta luodessa piti mennä kohtaan ”devices” ja painaa ”+” merkkiä kohdassa ”new device”, tällöin välilehti device-paneelissa näytti luodulle laitteelle uniikin ”Access tokenin”, joka kopioitiin (kuva 30) ohjelmistokoodiin varatulle ”#define” paikalle.

Kuva 30. Access token.

```
#define TOKEN "o2jnsqUSyQo9w1UdL4Qg"                    //access token thingsboardiin
```

7.3 JSON

ThingsBoard:ia käyttäessä helpoin vaihtoehto nopeaan demoamiseen oli, muuttaa lähetettävä data JSON-formaattiin (kuva 31).

ThingsBoard tallensi tiedon paikalliseen vakiojuureensa (kuva 33, s. 25), kansion alle nimeltä ”attributes” (ThingsBoard, 2021b). Tiedostoa voitiin kutsua visualisointiin ThingsBoard-käyttöliittymän DASHBOARD-olio valinnassa, eli entity-valintaikkunassa (kuva 33, s. 25), valitsemalla Access tokenin osoittama laite ja ”timeseries” muuttuja (kuva 33, s. 25; katso myös kuva 30).

Kuva 31. JSON-formatointi.

```
//.............Prepare a JSON payload string....eli muunnetaan json muotoon.
  String payload = "{";
  payload += "\"distance\":"; payload += filteredVal;
  payload += "}";
  //joshalutaan lisää payloadeja
  //lisätään suluissa oleva asia ilman sulkuja (payload; += ",";)
//.....................................................................
```

IoT-alustalle määritettiin kehitysalustan ohjelmistossa oletustiedostopolku datan tallennukseen (kuva 32) ja syötettiin ohjelmistoon haluttu kehitysalustan ohjelmistomuuttuja nimi JSON-formaattiin (kuva 31). Telemetriadata näin menetellen muokkaantui rakenteellisesti oikeaan tiedostomuotoon ja tiedostojuureen IoT-alustalla, eli JSON-muotoon IoT-alustan tiedostojuuressa, ”v1/devices/me/telemetry”.

Kuva 32. Lähetetään JSON.

```
char attributes[100];
payload.toCharArray( attributes, 100 );
client.publish( "v1/devices/me/telemetry", attributes );
Serial.println( attributes );
```

7.4 Tulos

ThingsBoard pystyi käyttämään määritysten jälkeen dataa visualisointiin, tulos ilmenee kuvassa 34, kutsuttaessa (kuva 33) telemetriatietoa lähdepolkupaikastaan. Kehitysalustan määrityksien jälkeen telemetriadata oli ”julkaisukelpoinen”.

Kehitysalustan datan visualisointiin tehtiin DASHBOARD-demo IoT-alustan käyttöliittymää käyttäen, vaihtoehtoisesti olisi voitu valita jo olemassa oleva DASHBOARDS-pohja. Termillä ”DASHBOARD”, tarkoitetaan tässä työssä prototyypiltä saadun datan visuaalista ympäristöä IoT-alustalla.

Prototyypin visuaalisen ympäristön määrittelemisessä noudatettiin kokeile ja opi prosessia. Prosessin seurauksena ympäristölle laadittiin toiminnallinen ohje, joka on esitetty kuvassa 33, kuva on numeerisessa järjestyksessä 1 - 5 välillä.

