Antureiden datan lukeminen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

HAMK Riihimäki, Tieto- ja viestintätekniikan koulutus

syyslukukausi, 2018

Miikka Neuvonen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tieto- ja viestintätekniikan koulutus
HAMK Riihimäki

Tekijä Miikka Neuvonen **Vuosi** 2018

Työn nimi Antureiden datan lukeminen

Työn ohjaaja Timo Karppinen

TIIVISTELMÄ
Opinnäytetyössä käsitellään Digipaali-hankkeelle tehtyä prototyyppiä anturidatan saamisesta ja lähettämisestä tietokantaan. Antureiden datan saaminen tapahtui Raspberry Pi 3 b-laitteella ja ohjelman ohjelmointikielenä toimi Python. Antureiden arvojen lähettäminen tietokantaan tapahtui käyttäen GET-menetelmää WLAN-yhteyden avulla. Prototyypin antureina toimi PMOD Hygro, PMOD GPS, PMOD TMP2. Lopputuloksena oli onnistunut prototyyppi, joka toimi pohjana teollisuustason mallille.

Avainsanat Raspberry Pi, Anturi, Python,

Sivut25 sivua, joista liitteitä 5 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and communications Techonology
HAMK Riihimäki

Author Miikka Neuvonen **Year** 2018

Subject Reading sensor data

Supervisors Timo Karppinen

ABSTRACT

Thesis was all about making a prototype for a Digipaali- project. Aim was to make a prototype of sensors to a balemaker and send the data to a database using Wifi connection. Content contains description of chosen prototype sensors and the process of doing the prototype. Prototype has been made with microcontroller being Raspberry Pi 3 b and program has been written with Python programming language. Sending data to database was made by using GET-method. Conclusion was a succesful prototype which was used as a base for industrial version.

Keywords Raspberry Pi, Sensors, Python

Pages 25 pages including appendices 5 pages

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheena oli tehdä Digipaali nimiselle hankkeelle paalaimeen tarkoitettu prototyyppi, joka lukee erilaisten anturien dataa ja lähettää sen eteenpäin verkossa sijaitsevaan tietokantaan. Digipaali-hankkeen tarkoituksena oli identifioida maataloudessa käytettävät rehupaalit myyntitarkoitukseen. Prototyyppini tarkoituksena oli saada näihin paaleihin sen hetkistä tietoa, joka voisi asiakasta kiinnostaa, kuten ulkolämpötila, heinänlämpötila, ilmankosteudet paalaimen sisältä ja ulkoa ja GPS koordinaatit paalaus paikalta. Laitteiston toimintaympäristönä toimi maataloudessa käytettävä paalainlaite. Prototyyppiä varten toimintaympäristöstä oli huomioitavana se, että ei tarvinnut ottaa huomioon eri sääolosuhteita. Lisätehtävänä oli kuitenkin etsiä ja nimetä tähän käyttötarkoitukseen sopivat teollisuuskäyttöiset ja eri lika- ja sääolosuhteita kestävät anturit.

Prototyypin toteutus tapahtui lopulta käyttäen Raspberry Pi 3 b-laitetta ja anturien data luettiin I2C-protokollaa käyttäen. Antureilta saatu data lähettiin automaattisesti tietokantaan, kun paalaimelta paali läpäisi etäisyysanturin. Tietokantaan datan lähettäminen prototyypissä tapahtui käyttäen GET-menetelmää. Vaihtoehtoisena keinona olisi ollut käyttää POST-menetelmää. Opinnäytetyö ei sisällä tarkempaa tietoa tavasta millä paalit yksilöitiin ja siitä minkälainen tietokanta on kyseessä tai miten tietokanta luodaan ja hoidetaan. Opinnäytetyön prototyypissä tapa on korvattu yksinkertaisella IDn annolla, sillä identifiointi tapahtui hankkeen tilaajan pohjalta. Prototyyppiin, joka ei kestä eri sääolosuhteita antureiksi valittiin PMOD Hygro, PMOD TMP2, PMOD GPS ja E18-D80NK. Prototyypin lopulliseksi ohjelmointi kieleksi valitsin Python2. Python-ohjelma laitettiin käynnistymään samalla kuin Raspberry Pi 3 b käynnisty. Prototyypin lopullisena tarkoituksena oli todentaa, että Raspberry Pi:n avulla pystyttiin lähettämään tietokantaan WLAN-verkon välityksellä eri anturien mittaustuloksia ja toimia pohjana tuleville toteutuksille.

2 LAITTEISTO JA VERTAILU

Työn aloituksen ja kytkemisen tekeminen vaati ensimmäisenä osien valitsemisen. Osien valitsemista varten tehtiin laitevertailu. Ensimmäisenä valittiin pohja, johon anturit kytkettiin ensimmäisenä kiinni. Pohjana pystyi toimimaan mikrotietokone (Single board computer) tai kehityspohja (development boards). Näihin laitteisiin kuuluu esimerkkinä Raspberry Pi, Orange plus ja Arduino MKR. Työn tarkoitukseen nähden piti laitteissa olla sisäänrakennettu WLAN-mahdollisuus ja I2C yhteys mahdollisuus. Hankkeen puolelta ehdotettiin käyttöön Raspberry Pi-laitetta. Tämän vuoksi laitevertailu keskittyi enemmän siihen minkä Raspberry Pi-sarjan laite valittiin. Arduino-laitteista tehtiin silti kevyt vertailu toteutuksen mahdolliselle vaihtoehdoksi. Antureiden valinta tapahtui seuraavaksi ja kriteerinä oli I2C mahdollisuus. Lisänä oli järkevien antureiden valinta, millä tarkoitettiin antureita, jotka antoivat ulos arvoja, joista oli hankkeelle jotain hyötyä. Näihin kuului muun muassa paikkatieto (GPS), lämpötila ja ilmankosteus. Viimeisenä tehtiin vertailu ohjelmointikielen valinnasta. Ohjelmointikielen valinnassa ei ollut kriteerejä, joten vertailun jälkeen sai vapaasti valita ohjelmointikielen. Vaihtoehtoina oli C++, Node-RED, Node.JS ja Python. Aikaisempaa tietoperustaa oli pelkästään C++ kielestä, mutta sen perusteella ei valintaa tehty. Valinnan suurin vaikuttaja oli vertailu, jonka lopputuloksena päädyin tekemään työn Python-ohjelmointikielellä.

2.1 Vertailu

2.1.1 Laite

Laitteen vertailussa otettiin huomioon käyttötarkoitus. Laitteiden ominaisuudeksi odotettiin I2C-protokolla toimiva tiedon saaminen antureilta ja WLAN-verkkoyhteys. Vertailu kohteena toimi pääosina Arduino ja Raspberry Pi-laitteet.
Vertailukohteina olivat eri Raspberry Pi–laitteet ja Arduino sarjan laitteet. Hankkeen puolelta tuli pyyntö tehdä työ Raspberry Pi–laitteella. Tämän vuoksi Arduino-laitteiden vertailua ei työn alussa suoritettu. Vaihtoehto ratkaisujen tutkiessa selvisi, että esimerkiksi Arduino MKR sarjan laitteet olisivat toimineet toteutukseen paremmin kuin Raspberry Pi. Näihin parempiin MKR–laitteisiin kuului esimerkiksi Arduino MKR4000 ja MKR1010. Kummassakin näissä laitteissa toimi WLAN-yhteys ja löytyi pinnejä jotka toimi käyttäen I2C-protokollaa.

