Tekninen loppuraportti, nro 1

**RASPBERRY PI - IOT PROJEKTI**

**Juho Syrjänen**

**Jan Nygrén**

**Joakim Tick**

**Heidi Laaksonen**

# Git-repositorio

Projektin julkaisua varten luotiin avoin Git-repositorio, joka sisältää kaikki projektia varten kirjoitetut koodit/skriptit ja muun dokumentaation.

http://github.com/yukusu/IoT-Rasp

Kaikki projektin dokumentit, koodit/skriptit on linsensoitu **GNU General Public License v. 2.0** -lisenssillä.

# Kokoonpano

## 2.1 Raspberry Pi

2.1.1 Asennus

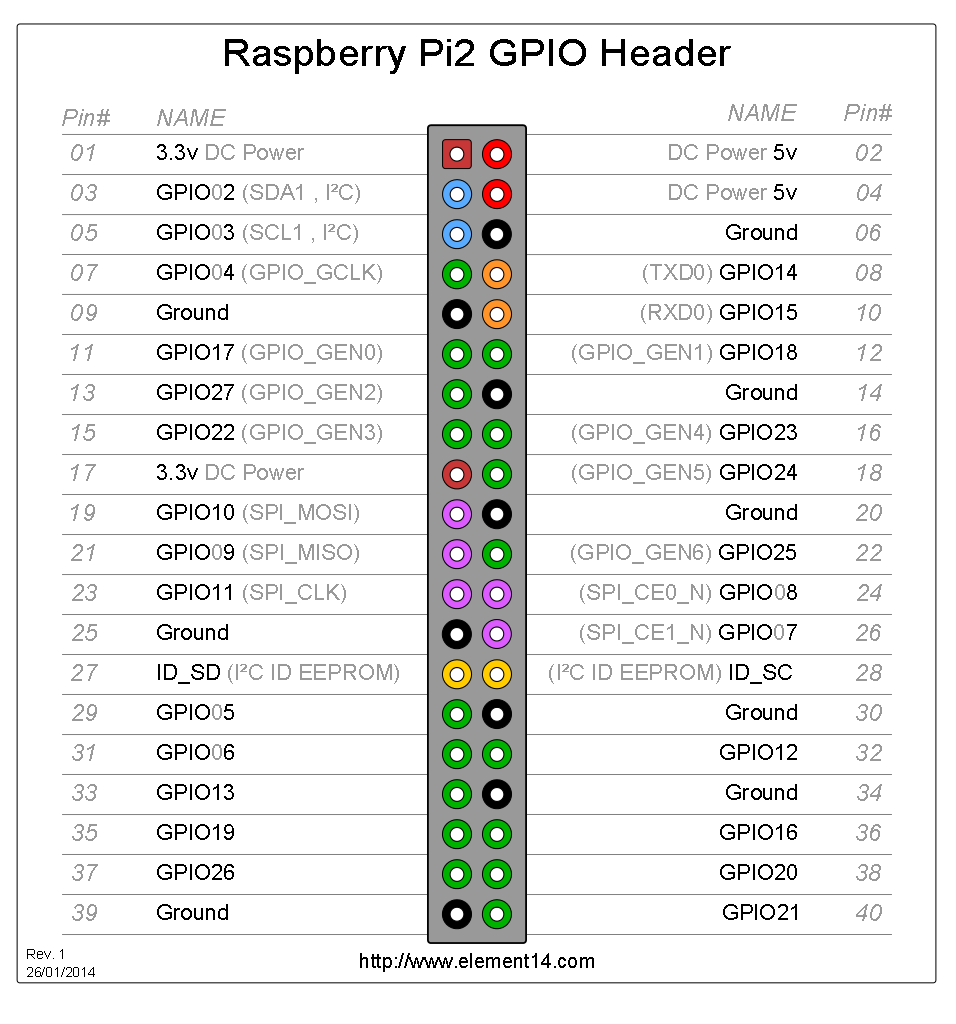
Raspberry Pi -mikrotietokoneen käyttöjärjestelmäksi valitsimme Raspbian GNU/Linux 7 (wheezy) Distron - joka on Debian-pohjainen Linux käyttöjärjestelmä.