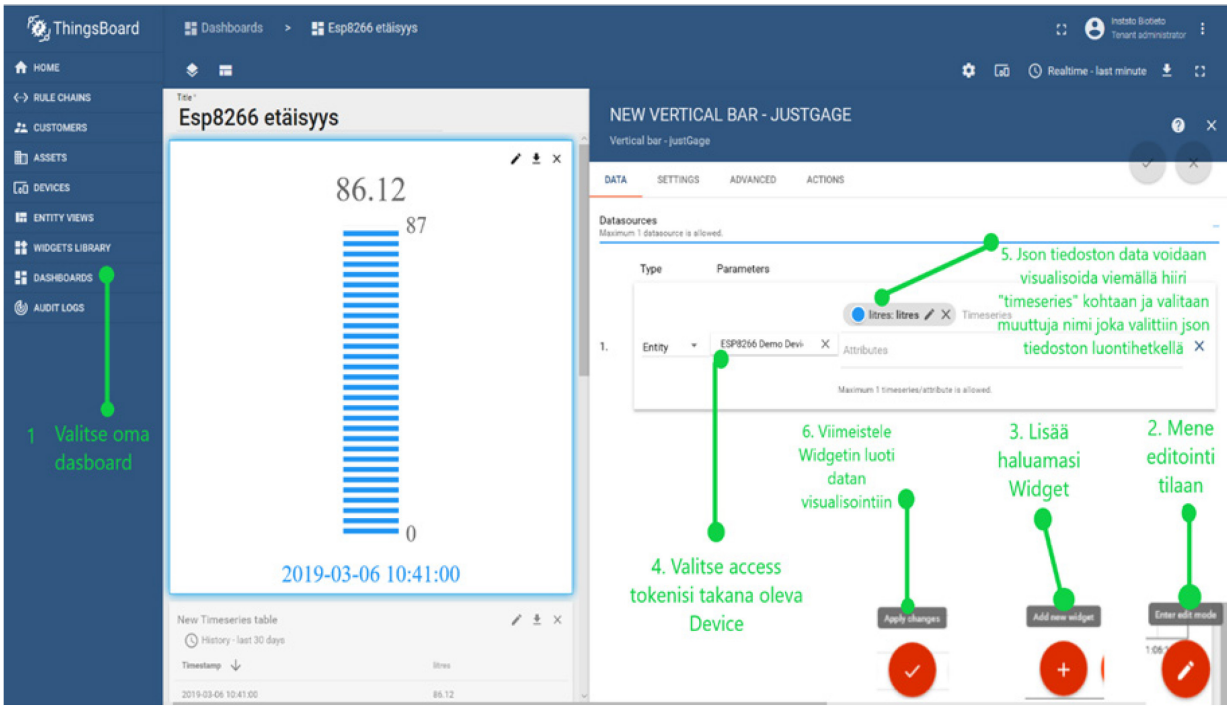
Datan visuaalinen ilme ympäristössä rakennettiin prototyypin mittadatasta, viivakaaviona, graafisesti pystypalkkina ja arvollisina riveinä aikaleimoineen (kuva 34).

Datan visuaalinen ympäristö demottiin (kuva 34) IoT-alustalle, ennen visualisaatiota mittadata oli verifioitu opinnäytetyön luvun 6.4 proseduurien mukaisesti.