Työ päätettiin tehdä Raspberry Pi-laitteella. Päätöksen jälkeen tehtiin vertailu millä Raspberry Pi-laitteella työ tehdään. Vertailutaulukosta huomasi, että monessa ei ole

WLAN-ominaisuutta. Raspberry Pi 3 model b ja siitä uudemmat versiot sisältävät WLAN-ominaisuuden. Tämä rajasi laitteen valintaa, jonka vuoksi valittiin Raspberry Pi 3 b–laite. Tämä laite on tarpeeksi tehokas ja ominaisuuksiltaan hyvä, jonka vuoksi valittiin se työn pohjaksi. (Raspberry Pi Foundation. n.d)

Taulukko 1 Raspberry vaihtoehdot (Wikipedia. 2018)

Raspberry Pi	Model B +	2, model b	3, model b	Model zero
Siru	Broadcom BCM2835	Broadcom BCM2836	Broadcom BCM2837	Broadcom BCM2835
prosessori	ARMv6 single-core	ARMv7 single-core	ARMv8 single-core	ARMv6 single-core
Prosessorin nopeus	700MHz	900MHz	1.2GHz	1 GHz
Jännite ja teho	600mA 5v	800mA 5v	800mA 5v	160mA 5v
GPU	Dual-core VideoCore IV	Dual-core VideoCore IV	Dual-core VideoCore IV	Dual-core VideoCore IV
muisti	512MB SDRAM 400MHz	1GB SDRAM 400MHz	1GB SDRAM 400MHz	512MB SDRAM 400MHz
muistikortti	microSD	microSD	microSD	microSD
GPIO	40	40	40	40
USB2	4	4	4	microUSB
Ethernet	10/100 mb	10/100 mb	10/100mb	Ei
WLAN	Ei	Ei	Kyllä	Ei
bluetooth	Ei	Ei	Kyllä	Ei
audio	HD Audio, 3.5mm	HD Audio, 3.5mm	HD Audio, 3.5mm	Mini HDMI
HDMI	Kyllä	Kyllä	Kyllä	Mini
koko	85 x 56mm	85 x 56mm	85 x 56mm	65 x 30mm
Julkaisu vuosi	2014	2015	2016	2014

3.1.2 **Ohjelmointikieli**

Opinnäytetyön toteutuksessa oli vapaus ohjelmointikielen valintaan, joten tein nopean vertailun, millä ohjelmointikielellä prototyypin toteutuksen tein. Vertailu keskittyi enemmän C++ ja Python-ohjelmointikielen vertailuun.

C++ tarvitsee enemmän vaivaa kirjoittaa, mutta on parempi eristämään ohjelman eri osa alueita ja sillä saa hyödynnettyä laitteiston tehokkuutta hyvin. Yleisesti käytetty ohjelmointikieli, jolla on kirjoitettu suuri osa käyttöjärjestelmä-, ohjelmisto-, laiteohjain-, verkko- ja sulautettu järjestelmä ohjelmista, esimerkkinä moni Microsoftin ja Adobe Systemsin luomat ohjelmat.

Python on hyvä ohjelmointikieli yleisesti ja sitä pystytään käyttää melkein kaikkialla ja se toimii hyvin muiden ohjelmointikielien kanssa. Python on myös tehokas käyttämään välineistöä. Pythonin plussina on myös jatkokehitystä varten se, että sen lukeminen on suhteellisen helppoa. Esimerkiksi C++ kielen kirjoittajat pystyvät ymmärtämään Python ohjelmassa, mitä missäkin yleisesti tapahtuu ja mitä funktiot yleisesti tekevät. (Python foundation, 2018)

Ohjelmointikielivertailun jälkeen, päädyin siihen lopputulokseen, että teen joko C++ tai Pythonilla. Vertailun aikana selvisi, että Node.JS ja Node-RED ei ollut myöskään prototyypin tarkoitukseen nähden mitenkään huonoja vaihtoehtoja. Nämä ohjelmointikielet olisivat helpottaneet näkemään suurempaa kokonaisuutta ja tietokannan kanssa toimiminen olisi ollut suhteellisen helppoa. Node-RED:stä löytyy myös hyvät dokumentaatiot toimia Raspberry Pi-laitteiden kanssa. Päädyin lopulta toteuttamaan ohjelman Python-kielellä, vaikka C++ oli aikaisempaa kokemusta ja Node.JS ja Node-RED olivat myös toteutukselle hyviä vaihtoehtoja. Tämä valinta perustui siihen, että Raspberry Pi mahdollisti työn tekemisen Python ohjelmointikielellä ja lisäksi Raspberry Pi:llä on parempi tehdä Pythonia kuin C++. Tämä ei silti ole merkittävä tekijä sillä toteutusta varten ohjelmointikielillä ole merkittäviä eroja. Lopulliseen päätökseen miksi Python vaikutti myös se, että koska aikaisempaa kokemusta ei ohjelmointikielestä ollut. Sai hyvän syyn opetella uusi ohjelmointikieli. Vertailun tekemisestä myös selvisi se, että Pythonin opettelu ei ole vaikeata ja siitä löytyy hyvät dokumentaatiot. Päädyin siihen tulokseen, että hyvä tehdä tällä ohjelmointikielellä. (Nodered jsfoundation, n.d) (Nodejs foundation, n.d)

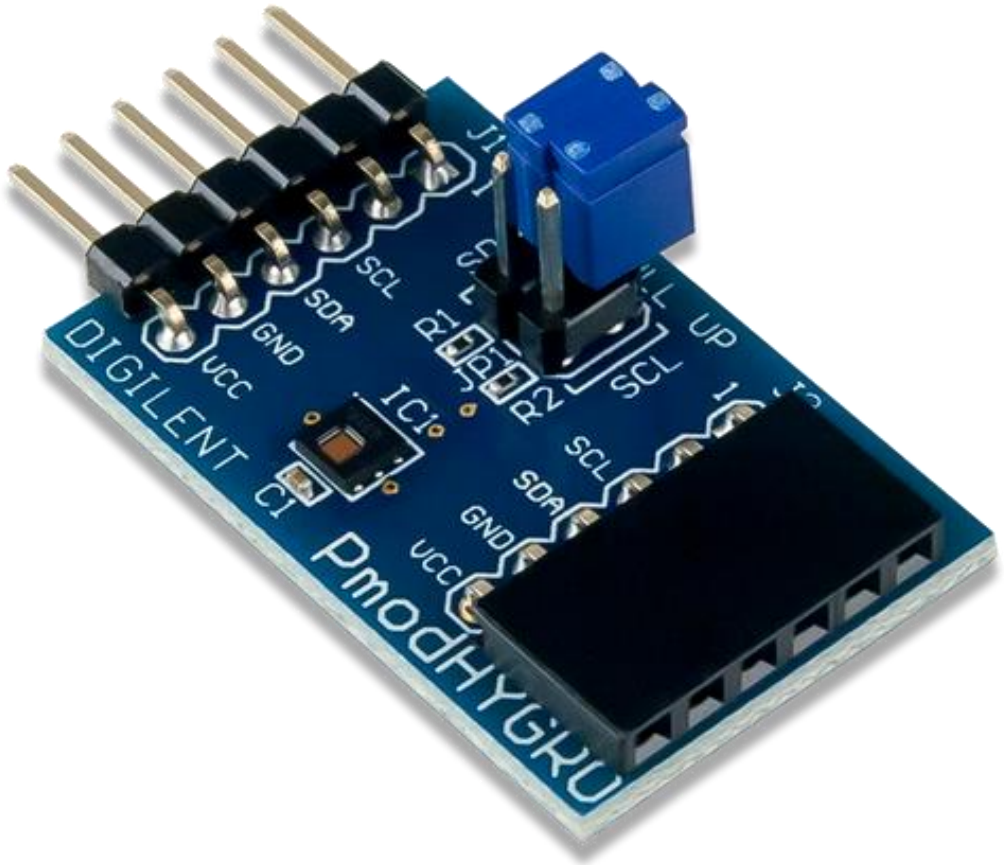
2.2 **Anturit**

Hanketta varten tehtiin prototyyppi, jonka toimintaan ei aluksi huomioitu sääolosuhteita. Prototyypin antureiden vertailussa oli monia eri antureita, mutta koska tarkoituksena oli vain testata antureita ja tulosten saamista ja lähettämistä. Valittiin hinnan puolesta halvat anturit, joissa oli I2C mahdollisuus ja sai järkeviä anturiarvoja kuten lämpötila, GPS ja ilmankosteus.