Raspberry Pi:lle asennettiin ennen sensori osuuden aloittamista seuraavat paketit:

* Python
  + Tarvittavat kirjastot:
    - RPi-GPIO
* Screen
* CoAPthon

## 2.1.2 Sensori

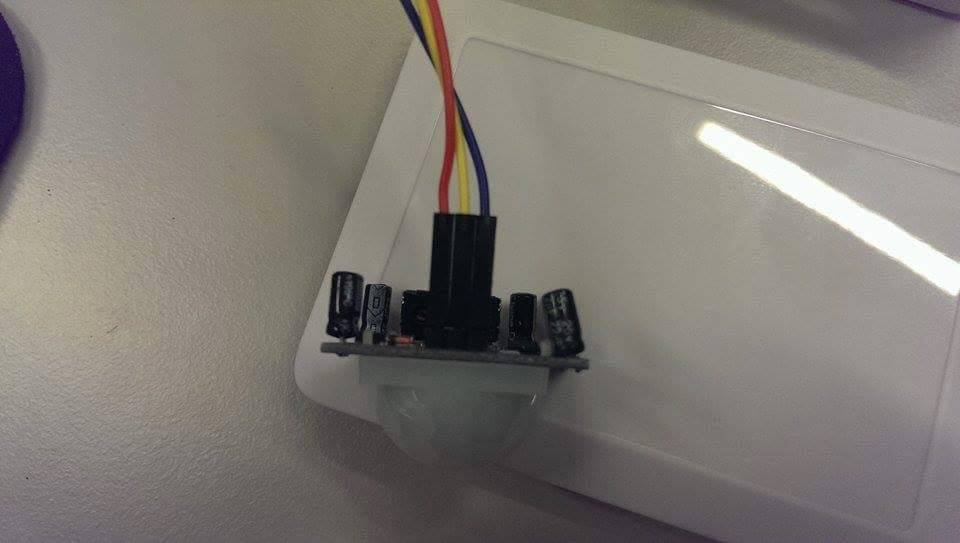
Projektin sensori-osuus toteutettiin Raspberry Pi 2 -mikrotietokoneella.

Sensori kiinnitettiin Raspberry Pi:n GPIO-kiinnityksiin seuraavasti.  


Kuvassa Raspberry Pi 2 GPIO-kiinnikkeet. (element14)

Käyttämämme PIR (passive Infrared) -sensori käytti kolmea eri kiinnikettä:

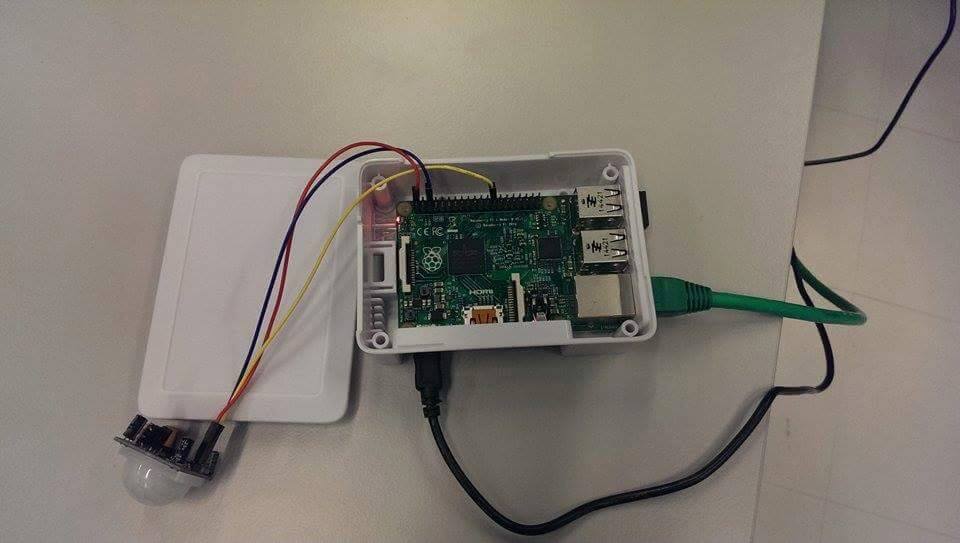
* Ground (maadoitus)
* Digital Out (data)
* +5v (virta)



Kuvassa PIR-sensori. Kuvassa punainen kaapeli on +5v, keltainen digital out ja sininen maadoitus.