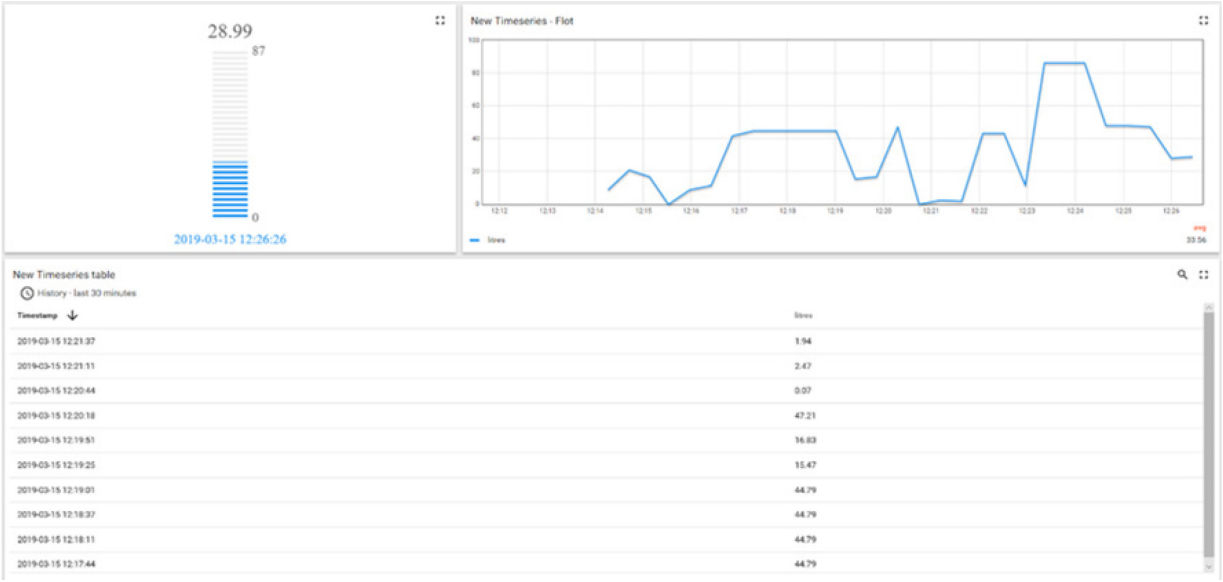
Datan arvollinen vaihtelu ilmenee yksittäisen mittauksen välittömänä visuaalisena tietona graafisessa pystypalkissa. Yksittäinen data ilmenee myös aikaleimallisena historiatietoina,

kuvassa 34 keskikohdasta alaspäin katsottuna, koko alareunan peittävässä taulussa. Datan visuaalinen ilme arvollisena vaihteluna ilmenee myös historiatietoina graafisessa viivakaaviossa, laskuina ja nousuina. (kuva 34)

Kuva 33. Ohje, ThingsBoard.



Kuva 34. Datanäkymä, ThingsBoard.



8 Johtopäätökset ja pohdinta

Tutkimuskysymyksen ratkaisu synnytti prototyypin kehitysversion, joka on lokakaivon tilavuuden seuraamiseen tarkoitettu pinnanmittauksen ohjausjärjestelmä, lisäksi ratkaisu toimii paikallisesta testiympäristöstä IoT-alustalle asti autonomisesti. Prototyyppi antoi testiympäristössä verifioidusti lukemat paikallisesti sarjaporttiin, sekä internetin ylitse IoT-alustalle. Ratkaisu käytiin läpi työvaiheineen, jotka olivat: tekninen toteutus, suunnittelu, valmistus ja mittatuloksien tarkistus. Mittauksien paikkansapitävyys tarkastettiin anturilta saatua digitaalista tietoa vasten verraten tiedettyyn faktatulokseen. Tiedetty tulos voitiin osoittaa faktaksi mittaamalla mittatikku fyysisellä mitalla ja merkitsemällä senttimetrit mitan osoittamalla tavalla mittatikun kylkeen. Mittatikku asetettiin pystyasentoon testiastian pohjasta ylöspäin ja testiastia täytettiin ennalta määrätyllä litramäärällä hiekkaa. Testiastia oli suora lieriö ja tilavuus tiedettiin, tällöin voitiin korkeuden mitta todeta myös tarkasti matemaattisesti kaavalla h = pii * r*2 / V. Johtopäätöksenä ratkaisusta voidaan todeta prototyypin ensimmäisen version olleen onnistunut ja palvelukonsepti idean mukainen ja vastaavan kustannustehokkuuteen teoriassa akun optimoinnillaan ja käytännössä edullisuudellaan per valmistettu prototyyppi, lisäksi mittatuloksen verifiointi on samalla todistus ohjausjärjestelmän potentiaalisuudesta ja kelpoisuudesta projektin jatkokehitykselle.