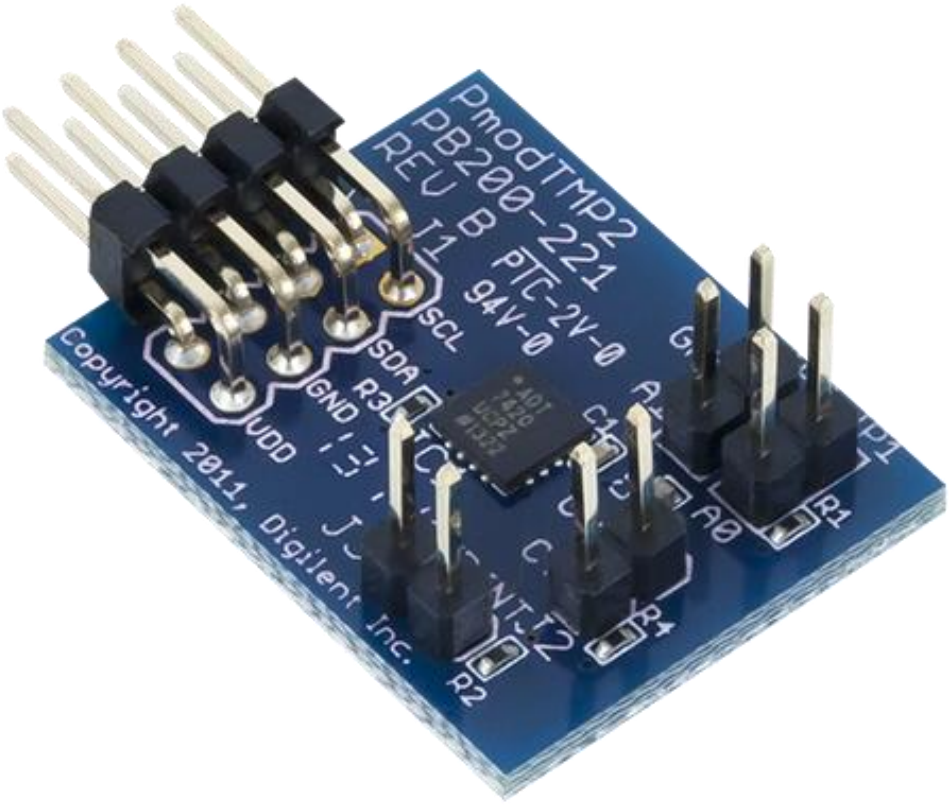
Tarkoituksena oli testata pystytäänkö saamaan antureilta tuloksia Raspberry Pi 3 b-laitetta käyttäen. Tämän vuoksi prototyyppiin valittiin anturit PMOD GPS (kuva 1), PMOD Hygro (kuva 2) PMOD TMP2 (kuva 3) ja infrapuna etäisyysanturia E18-D80NK (kuva 4). Säätilat huomioidessa PMOD laitteet eivät selviä vaikeissa sääolosuhteissa ja sen sijaan käytettäisiin esimerkkinä Telaire T9602 kosteus ja lämpötila sensoreita, jotka on rakennettu kestämään ympäristön vaativia olosuhteita. Lisäksi GPS tarvitsisi vaihtaa

sillä GPS moduuliin pitäisi saada lisättyä erillinen antenni. Kotelointi olisi tärkeä osa sen jälkeen, kun osille on rakennettu ”hattu”, hatulla tarkoitetaan piirilevyä, johon kaikki johdot, sekä vastukset tinattu kiinni, jolloin itse viritelmässä olisi mahdollisimman vähän johtoja, jolla pienettään virheiden ja rikki menemisen mahdollisuutta.

Kuva 1. PMOD GPS



Kuva 2. PMOD Hygro



Kuva 3. PMOD TMP2

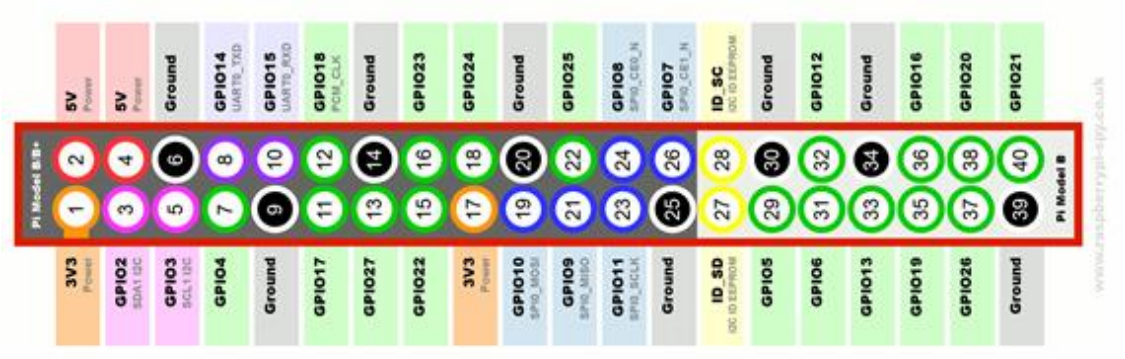
Kuva 4. E18-D80NK

2.3 **Raspberry Pi 3 b**

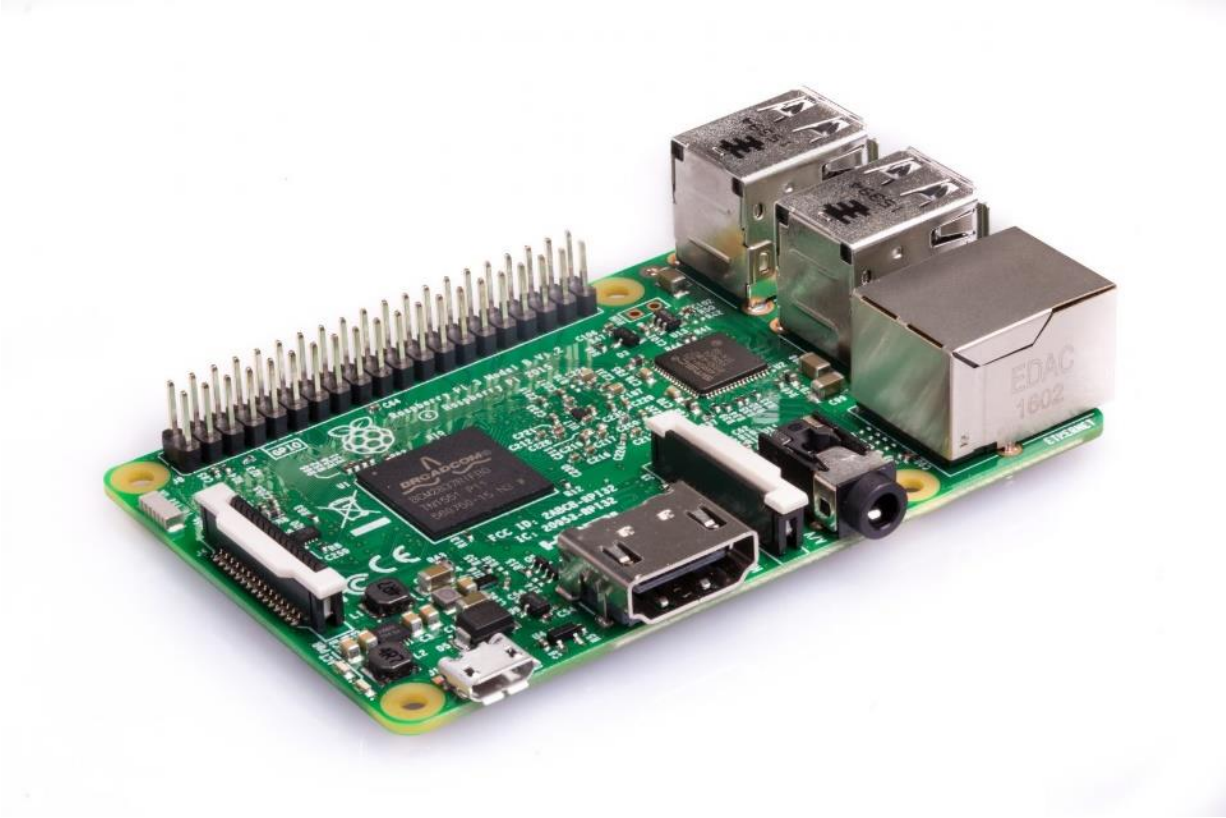
Työn pohjaksi valittiin Raspberry Pi 3 b-mikrotietokone. Vertailutaulukosta 1 näkee Raspberry Pi 3 b:n tekniset tiedot. Niistä Tärkeimpinä on WLAN-mahdollisuus ja GPIO-pinnien määrä. Kuvasta 5 näkee Raspberry Pi 3 b:n pinnien merkitykset. Huomioitavaa on se, että joidenkin anturien datan saa vain tietyllä yhteydellä. Esimerkkinä on se, että valituista antureista PMOD Hygro ja TMP2 vaativat SDA-yhteyden. Kuvasta näkee, että alkuperäisenä Raspberry Pi:ssä on vain 1 GPIO-pinni, johon on sisäänrakennettu SDA yhteys. Normaaleista GPIO-pinneistä on kumminkin mahdollisuus tehdä SDA yhteydellä toimivia pinnejä. Tämä tapahtuu erillisellä muuttaja osalla, joka muuttaa datan lukemismuodon. Työn toteutus ei tarvinnut mitään lisäosaa muuttamaan