Kaapelit ovat kiinnitetty Raspberry Pi:n kiinnikkeisiin seuraavasti:  


+5v kiinnitettynä GPIO-paikkaan 2. Data-kaapeli kiinnitettynä GPIO-paikkaan 7. Maadoitus kiinnikkeessä 6.

Raspberry Pi kokonaisuudessaan:   


## 2.2 Tietokantapalvelin

2.2.1 Ubuntu-palvelin

Projektia varten luotiin virtuaalikone CloudPlatform -pilvipalveluun, johon asennettiin palvelin käyttöjärjestelmällä Ubuntu 14.04 64-bit. Palvelimen tarkoitus on pyörittää projektia varten luotua tietokantaa, johon PIR-sensorin lähettämä tieto tallennetaan.

Palvelimelle on avattu seuraavat portit: portti 5683 CoAP-protokollalle, portti 3306 MySQL -palvelimelle, portti 22 SSH. Ubuntu-palvelinta on ylläpidetty SSH-yhteyden kautta ja päivitetty aina ajan tasalle etäyhteyksien yhteydessä.

2.2.2 Apache

Koneelle on asennettu Apache (apache2), joka on HTTP palvelinohjelmisto. Apache toimii palvelinalustana PhpMyAdminille ja MySQL-tietokantapalvelimelle. Apachen asennuksessa asennettiin paketti: ”*apache2*” ja aktivoitu “*a2enmod userdir”* -moduuli komennolla “*sudo a2enmod userdir*”.

2.2.3 MySQL – palvelin

Ubuntu-palvelimelle asennettiin paketit ”*mysql-client”* ja ”*mysql-server*” . MySQL tietokanta luotiin tekstieditorilla. Tietokantaan luotiin taulut “raw\_data” ja “parsed\_data”. Raspberry Pi:n sensorin tieto tallennetaan ”raw\_data” tauluun. “Raw\_data” -tauluun luotiin kolme arvoa, “entry”, “movement” ja “datetime”.

* Entry - auto increment, kertoo merkinnän järjestyksen.
* Movement - 0 tai 1. 1 = PIR-sensori on aktiivinen, 0 = PIR-sensori on passiivinen
* Datetime - timestamp.

Tietokannan SQL-dump löytyy muiden dokumenttien tavoin Git-repositoriosta.

http://github.com/yukusu/IoT-Rasp

2.2.4 PhpMyAdmin

PhpMyAdmin on graafinen käyttöliittymä tietokantahallintaan. Projektissa PhpMyAdminiä käytettiin Ubuntulle asennettua MySQL-kannan hallintaan. Palvelimelle asennettiin asennuspaketti ”*phpmyadmin*”.

2.2.5 Node.js ja CoAP-Cli

Ubuntu-palvelimelle asennettiin Node.js framework-paketti “nodejs”.

Paketin asentamisen jälkeen Ubuntu asetettiin käyttämään Nodejs kutsumalla terminaalissa “nodejs”. Oletuksena olevan “*node*” sijaan, komennolla:

*sudo update-alternatives --install /usr/bin/node node /usr/bin/nodejs 10*

Frameworkin asennuksen jälkeen asennettiin “*CoAP-cli*” sovellus, jota käytetään hakemaan PIR-sensorin tila Raspberry-Pi:n CoAP-palvelimelta. Coap-cli asennettiin käyttäen Nodejs:än omaa pakettienhallintaa komennolla:

*npm install coap-cli -g*

2.2.6 SSH

Ubuntu-palvelimelle asennettiin SSH paketti ”*openssh-server*”.