Opinnäytetyössä tehtiin myös toimiva visuaalinen demo IoT-alustan testiympäristössä. Lisäksi määriteltiin sanallisesti ja visuaalisesti testiympäristölle ja testiympäristössä tapahtuvan tiedonsiirron tekninen teoria, sekä testiympäristön alustamisen työvaiheet ja kehitysalustalla tapahtuva ohjelmointi liittyen testiympäristöön ja sen protokolliin. IoT-alustan työvaiheiden kuvauksilla varmistettiin testiympäristön toisinto, toimeksiantajan tarpeen niin vaatiessa.

Teoriassa prototyyppi mahdollistaa mittausten perusteella n. 506 päivän akunkeston, akun kapasiteetin ollessa 2450 mAh ja nimellisjännitteen ollessa tasavirrassa 3,6 volttia. Jatkokehittelyä pohdittaessa voitaisiin ajatella akun keston optimoinnissa vaikuttavia tekijöitä olevan, akun arkkitehtuurin sähkökemiallinen tyyppi, akkuratkaisun jännitetasot, sekä lopullisen tuotteen kenttäolosuhteet, muun muassa, kosteuden-, paineen- ja lämmön pitoisuus ilmassa ja koteloinnin sisätiloissa.

Jatkokehityksen tehtäviksi jää vielä aidoissa kenttäolosuhteissa akunkeston validointi ja siitä mahdollisesti seuraava optimointi. WiFi-verkon läpäisykyvyn testaaminen eri materiaaleilla, aidoissa kenttäolosuhteissa ja siitä mahdollinen optimointi lähetys- tai vastaanotinpään lähetystehoon. Kenttäolosuhteisiin soveltuvan koteloratkaisun valinta, tai vastavuoroisesti koteloiden kenttäkohtainen suunnitteleminen ja rakentaminen. Prototyypille IoT-pilvialustan suunnittelu ja tekeminen, tai valmiin IoT-alusta ratkaisun kilpailuttaminen ja hyödyntäminen prototyypille, Envor Group Oy:n tarpeiden mukaan. Yhtiön oman datahallintajärjestelmän tai kilpailutetun/rakennetun IoT-alustalle saadun tiedon integroiminen loka-autojen ajonhallintajärjestelmään.

Lisäksi tehtäväksi vielä jää valmiin prototyypin tuotteistaminen. Tuotteistamisella tässä yhteydessä tarkoitetaan, kehitysalustalla valmiiksi juotettujen, tälle työlle turhien komponenttien karsimista, eli ostopalveluna oman PCB:n tilaamista. PCB:n pitäisi sisältää vain tarpeelliset komponentit alustalle ja alustalta anturille. Lisäksi sillä tarkoitetaan tilatun PCB:n testausohjelmistoja ja varsinaisen alustan ohjelmistoa ja tarvittavaa logiikkaa.

Envor Group Oy voi hyödyntää opinnäytetyössä tehtyä prototyyppiä suoraan tulevissa vastaavan kaltaisissa projekteissa tai hyödyntää prototyyppiä palvelukonseptin jatkokehityksessä. Jatkokehityksen tulos voi johtaa tehokkaaseen loka-autojen reittioptimointiin, joka tällöin alentaisi yrityksen logistisia kustannuksia. Kustannusten alentuminen mahdollistaisi palvelukonseptin pohjalta uusia hinnoittelumalleja lokakaivojen tyhjennyksiin yrityksen asiakkaille. Syntyneistä malleista yrityksen asiakkaat hyötyisivät hinnan alennusta lokakaivon tyhjennyksestä, sillä tyhjennys tehtäisiin täysin automaattisesti perustuen mittadataan. Asiakkaiden ei olisi enää tarpeen tarkistaa fyysisesti lokakaivon täyttöastetta ja tilata puhelimitse tyhjennystä, näin syntyy myös yrityksen ja yrityksen asiakkaiden välinen uudistettu palvelusuhde, joka hyödyttää ajallisesti ja rahallisesti molempia osapuolia.