pinnejä, vaan pärjättiin ihan oletus pinneillä. Työhön tarvittavat pinnit olivat 1-5 ja 8-10 (kuva 5. Pinnien merkitykset) (Raspberry Pi Foundation,n.d)

Raspberry Pi -laitteet saapuvat yleisesti mikroSD muistikortin kanssa, johon on esiladattuna käyttöjärjestelmä NOOBS. NOOBS tulee sanoista New Out Of the Box. NOOBS on Raspberry Pi:n oletus käyttöjärjestelmä, jonka avulla saadaan helposti ladattua eri käyttöjärjestelmiä. NOOBSille on valmiiksi asennettu Rasbian, mutta käyttäjä voi asentaa muita haluamiansa käyttö- järjestelmiä NOOBS:lta. Ladattaviin käyttöjärjestelmiin kuuluu ubuntu mate, Windows 10 IOT Core, OSMC, LIBREELEC, RISC OS ja PINET. Erilliset käyttöjärjestelmät NOOBS lataa omilta kehitys sivuiltaan, joka tarvitsee internetyhteyden. Käyttöjärjestelmällä ei ollut prototyypin toteutukseen suurta merkitystä, joten käytin valmiiksi asennettua Raspbian käyttöjärjestelmää. Rasbian käyttöjärjestelmälle on esiladattuna mm Python, Scratch, Sonic Pi, Java ja Mathematica ja sen mukana tulee myös yli 35 000 esikäännettyä ohjelmistoa. (Raspberry Pi Foundation,n.d)



Kuva 5. Raspberry Pi 3 b Pinnien merkitykset.



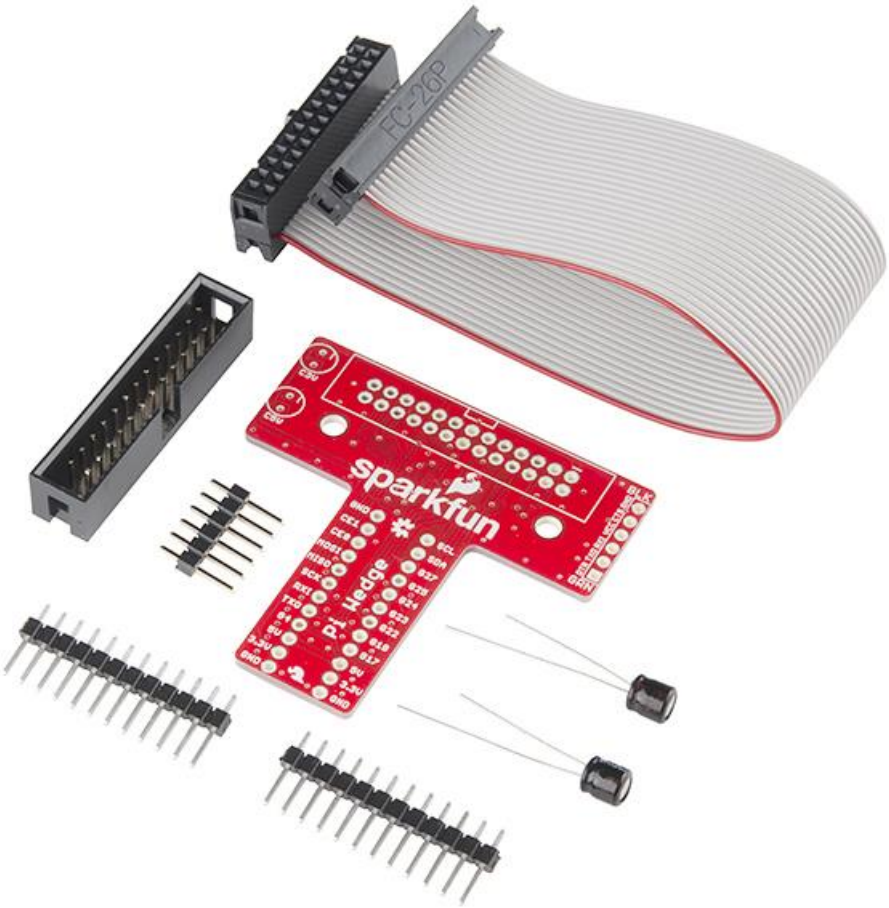
Kuva 6. Raspberry Pi 3 b

3 TOTEUTUS JA OHJELMA

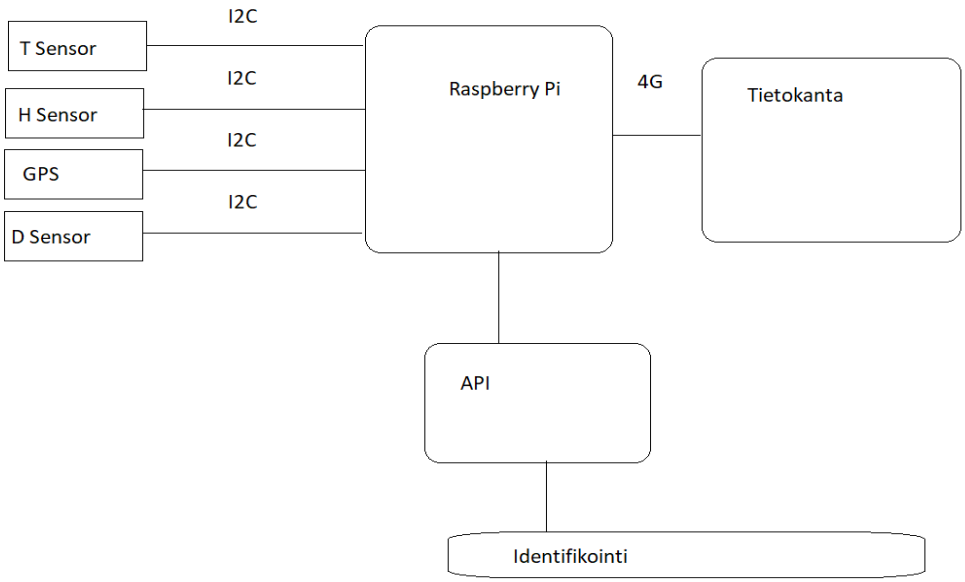
Opinnäytetyön toteutus aloitettiin looginen pala kerrallaan. Ensimmäisenä suunnittelu, suunnittelussa määriteltiin osa alue mihin opinnäytetyössä keskityttiin ja lisäksi aikataulu mihin mennessä prototyyppi tarvitsi olla valmiina. Suunnitelman aikana myös luotiin vertailu eri osista mitä testi prototyyppiin tarvittiin. Työn toteutuksesta myös luotiin kaavio. Suunnitelman kaaviona toimi kuva 8 kaavio. Kuvasta näkee minkä tyyppinen järjestelmä oli tarkoituksena tehdä. Yhteydet antureilta Raspberrylle piti tapahtua I2C protokollaa käyttäen.