# Koodit

Projektia varten kirjoitetut skriptit sekä yksinkertaiset sovellukset toteutettiin Pythonilla ja Bashillä. Projektia varten kirjoitettua koodia esitellään tässä osiossa.

## 3.1 Pirsensori.py

Projektin pääkomponentti – PIR-sensori ja sen syöttämää tietoa tulkittiin Python-scriptillä.

Ohjelman tarkoitus on käsitellä Raspberry Pi:n infrapunasensorin tulkitsemaa dataa. Kun sensori havaitsee liikettä ohjelma kirjoittaa tapahtumasta aikaleimallisen tiedoston. Ohjelmassa tuotuna RPIO-kirjasto, joka on GPIO-moduuli Rasperry Pi:llä.

Pirsensori.py ajetaan Raspberry Pi:llä aina tietokoneen käynnistyessä.

1. import RPi.GPIO as GPIO
2. import time

Käytetyt kirjastot tuodaan koodiin.

1. GPIO.setmode(GPIO.BCM).

Ohjelman alussa GPIO – kirjasto joka tulkitsee Raspberry Pi:n liitettyjen laitteiden syöttämää tietoa, käyttämään BCM GPIO-viitettä fyysisten kiinnitysten numeroiden sijaan.

1. GPIO.setup(GPIO\_PIR,GPIO.IN)
2. GPIO\_PIR = 7

Kerrotaan GPIO:lle millaisia asetuksia käytetään ja määritellään PIR-sensorin kiinnityksen numero numeroksi 7.

Ohjelma käyttää muuttujia:

1. Current\_State  = 0
2. Previous\_State = 0
3. Movement  = 0
4. Timenow **=** time**.**strftime("%Y-%m-%d %H:%M:%S")

Jotka ovat kaikki oletuksellisesti ’0’, lukuunottamatta muuttujaa ”Timenow”.

Muuttuja ”Current\_State” seuraa PIR-sensorin lukeman nykyistä tilaa. Muuttuja ”Previous\_State” seuraa muttujan ”Current\_State” edellistä tilaa. Muuttuja ”Movement” seuraa PIR-sensorin lukemaa, arvoilla 1 = sensori on aktiivinen ja 0 = sensori ei ole aktiivinen.

Muuttuja ”Timenow” hakee nykyisen aikaleiman, ja muuttaa sen MySQL-ystävälliseen datetime-muotoon.

1. **while** GPIO.input(GPIO\_PIR)==1:

Ohjelma pyörii while-silmukassa, jossa ehtona sensorin aktiivisuus. Silmukka toistetaan kun sensori palaa aktiiviseksi passiivisesta tilasta.

1. **while** True :
3. Current\_State = GPIO.input(GPIO\_PIR)
5. **if** Current\_State==1 **and** Previous\_State==0:
6. print " -- Motion detected -- "
7. Previous\_State=1
8. Movement = 1

Ohjelma lukee PIR-sensorin tilan ja jos tila on aktiivinen asetetaan muuttujille oikeat arvot. Aikaisempi tila (previous\_state) saa nyt nykyisen tilan (current\_state) arvon, sekä liike tallennetaan muuttujaan ”Movement”.

1. print "Movement: %d " % Movement
2. print ("Time: " + time.strftime("%Y-%m-%d %H:%M:%S"))
3. #save the sensor data into the CoAP -directory
4. file = open("/home/pi/CoAPthon/lukema.txt","w",0)
5. file.write(time.strftime("%Y-%m-%d %H:%M:%S "))
6. file.write("%d"%Movement)
7. print "Writing file..."
8. time.sleep(2)
9. print "Done!"
10. print "Sending the log to NSA.."
11. file.close()

Kun silmukan ehdot ovat täyttyneet ja muuttujille on asetettu arvot – kirjoitetaan tiedot ylös. ”print” –komennot tulostavat muuttujat konsoliin vianetsintää varten.