LÄHTEET

Ai-Thinker. (2018). *ESP-12F Datasheet.* Haettu 19.10.2019 osoitteesta https://docs.ai-thinker.com/_media/esp8266/docs/esp-12f_product_specification_en.pdf

Arduino.cc. (n.d.). math.h library. Haettu 2.12.2019 osoitteesta https://www.arduino.cc/en/math/h

DFROBOT. (n.d.). *Weather - proof Ultrasonic Sensor with Separate Probe SKU SEN0208.* Haettu 8.2.2020 osoitteesta https://www.dfrobot.com/wiki/index.php/Weather_-_proof_Ultrasonic_Sensor_with_Separate_Probe_SKU_:_SEN0208

Espressif. (2018). *ESP8266EX Datasheet.* Haettu 19.10.2019 osoitteesta https://lastminuteengineers.com/datasheets/ESP8266-datasheet.pdf

Eckel, T. (30.7.2016). *NewPing Library for Arduino.* Haettu 2.12.2019 osoitteesta https://playground.arduino.cc/Code/NewPing

Ellison, T. (2017). Lua Developer FAQ. Haettu 3.1.2021 osoitteesta https://nodemcu.readthedocs.io/en/release/lua-developer-faq/

Grokhotkov, I. (2017). *Welcome to ESP8266 Arduino Core's Documentation.* Haettu 3.2.2020 osoitteesta https://arduino-esp8266.readthedocs.io/en/latest/

Last Minute ENGINEERS. (n.d.). *Insight Into ESP8266 NodeMCU Features & Using It With Arduino IDE.* Haettu 5.6.2020 osoitteesta https://lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/

NODE MCU TEAM. (25.1.2015). *NodeMCU Devkit V1.* Haettu 2.12.2019 osoitteesta https://github.com/nodemcu/nodemcu-devkit-v1.0/blob/master/NODEMCU_DEVKIT_V1.0.PDF

O'Leary, N. (20.5.2020). *Arduino Client for MQTT* Haettu 4.1.2021 osoitteesta https://pubsubclient.knolleary.net/

SILICON LABS. (3.9.2020). *CP2102 Classic USB Bridges. CP210x Universal Windows Driver.* Haettu 4.1.2021 osoitteesta https://www.silabs.com/interface/usb-bridges/classic/device.cp2102

ThingsBoard. (2021a). *Architecture*. Haettu 5.1.2021 osoitteesta https://thingsboard.io/docs/reference/

ThingsBoard. (2021b). *MQTT Device API Reference*. Haettu 5.1.2021 osoitteesta https://thingsboard.io/docs/reference/mqtt-api/