Seuraavana loogisena palana osien tilaamisen jälkeen oli tehdä kytkentä koekytkentäalusta. Osien kytkennän aikana ilmeni pieniä ongelmia, jotka saatiin ratkaistua nopeasti. Yksi näistä ongelmista oli esimerkiksi Raspberry Pi 3 b -pinnien saaminen järkevästi koekytkentäalustaan. Tämä tuli nopeasti ratkaistua käyttäen Sparkfun Pi Wedge adapteria (kuva 7 Sparkfun Pi Wedge), jolla saatiin Raspberry Pi 3 b:n pinnit helposti käyttöön. Anturit kytkettiin anturi kerrallaan ja kytkennän jälkeen

luotiin testi ohjelma, jolla tarkistettiin miten ja saiko anturista oikeata dataa. Tämä tapahtui käyttäen Rasbian -käyttöjärjestelmän sisältämää Shell- ohjelmaa johon pystyttiin tulostamaan arvot mitä antureista saatiin ulos näkyviin. Tämä prosessi tehtiin jokaiselle anturille alkaen PMOD GPS:stä. Kun koekytkentälautaan oli saatu kytkettyä kaikki anturit ja niille oli luotu testi ohjelmat. Alettiin luomaan yhtenäistä ohjelma kokonaisuutta, missä kaikista antureista saatiin ulos oikeat arvot ja ohjelma hakisi arvot aina kun infrapuna etäisyysanturin edestä kulki esine. Ohjelmakoodin toimiessa lisättiin ohjelmaan viimeisenä tietokantaan lähettäminen. Tämä tapahtui GET-menetelmää käyttäen, jota testattiin ensimmäisenä lokaalisti omalle palvelimelle. Viimeisenä ohjelma laitettiin pyörimään käynnistykseen. Prototyypin kytkentä näkyy liitteestä 1 kytkentäkaavio



Kuva 7 Sparkfun Pi Wedge osina



Kuva 8 Kaavio

LIITE 2, Piirtokaavio

3.1 Ohjelma

Kaikkien sensoreiden ohjelma kirjoitettiin Python ohjelmointi kielellä. Työkaluna toimi Python 2 shell-ohjelma, mikä löytyy Raspberry Pi 3 b sisältämästä Rasbian -käyttöjärjestelmästä valmiina. Ensimmäisenä, että antureiden dataa sai GPIO ja Serial -pinneistä ulos piti antaa serial -pinnille READ/WRITE oikeudet. Tämä tapahtui helposti vain kirjoittamalla ROOT käyttäjänä.

(1)

CHMOD 777 /dev/ttyS0

Prototyyppi ohjelman tarkoituksena on juosta taustalla ja koko ajan tarkistaa milloin paalaimen etäisyysanturin edestä meni paali. Tähän tarkoitukseen ohjelma laitettiin toimimaan aloitusohjelmaksi Raspberry Pi -laitteeseen. Tämä tarkoittaa sitä, että kun Raspberry Pi käynnistettiin, käynnistyi tämä python ohjelma samalla ja alkoi saman tien tarkistamaan etäisyysanturia. Etäisyysanturi laitettiin sen vuoksi toimimaan loputtoman Loopin sisälle, joka tarkistaa Etäisyysanturilla onko esine läpäissyt sen. Prototyypissä vaihtoehtona oli tehdä myös laite, jossa etäisyysanturi oli korvattu napilla, jolloin käyttäjän täytyi manuaalisesti merkata, onko objekti valmis ja läpäissyt paikan, jossa liikesensori oli kytketty kiinni. Kummankin version ohjelmat tein, ja ainoana erona oli se, että sensoreiden dataa ei luettu ennen kuin nappi oli painettu.

Tämä ratkaisi ainoastaan sen ongelman, että prototyypin liikesensori oli todella herkkä, jolloin jos sensorin läpäisi jokin muu kuin haluttu objekti, käynnisti se ohjelmassa sensorien datan lähettämisen.

Ohjelma tarvitsee myös verkkoyhteyden. Tähän käytetään Wlan-yhteyttä. Koska ohjelma tulee toimimaan automaattisesti Raspberry Pi:n käynnistyksessä, tarvitsee Raspberry Pi:lle määritellä Wlan-verkko mihin otetaan yhteys automaattisesti. Tämä tapahtui menemällä ensin wpa_supplicant konfigurointi tiedostoon(1). Tiedoston sisälle lisättiin network, ja networkin sisälle SSID, joka on Wifin nimi ja psk mikä on salasana. Tähän työhön en tehnyt salattu Wifi salasanaa, mutta on mahdollista luoda encrypted PSK. Tämä voi olla 32 byte hexadecimal numero tai tehdä erillinen tekstitiedosto mistä laite ensin etsii Wifin salasanan, jonka jälkeen poistetaan vanha tekstitiedosto, ettei Raspberry Pi -laitteen sisälle jää kopioitavaa salasanaa. (Raspberry Pi Foundation,n.d)

(2)

sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

Tämän jälkeen pythonissa luotiin loop, jotta nähtiin jatkuvasti mitä informaatiota saadaan ulos anturi moduulista. Tämä luotiin helposti käyttämällä While-lauseketta.

(3)

While true:

Python ohjelman käynnistyminen samalla kuin Raspberry pi 3 b-käynnistyi, toteutui seuraavalla tavalla:

(4)

startup --> /etc/rc.local
yourpath/bin/FINAL_v2_DiSe.py&
yourscript.sh ---> #!/bin/sh
sleep 1
sudo python FINAL_v2_DiSe.py

Ohjelmassa myös käytettiin yleisiä kirjastoja.

(5)

import smbus.

SMBus (System Management Bus) on I2C-protokollan osajoukko. Tämä mahdollistaa laitteen (tässä tapauksessa raspberry pi 3 b) ajureiden käytön eri mukautujien välillä jotka käyttävät joko SMbus tai I2C. Koska toiminta tapahtuu

(6)

import serial.

Serial -kirjaston tarkoituksena oli mahdollistaa Serial -yhteys Raspberry Pi:n ja Serial -yhteyttä käyttävän anturin pinnien välillä.

(7)

```
import time.
```

Mahdollistaa eri aika käskyjen käytön ohjelmassa.

(8)

```
from decimal import *
```

Decimal-kirjastosta kaikki, auttamaan PMOD GPS:n arvojen muuttamista luettavaan muotoon varten.

3.1.1 PMOD GPS

PMOD GPS:n kytkeminen Raspberry Pi-laitteeseen tapahtui RXD ja TxD pinneillä. Nämä kontrolloivat anturin sisällä olevien kirjastojen bittejä Serial I/O:n perusteella. Tällä pystyi hakemaan tietyt bitit ja muuttamaan määrää kuinka paljon eri bittejä haettiin. Huomioitavana asiana oli se, että jos alettiin muuttamaan PMOD Gps:n rekisterin sisällä olevien rekistereiden bittien lukumäärää tai bittien lukumääriä. Täytyi anturin muutos funktiota mennä muuttamaan.

Ohjelmassa aluksi lisättiin tarvittavat kirjastot, joiden avulla pystyttiin lukemaan koodissa GPSn rekisteristä oikeat bitit ja bittimäärä. Tämän jälkeen python koodiin määriteltiin PMOD Gps:lle muuttuja, jonka tarkoituksena oli helpottaa ja nopeuttaa ohjelman kirjoittamista. Tässä tapauksessa määriteltiin tyyppi minkälainen yhteys Raspberry Pi:n ja PMOD Gps:n välillä oli. PMOD GPS toimi serial-yhteydellä, jonka vuoksi piti ohjelmaan määritellä portille, että se on Serial eikä esimerkiksi normaali GPIO tai SDA -yhteys. (Digilent inc, n.d)

(9)

```
port = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=1)
def find(str, ch):
for i, ltr in enumerate(str):
if ltr == ch:
yield i
```

Ohjelmaan luotiin tämän jälkeen muuttuja ck, jonka tarkoituksena oli antaa ohjelmalle aikaa etsiä GPS rekisteristä rekisteri GPRMC. Tämä tapahtui kymmenen kertaa, jos siinä

aikana ei ohjelma löytänyt GPRMC-rekisteriä, poistui ohjelma while-lausekkeesta ja nollasi muuttujan ck. While-lausekkeen sisällä taas määriteltiin muuttuja rcv, joka luki portin 10, mikä oli se pinni johon PMOD GPS oli kiinnitetty (kuva 5). Tämän jälkeen tehtiin muuttuja fd, jonka tarkoitus oli sisältää kaikki rekisterit mitä PMOD GPS:n sisältä löytyi.