Liite 1: C++ -ohjelmistoratkaisu pinnankorkeuden mittaamiseen

```
1. #include <math.h>
2. #include <PubSubClient.h>
3. #include <ESP8266WiFi.h>
4. #include <NewPing.h>
5.
6. #define WIFI_AP "HAMKvisitor"
7. #define WIFI_PASSWORD "hamkvisitor"
8.
9. #define TOKEN "o2jnsqUSyQo9w1UdL4Qg"            //thingsboard token
10.  #define SONAR_NUM 1                            // sensorien lukumäärä
11.
12.  const int MAX_DISTANCE = 56;                 //max distance mittausksessa
13.  const int nMeasures = 20;    //yhden mittauskerran mittauksien lukumäärä
14.
15.      //** SENSOR PINS
16.
17.      const int PingPin1 = 5            //  GPIO5,  D1
18.      const int EchoPin1 = 4;            //  GPIO4,  D2
19.      const int SensorPower1 = 14;      //  GPI014, D5, uä-sensorin syöttö
20.
21.
22.
23.      NewPing sonar[SONAR_NUM] = {                 // Sensor object array.
24.      NewPing(PingPin1, EchoPin1, MAX_DISTANCE)
25.  // Each sensor's trigger pin, echo pin, and max distance to ping.
26.
27.      };
28.
29.  char thingsboardServer[] = "demo.thingsboard.io";
30.
31.  WiFiClient wifiClient;
32.  PubSubClient client(wifiClient);
33.
34.  int status = WL_IDLE_STATUS;
35.  unsigned long lastSend;
36.
37.  void setup() {
38.    Serial.begin(115200);
39.    Serial.setTimeout(2000);
40.    InitWiFi();
41.    client.setServer( thingsboardServer, 1883 );
42.    lastSend = 0;
43.    // Wait for serial to initialize.
44.    if ( !client.connected() ) {
45.      reconnect();
46.    }
47.   while (!Serial) { }
48.    pinMode(SensorPower1, OUTPUT);
49.    digitalWrite(SensorPower1, HIGH);
50.    Serial.println("I'm awake.");
51.
52.
53.  //mittaukset anturilta
54.   float val[nMeasures];
55.   float filteredVal = 0;
56.    for(int i = 0; i < nMeasures; i++) {  //mittaa 20 krt
57.        delay(80);            //viive ennen mittausta
```

```
58.        val[i]= sensorRead();
59.    }
60.
61.  // järjestä arvot
62.     for(int i = 0; i < nMeasures - 1; i++) {
63.      for(int j = 0; j < nMeasures - i - 1; j++) {
64.      if (val[j] > val[j+1]) {
65.          int temp = val[j];
66.          val[j] = val[j+1];
67.          val[j+1] = temp;
68.        }
69.      }
70.     }
71.
72.  //järjestyksessä olevien tulosten tulostus sarjaporttiin
73.    for(int i=0; i<nMeasures;i++){
74.      Serial.print(val[i]);
75.     if (i < nMeasures - 1)
76.    Serial.print(", ");
77.    }
78.    Serial.println();
79.
80.  //keskiarvo ja persentiili
81.       for(int i = 3; i < nMeasures-3; i++) {     // laske yhteen arvot
   3...nMeasures-3
82.      filteredVal += val[i];
83.     }
84.     filteredVal = filteredVal/(nMeasures-6);   //laske keskiarvo
85.
86.    Serial.println(filteredVal);
87.
88.  // Prepare a JSON payload string
89.    String payload = "{";
90.    payload += "\"distance\":";
91.    payload += filteredVal;
92.    payload += "}";
93.
94.    // Send payload
95.    char attributes[100];
96.    payload.toCharArray( attributes, 100 );
97.    client.publish( "v1/devices/me/telemetry", attributes );
98.    Serial.println( attributes );
99.
100.
101.   Serial.println("Menen 20sekunnin unille, 1.9 sekunnin päästä");
102.   delay(1900);
103.   ESP.deepSleep(20e6); // 20e6 is 20 seconds
104.
105. }
106.
107. void loop(){
108.   //ei mitään tänne, ajetaan ohjelma vain yhden kerran
109. }
110.
111.
112. void InitWiFi()
113. {
114.   Serial.println("Connecting to AP ...");
115.   // attempt to connect to WiFi network
116.
117.   WiFi.begin(WIFI_AP, WIFI_PASSWORD);
118. while (WiFi.status() != WL_CONNECTED) {
119.     delay(500);
```

```
120.     Serial.print(".");
121.   }
122.   Serial.println("Connected to AP");
123. }
124.
125. void reconnect() {
126.   // Loop until we're reconnected
127.
128. while (!client.connected()) {
129.     status = WiFi.status();
130.   if ( status != WL_CONNECTED) {
131.       WiFi.begin(WIFI_AP, WIFI_PASSWORD);
132.     while (WiFi.status() != WL_CONNECTED) {
133.         delay(500);
134.         Serial.print(".");
135.       }
136.       Serial.println("Connected to AP");
137.     }
138.     Serial.print("Connecting to ThingsBoard node ...");
139.     // Attempt to connect (clientId, username, password)
140.   if ( client.connect("ESP8266 Device", TOKEN, NULL) ) {
141.       Serial.println( "[DONE]" );
142.     } else {
143.       Serial.print( "[FAILED] [ rc = " );
144.       Serial.print( client.state() );
145.       Serial.println( " : retrying in 5 seconds]" );
146.       // Wait 5 seconds before retrying
147.       delay( 5000 );
148.     }
149.   }
150. }
151.
152. float sensorRead()
153. {
154. //***********Readings Tank 1
155.    float Distance1, Litrat, Level, xR, keskiR;
156.    float pHalk = 39.5;
157.    float yHalk = 49.0;
158.    float pii = 3.14159;
159.
160.     Distance1 = sonar[0].ping_cm();   //saa distance tankki 1 yläpintaan
161.     Level = MAX_DISTANCE - Distance1;
162.     xR = (((yHalk-pHalk)/2)/MAX_DISTANCE * Level) + (pHalk/2);
163.        // ^^ // säde korkeudessa Level
164.     keskiR = (xR + (pHalk / 2))/2;
165.     Litrat = (pii * keskiR* keskiR * Level)/1000;
166.
167.
168.     Serial.println("distance: " + String(Distance1));   //printtaa depth
169.     Serial.println("litrat: " + String(Litrat));        //tulosta litrat
170.
171.     return Litrat;
172. }
```

Liite 2: C++ -ohjelmistoratkaisu litratilavuuden mittaamiseen

```
1. #include <math.h>
2. #include <PubSubClient.h>
3. #include <ESP8266Wifi.h>
4. #include <NewPing.h>
5.
6. #define WIFI_AP "HAMKvisitor"
7. #define WIFI_PASSWORD "hamkvisitor"
8.
9. #define TOKEN "o2jnsqUSyQo9w1UdL4Qg"            //thingsboard token
10.  #define SONAR_NUM 1                             // sensorien lukumäärä
   Change to suit your requirements.
11.
12.  const int MAX_DISTANCE = 56;                 //max distance mittausksessa
13.  const int nMeasures = 20;    //yhden mittauskerran mittauksien lukumäärä
14.
15.      //** SENSOR PINS
16.
17.      const int PingPin1 = 5;               //  GPIO5,  D1
18.      const int EchoPin1 = 4;             //  GPIO4,  D2
19.      const int SensorPower1 = 14;    //  GPI014, D5, uä-sensorin syöttö
20.
21.
22.
23.      NewPing sonar[SONAR_NUM] = {                // Sensor object array.
24.      NewPing(PingPin1, EchoPin1, MAX_DISTANCE)
25.  // ^^ // Each sensor's trigger pin, echo pin, and max distance to ping.
26.
27.      };
28.
29.  char thingsboardServer[] = "demo.thingsboard.io";
30.
31.  WifiClient wifiClient;
32.  PubSubClient client(wifiClient);
33.
34.  int status = WL_IDLE_STATUS;
35.  unsigned long lastSend;
36.
37.  void setup() {
38.    Serial.begin(115200);
39.    Serial.setTimeout(2000);
40.    InitWifi();
41.    client.setServer( thingsboardServer, 1883 );
42.    lastSend = 0;
43.    // Wait for serial to initialize.
44.      if ( !client.connected() ) {
45.      reconnect();
46.    }
47.    while(!Serial) { }
48.    pinMode(SensorPower1, OUTPUT);
49.    digitalWrite(SensorPower1, HIGH);
50.    Serial.println("I'm awake.");
51.
52.
53.  //mittaukset anturilta
54.    float val[nMeasures];
55.    float filteredVal = 0;
56.    for(int i = 0; i < nMeasures; i++) {  //mittaa 20 krt
57.        delay(80);            //viive ennen mittausta
58.        val[i]= sensorRead();
```

```
59.    }
60.
61.  // järjestä arvot
62.     for(int i = 0; i < nMeasures - 1; i++) {
63.      for(int j = 0; j < nMeasures - i - 1; j++) {
64.        if (val[j] > val[j+1]) {
65.          int temp = val[j];
66.          val[j] = val[j+1];
67.          val[j+1] = temp;
68.        }
69.      }
70.     }
71.
72.  //järjestyksessä olevien tulosten tulostus sarjaporttiin
73.    for(int i=0; i<nMeasures;i++){
74.      Serial.print(val[i]);
75.      if(i < nMeasures - 1)
76.    Serial.print(", ");
77.    }
78.    Serial.println();
79.
80.  //keskiarvo ja persentiili
81.       for(int i = 3; i < nMeasures-3; i++) {
82.  // ^^ // laske yhteen arvot 3...nMeasures-3
83.      filteredVal += val[i];
84.     }
85.     filteredVal = filteredVal/(nMeasures-6);    //laske keskiarvo