(10)

```
ck=0
while ck <= 10:
rcv = port.read(10)
fd=fd+rcv
ck=ck+1
if '$GPRMC' in fd:
```

Kun anturin GPRMC-rekisteriin päästiin sisälle, poimittiin sieltä sisältä ensimmäisenä leveys koordinaatit ja tämän jälkeen pituus koordinaatit. Muuttujan dif tarkoituksena oli määritellä kuinka paljon informaatiota rekisteristä GPRMC kaivettiin esille ja tässä tapauksessa poimittiin sisältä 50 sanaa. Nämä kerättiin muuttujaan ‘p’, joka toimi data listana mistä poimittiin ja tehtiin muuttujat pituus ja leveys asteille.

(11)

```
ps=fd.find('$GPRMC')
dif=len(fd)-ps
if dif > 50:
data=fd[ps:(ps+50)]
p=list(find(data, ","))
lat=data[(p[2]+1):p[3]]
lon=data[(p[4]+1):p[5]]
```

Näiden muuttujien lat ja lon sisältävä data muutettiin luettavaan muotoon käyttäen kirjastoa Decimal hyväksi. Tämän avulla luotiin muuttujat s1 ja s2, jossa s1 on leveysasteet ja s2 on pituus asteet. Huomasin, että kun käyttää kirjastoa Decimal, tulee ongelmia, kun GPS ei saa yhteyttä satelliittiin. Seurauksena GPRMC-rekisteriin ei tule tarpeeksi dataa, jolloin s1=Decimal ei saa kirjastostaan muuttujaa ja ohjelma lakkasi toimista.

(12)

```
s1=lat[2:len(lat)]
s1=Decimal(s1)
s1=s1/60
s11=int(lat[0:2])
s1=s11+s1
```

```
s2=lon[3:len(lon)]
s2=Decimal(s2)
s2=s2/60
s22=int(lon[0:3])
s2=s22+s2
```

3.1.2 PMOD Hygro

PMOD Hygroa varten käytettiin kirjastoja smbus, time ja RPi.GPIO as GPIO. Anturissa PMOD Hygro on sisällä monta eri rekisteriä. Lämpötilan ja ilmankosteuden saantia varten käytettiin kumminkin vain kahta niistä rekistereistä. Toinen sisältää lämpötila ja toinen ilmankosteuden datan. Rekistereiden sisällä oleva data on kumminkin bitteinä, joten nekin täytyy lopulta muuttaa numeraalisiksi arvoiksi. Muuttuja addr1 on laitteen oma ID-rekisteri, hexa muodossa. (Digilent inc, n.d)

(13)

```
import smbus
import time
import RPi.GPIO as GPIO

bus = smbus.SMBus(1)
addr1 = 0x40
```

Kun ohjelmaan oli lisätty tarvittavat kirjastot, ja tehty tarvittavat muuttujat helpottamaan ohjelman kirjoittamista, alettiin käsitellä PMOD Hygro:n sisäisiä rekistereitä. Ensimmäisenä mentiin sisälle PMOD Hygro:on ja kirjoitettiin sen sisälle ensimmäinen bitti nollaksi. Tämän tarkoituksena oli se, että rekisterin sisällä oleva vanha data ei vaikuttaisi ja häiritsisi uusia mittausarvoja. Seuraavaksi hyväksi käytettiin kirjastoa smbus, jonka avulla luettiin anturin kummatkin rekisterit, lämpötila ja ilmankosteus. Käyttäen smbus-kirjastoa hyväksi saatiin rekistereistä poimittua ne bitit, jotka sisälsivät lämpötila ja ilmankosteuden. (Digilent inc, n.d)

(14)

```
bus.write_byte(addr1, 0x00)
time.sleep(0.2)

a=bus.read_byte(addr1)
b=bus.read_byte(addr1)
```

Luettu bitti data tämän jälkeen piti muuttaa luettavaan muotoon, tässä tapauksessa celsiuksiksi. Käännös tapahtui PMOD Hygro:n datasheetistä löytyvällä

käännöslausekkeella. kTemp on lämpötila celsiuksena pyöristettynä yhteen desimaaliin ja hTemp on ilmankosteus prosentteina pyöristettynä yhteen desimaaliin.

(15)

```
hTemp = ( a* 256 + b) / 65536.0 * 100.0
hTemp = hTemp - hTemp % 0.1
kTemp = (a*256+b)*165/65536-40
```

3.1.3 PMOD TMP2

Aluksi kirjoitetaan rekisterin ensimmäinen bitti nollaksi, eli nollataan anturilta tuleva arvo, Tämän jälkeen otetaan lämpötilarekisteristä arvo pihalle ja muutetaan se muuttujaksi temp. 0x4b on laitteen ID, 0x01 on laitteen lämpötila rekisteri ja 0x60 on bittien kirjoitus määritelmä.

PMOD TMP2 käyttää samoja kirjastoja kuin PMOD Hygro. Erona on mikä on laitteen rekisterin ID ja lauseke millä muutetaan rekisteristä tuleva bitti data ymmärrettävään muotoon, joka on tässä tapauksessa celsius asteina. (Digilent inc, n.d)

(16)

```
import smbus
import time
bus = smbus.SMBus(1)
bus.write_byte_data(0x4b,0x01,0x60)
time.sleep(0.5)
data = bus.read_i2c_block_data(0x4b,0x00,2)
cTemp = (temp>>3)*0.0625
cTemp = cTemp - cTemp % 0.1
```

3.1.4 E18-D80NK

Anturin kytkentä tapahtui piirikaavion mukaisesti. Anturissa tarvitsi vain lukea arvoja mitä sieltä tuli ulos. Tämä oli helppoa sillä, kun sensorin eteen tuli esine, muuttui arvo 1 arvoksi 0. Ohjelmassa tarvitsi vain laittaa se, ”False” joka vastasi sitä, että anturin

arvo muuttui 0:ksi. Lisäksi käytettiin time-kirjastoa hyväksi, että saatiin anturin herkkyyteen pieni viive.

E18-D80NK-infrapuna etäisyysanturin kytkentä tapahtui piirikaavion mukaisesti. Anturin toimi ideaalisesti prototyyppiin, sillä anturista löytyi manuaali etäisyyden säätö ja anturista tulee ulos pelkästään kahta eri arvoa. Kun anturin infrapunan edessä ei ollut esinettä tai mitään muuta tavaraa tuli anturista ulos arvo 1. Kun eteen tuli esine muuttui arvo 0:ksi. Huomioitavaa oli se, että arvo 0 vastasi koodissa ”False”, tämä johtui anturin kytkennästä ja miten anturin sisäisen virta muuttui, eli kun anturin eteen tuli esine muuttui anturin sisäinen kytkentä ”Falseksi” ja kun edessä ei ollut mitään oli anturin arvo ”True”. Python ohjelmaan piti aluksi määritellä se minkälainen pinni tyyppiin anturi oli kytketty, tässä tapauksessa GPIO.BOARD.

Anturi oli kiinni pinnissä numero 37, ja määriteltiin pinni IN. Ohjelmassa oli kaksi vaihtoehtoa, miten saada anturin herkkyyttä muokattua. Herkkyyttä ei itsessään saatu muutettua, mutta pythonin avulla pystyi tekemään kiertoreitin. Tähän loin kaksi erillistä if-lauseketta joista toisen tarkoituksena oli tarkistaa onko infrapuna-anturin edessä esine, joka lisäsi muuttujaan false arvon 1. Tarkoituksena oli, että kun anturi näyttää 10 kertaa ”False” tarkoittaisi se sitä, että sen edessä olisi paali. Tämän jälkeen se menisi seuraavaan IF-lausekkeeseen sisälle ja muuttaisi muuttujan ID-arvoa yhdellä, jolloin annettaisiin paalille oma ID.

(17)

```
false = 0
id = 0
GPIO.setmode(GPIO.BOARD)
GPIO.setup(37, GPIO.IN)
cd = GPIO.input(37)
if (cd == False):
false = false + 1
if ( False == 10);
id = id + 1
if ( cd == True);
false = 0
```

4 TIETOKANTAAN LÄHETTÄMINEN

Tapahtuma tapahtui GET-menetelmällä. Vaihtoehtoisesti myös pystyi lähettämään POST-menetelmällä. POST-menetelmä nimensä mukaisesti lähettää suoraan tietokantaan arvot, jotka on määritelty mukaan. GET-menetelmässä taas välikätenä

käytetään PAYLOAD-pyyntöä, jonka avulla tiedot lähetetään eteenpäin tietokantaan. Datan lähettäminen testattiin aluksi asentamalla Raspberry Pi:n sisälle oma NodeJS -palvelin, (localHost:3000) johon lähettiin dataa aluksi, tämä vain korvattiin tietokannan osoitteella. GET-menetelmällä käytetään PAYLOAD pyyntöä, jonka sisälle laitetaan ensimmäisenä ID ja tämän jälkeen arvo. Tässä tapauksessa käytettiin muuttujia, mitkä määrättiin eri antureille kuten PMOD Hygro:n muuttujat hTemp ilmankosteus ja kTemp lämpötila. Datan lähettämisessä on myös huomioitavana tietoturvallisuus. Tietokannan puolelta luodaan laitteille ja ohjelmalle salatut turvallisuus avaimet. Pidemmälle kaavalle pitäisi jokaiselle Raspberry Pi-laitteelle määritellä omat oikeat ID ja turvallisuus avaimet, jotta pystyttäisiin valvomaan laitteistoa ja ettei tapahtuisi väärinkäyttöä lähetyille mittausarvoille, ja lisäksi ei pystyttäisi väärentämään tietoja tietokantaan. Testattavana oli myös, se mitä tapahtui, kun Raspberry Pi:llä ei ollut verkkoyhteyttä ja myös sitä, miten ohjelma reagoi väliaikaiseen verkon menetykseen. Näissä tapauksissa ohjelma ei mennyt rikki, mutta lähetty PAYLOAD ei ikinä saapunut perille tietokantaan. Ohjelma lähetti PAYLOAD request-pyynnön palvelimelle, mutta koska sillä ei ollut verkkoyhteyttä katosi tämä palvelupyyntö ja PAYLOAD katosi. Verkkoyhteyden palatessa ohjelma jatkoi normaalia toimintaa ja lähetti PAYLOADin ja PAYLOAD saapui palvelimelle. (Geeksforgeeks inc, n.d)

(18)

```
payload = {‘id:id,’'Lat':s1,'Lng':s2, 'Humidity': hTemp, 'Temperature1': kTemp,
'Temperature2': cTemp}

r = requests.get(‘LocalHost:3000', params=payload)
```

Lisäksi tein yhden vikakoodi napin, joka lähettää vain tiedon, että äsken lähetyssä paalissa tapahtunut jokin virhe tai siinä on jotakin huomioitavaa. Tämä tapahtui else-lausekkeella. Ohjelmassa taas määriteltiin minkä tyyppinen pinni on kyseessä ja laitetaan pinni päälle muotoon IN.

(19)

```
else:
GPIO.setmode(GPIO.BOARD)
GPIO.setup(13, GPIO.IN)
ErrorButton1 = GPIO.input(13)
time.sleep(0.5)
if ErrorButton1
vikaviesti = "ROSKAA PAALAIMESSA"
payload = {'ErrorMSG': vikaviesti}
r = requests.get(localhost', params=payload)
```

5 YHTEENVETO

Opinnäytetyölle alussa asetetut vaatimusmäärittelyt olivat, että valmiin prototyypin on saatava eri antureiden arvoja I2C-protokollalla, pystyä lähettämään se WLAN-yhteydellä tietokantaan ja toimia automaattisesti. Lisätavoitteina oli tehdä prototyypistä piirilevy "hattu", mahdollinen versio teollisuustason antureilla ja prototyypin kiinnittäminen oikeaan paalaimeen. Lisätavoitteet eivät toteutuneet kokonaisuudessaan, lisätavoitteista vain täyttyi osa mitä haluttiin, kuten teollisuustason antureiden löytyminen, pohjapiirustus, jonka perusteella piirilevy pystytään rakentamaan. Vaatimusmäärittelyt toteutuivat prototyypin osalta, vaikka prototyyppiä ei kiinnitetty paalaimeen. Opinnäytetyöni lähettää eri antureiden arvoja tietokantaan Raspberry Pi 3 b–laitteen avulla. Tietokantaan lähettäminen tapahtui Python ohjelmointikielellä ja tapahtui automaattisesti käyttäen E18-D80NK etäisyysanturia. Prototyypin antureihin kuuluu PMOD Hygro, PMOD GPS ja PMOD TMP2.

Valitut anturit ja Raspberry Pi 3 b olivat riittävät alkuperäiseen tarkoitukseen, mikä oli antureiden datan saamista ja lähettämistä tietokantaan. Antureiden arvojen saanti tapahtui Python-ohjelmalla ja lähettäminen tapahtui GET-menetelmällä.

Prototyypin jatkokehittäminen ei ole vaikeata, kun korvataan nykyiset anturit teollisuustason antureilla. Myös erilaisten anturien yhdistäminen olisi mahdollista pohjaan, eikä niille ohjelman tekeminen olisi vaikeata. Prototyypin toteutukselle myös löytyy paljon eri vaihtoehtoja. Raspberry Pi 3 b–laitteen pystyy korvaamaan monella eri laitteella esimerkiksi Arduino MKR-sarjan laitteilla. Monelle ratkaisulle löytyy myös vaihtoehtoisia menetelmiä, kuten tiedon lähettäminen POST- eikä GET-menetelmällä. Lisäksi myös ohjelmointikielen pystyy tässä vaiheessa helposti vaihtamaan esimerkiksi C++ kieleksi, joka olisi parempi vaihtoehto Arduino–laite toteutukselle. Lisäksi tietoturvallisuuteen liittyviä asioita pitäisi parannella ja tehdä, ettei laitteen ohjelmaan ja tietokantaan liittyviin asioihin ulkopuolinen pääsisi käsiksi.

LÄHTEET

3dmodularsystems (n.d) E18-D80NK. Haettu 11.12.2018 osoitteesta http://3dmodularsystems.com/en/electronic/130-e18-d80nk-3-80cm-optical-sensor.html

Mouser electronic, (n.d) GPS-14414. Haettu 11.12.2018 osoitteesta https://www.mouser.fi/ProductDetail/SparkFun/GPS-14414?qs=/ha2pyFaduilzglDv8EENsGgI3iWsrxps0la2EvS2BYD7MP4QbpQwg%3d%3d

Nodejs foundation (n.d) NodeJS. Haettu 11.12.2018 osoitteesta https://nodejs.org/en/about/

Nodered jsfoundation (n.d) Node-RED. Haettu 11.12.2018 osoitteesta https://nodered.org/

Digilent inc (n.d) PMOD GPS. Haettu 11.12.2018 osoitteesta https://reference.digilentinc.com/reference/pmod/pmodgps/start

Digilent inc (n.d) PMOD Hygro. Haettu 11.12.2018 osoitteesta https://store.digilentinc.com/pmod-hygro-digital-humidity-and-temperature-sensor/

Digilent inc (n.d) PMOD TMP2. Haettu 11.12.2018 osoitteesta https://reference.digilentinc.com/reference/pmod/pmodtmp2/start

Geeksforgeeks inc (n.d) POST ja GET menetelmä. Haettu 11.12.2018 osoitteesta https://www.geeksforgeeks.org/get-post-requests-using-python/

Python foundation (2018) Python. Haettu 11.12.2018 osoitteesta https://docs.python.org/3/faq/general.html

Raspberry Pi Foundation (n.d) Raspberry Pi Käyttöjärjestelmät. Haettu 11.12.2018 osoitteesta https://www.raspberrypi.org/downloads/