86.
87.    Serial.println(filteredVal);
88.
89.  // Prepare a JSON payload string
90.    String payload = "{";
91.    payload += "\"distance\":";
92.    payload += filteredVal;
93.    payload += "}";
94.
95.    // Send payload
96.    char attributes[100];
97.    payload.toCharArray( attributes, 100 );
98.    client.publish( "v1/devices/me/telemetry", attributes );
99.    Serial.println( attributes );
100.
101.
102.   Serial.println("Menen 20sekunnin unille, 1.9 sekunnin päästä");
103.   delay(1900);
104.   ESP.deepSleep(20e6); // 20e6 is 20 seconds
105.
106. }
107.
108. void loop(){
109.   //ei mitään tänne, ajetaan ohjelma vain yhden kerran
110. }
111.
112.
113. void InitWifi()
114. {
115.   Serial.println("Connecting to AP ...");
116.   // attempt to connect to Wifi network
117.
118.   Wifi.begin(WIFI_AP, WIFI_PASSWORD);
119.   while (Wifi.status() != WL_CONNECTED) {
120.     delay(500);
121.     Serial.print(".");
```

```
122.   }
123.   Serial.println("Connected to AP");
124. }
125.
126. void reconnect() {
127.   // Loop until we're reconnected
128.
129.   while (!client.connected()) {
130.     status = Wifi.status();
131.     if ( status != WL_CONNECTED) {
132.       Wifi.begin(WIFI_AP, WIFI_PASSWORD);
133.       while (Wifi.status() != WL_CONNECTED) {
134.         delay(500);
135.         Serial.print(".");
136.       }
137.       Serial.println("Connected to AP");
138.     }
139.     Serial.print("Connecting to ThingsBoard node ...");
140.     // Attempt to connect (clientId, username, password)
141.     if ( client.connect("ESP8266 Device", TOKEN, NULL) ) {
142.       Serial.println( "[DONE]" );
143.     } else {
144.       Serial.print( "[FAILED] [ rc = " );
145.       Serial.print( client.state() );
146.       Serial.println( " : retrying in 5 seconds]" );
147.       // Wait 5 seconds before retrying
148.       delay( 5000 );
149.     }
150.   }
151. }
152.
153. float sensorRead()
154. {
155. //***********Readings Tank 1
156.     float Distance1, Litrat, Level, xR, keskiR;
157.     float pHalk = 39.5;
158.     float yHalk = 49.0;
159.     float pii = 3.14159;
160.
161.     Distance1 = sonar[0].ping_cm(); //saa distance tankki 1 yläpintaan
162.     Level = MAX_DISTANCE - Distance1;
163.
164.     xR = (((yHalk-pHalk)/2)/MAX_DISTANCE * Level) + (pHalk/2);
165.      // ^^ // säde korkeudessa Level
166.
167.     keskiR = (xR + (pHalk / 2))/2;
168.     Litrat = (pii * keskiR* keskiR * Level)/1000;
169.
170.
171.     Serial.println("distance: " + String(Distance1));   //printtaa depth
172.     Serial.println("litrat: " + String(Litrat));        //tulosta litrat
173.
174.     return Litrat;
175. }
```