Raspberry Pi Foundation (n.d) Raspberry Pi NOOBS käyttöjärjestelmä. Haettu 11.12.2018 osoitteesta https://www.raspberrypi.org/blog/introducing-noobs/

Raspberry Pi Foundation (n.d) Raspberry Pi 3 b. Haettu 11.12.2018 osoitteesta

https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Raspberry Pi Foundation (n.d) Raspberry pi WLAN konfikurointi. Haettu 11.12.2018 osoitteesta
https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md

wiki.erazor (n.d) SMBUS Python I2C. Haettu 11.12.2018 osoitteesta
http://wiki.erazor-zone.de/wiki:linux:python:smbus:doc

IBEX (n.d) SMBUS Raspberry Pi Python I2C. Haettu 11.12.2018 osoitteesta
http://www.raspberry-projects.com/pi/programming-in-python/i2c-programming-in-python/using-the-i2c-interface-2

SparkFun Electronics (n.d) Sparkfun Pi Wedge tutorial. Haettu 11.12.2018 osoitteesta
https://learn.sparkfun.com/tutorials/preassembled-40-pin-pi-wedge-hookup-guide/all

Mouser Electronics (2018) Telaire T9602. Haettu 11.12.2018 osoitteesta
https://www.mouser.fi/datasheet/2/18/AAS-920-638H-Telaire-T9602-041318-web-1315842.pdf

Wikipedia (2018) Vertailu Raspberry laitteet. Haettu 11.12.2018 osoitteesta
https://en.wikipedia.org/wiki/Raspberry_Pi

Liite 1

MÄÄRITTELYT

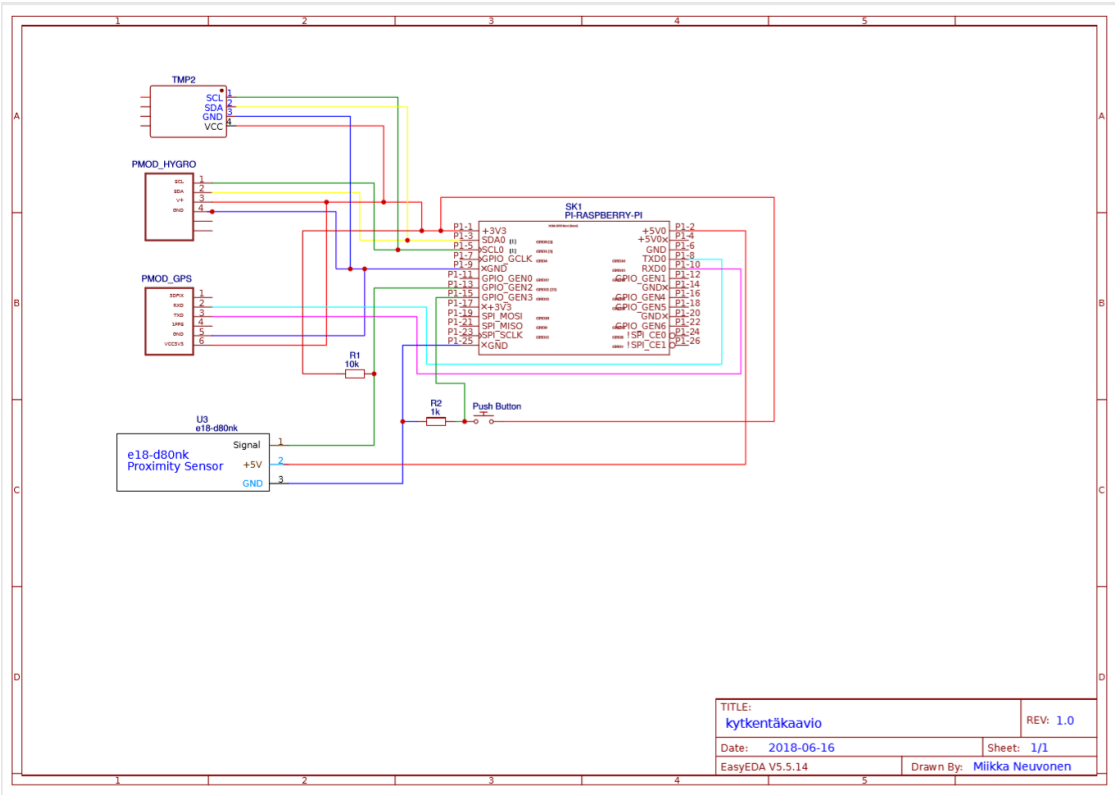
Lyhenteet:

I2C Inter-Integrated Circuit

GPIO	General Purpose I/O
DPIO	Data Purpose I/O
SDA	Serial Data Line
SCL	Serial Clock Line
SPI	Serial Peripheral Interface Bus
ROOT	Järjestelmänvalvoja / admin oikeudet
SD	Secure Digital, digitaalinen turvayhteys
WLAN	wireless local area network, langaton verkkoyhteys
GPRMC	APRS GPS Mobile RMC, Yleinen paikkatieto, koordinaatit
GPS	Global Positioning System, Maailmanlaajuinen paikallistamisjärjestelmä

Liite 2

PIIRTOKAAVIO



OHJELMAKOODI

Liite 3

```
import smbus                                                    (16)
import time
import serial
import RPi.GPIO as GPIO
import os, time
import requests
from decimal import *


GPIO.setmode(GPIO.BOARD)
GPIO.setup(37, GPIO.IN)
def find(str, ch):
   for i, ltr in enumerate(str):
      if ltr == ch:
         yield i

port = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=1)
bus = smbus.SMBus(1)
addr1 = 0x40
cd=1
id=0
false=0
while True:
                ck=0
                cd = GPIO.input(37)
                if (cd == False):
                                false = false + 1
                if ( False == 10);
                                id = id + 1
                                 fd=''
                                 ck=0
                                while ck <= 10:
                                                rcv = port.read(10)
                                                fd=fd+rcv
                                                ck=ck+1
                                                if '$GPRMC' in fd:
                                                                ps=fd.find('$GPRMC')
                                                                dif=len(fd)-ps
                                                                if dif > 50:
                                                                data=fd[ps:(ps+50)]
```

```
                                        p=list(find(data, ","))
                                        lat=data[(p[2]+1):p[3]]
                                        lon=data[(p[4]+1):p[5]]

                                        s1=lat[2:len(lat)]
                                        s1=Decimal(s1)
                                        s1=s1/60
                                        s11=int(lat[0:2])
                                        s1=s11+s1

                                        s2=lon[3:len(lon)]
                                        s2=Decimal(s2)
                                        s2=s2/60
                                        s22=int(lon[0:3])
                                        s2=s22+s2

                                bus.write_byte(addr1,0x00)
                                time.sleep(0.2)
                                bus.write_byte(addr1, 0x00)
                                time.sleep(0.2)

                                bus.write_byte_data(0x4b,0x01,0x60)
                                time.sleep(0.2)
                                a=bus.read_byte(addr1)
                                b=bus.read_byte(addr1)
                                data = bus.read_i2c_block_data(0x4b,0x00,2)
                                temp = (data [0] << 8 | data [1] )
                                cTemp = (temp>>3)*0.0625
                                cTemp = cTemp - cTemp % 1
                                hTemp = ( a* 256 + b) / 65536.0 * 100.0
                                hTemp = hTemp - hTemp % 1
                                kTemp = (a*256+b)*165/65536-40

                                payload = {'id':id,'Lat':s1,'Lng':s2, 'Humidity':
hTemp, 'Temperature1': kTemp, 'Temperature2': cTemp}
                                r = requests.get('localhost:3000',
params=payload)

                                ck=11
                                time.sleep(5)
                        if ( cd == True);
```

```
                    false = 0

            else:
                    GPIO.setmode(GPIO.BOARD)
                    GPIO.setup(13, GPIO.IN)
                    ErrorButton1 = GPIO.input(13)
                    time.sleep(0.5)
                    if ErrorButton1:

                            vikaviesti = "ROSKAA PAALAIMESSA"
                            payload = {'ErrorMSG': vikaviesti}
                            r = requests.get('localhost:3000',
params=payload)
```