Pythonin file-toiminnolla voidaan kirjoittaa tiedosto helposti. Huomataan, että parametreiksi määritelty ”w” tarkoittaa sitä, että mahdollinen vanha tiedosto ylikirjoitetaan, eikä vanha tieto säily paikallisesti.

”file.write” –kirjoittaa tiedostoon aikaleiman ja liikkeen tilan (Movement)

”file.close” –lopettaa tiedoston käsittelyn.

1. elif Current\_State==0 **and** Previous\_State==1:

Lopuksi silmukka asettaa nykyisen tilan (current\_state) passiiviseksi ja on valmis uuteen kierrokseen.

1. except KeyboardInterrupt:
2. print "Quit"
3. # Reset GPIO settings
4. GPIO.cleanup()

Silmukasta poistutaan ctrl-c –komennolla

## 3.2 db.py

Tietokantayhteys toteutetaan db.py –skriptissä. Skripti ajetaan paikallisesti MySQL-tietokantapalvelimessa. Skriptin ajaa paikallisen bash-skriptin ja syöttää sen tuomat tiedot muokattuna tietokantaan.

db.py ajetaan tietokanta palvelimessa saatuaan käskyn dbrun.py skriptiltä.

1. import MySQLdb
2. import os

Db.py käyttää yllä mainittuja Python kirjastoja.

1. os.system("sudo bash /home/pidb/projekti/iot-projekti/get.sh")

Ohjelmassa ajetaan paikallinen bash-skripti. Skripti hakee ohjelmalle tekstitiedoston, joka myöhemmin viedään kantaan.

1. db = MySQLdb.connect("localhost", "pi", "raspberry", "raspberry" )

Sillä projektissa tietoturvan tarkastamiseen ei riittänyt aikaa, otetaan yhteys tietokantaan selkokielisillä salasanalla.

1. cursor = db.cursor()
3. #Reads the file fetched by the .sh script
4. print "Reading PIR-sensor data.."
5. file = open("/home/pidb/projekti/iot-projekti/get.txt", "r")
6. file\_content = file.read()
7. file.close()
9. #Modifies the file to fit into the db insert
10. Datetime = file\_content[:-3]
11. Movement = file\_content[20:-1]
13. #printing variables
14. print "Printing variables to be inserted into DB"
15. print Datetime
16. print Movement
17. quote = "'"
19. #Run query with variables from CoAP
20. query = "INSERT IGNORE INTO RAW\_DATA (Datetime, Movement) VALUES ("+quote+Datetime+quote+","+Movement+")"
21. print query
23. cursor.execute(query)
24. db**.**commit()

Seuraavaksi tehdään MySQL INSERT-lause ja viedään se kantaan. ”cursor” on tietokantayhteyden väli, jossa suoritetaan tietokantaa koskeva osuus.

”file” –komennoilla avataan ja käsitellään tiedosto, jonka bash-skripti loi. Muuttujat luodaan muokkaamalla tekstitiedoston sisältöä haluttuun muotoon (file\_content). Tekstitiedoston sisältö on vakio, jonka takia merkkejä poistamalla saadaan haluttu tieto muuttujiin.

query = tietokannan insert-lauseke.

curose.execute – tietokannan osuus päättyy.

db.commit – koodi ajetaan.

1. db.close()

Yhteys suljetaan.

## 3.3 dbrun.py

Dbrun.py ajaa db.py –koodin tietyin aikavälein. Sillä db.py ei itsessään sisällä silmukkaa, halusimme luoda yksinkertaisen ohjelman, joka suorittaa ohjelman joka kymmenes sekunti.

dbrun.py ajetaan tietokantapalvelimella, heti palvelimen käynnistyessä.

1. **import** os

Koodinpätkän toteuttamiseen tarvitsimme vain os –kirjaston.

timeout **=** 10.0 *#wait = 10 sec*

*#Work loop*

**def** **doWork**():

os**.**system("sudo python /home/pidb/projekti/iot-projekti/db.py")

**pass**

l **=** task**.**LoopingCall(doWork)

l**.**start(timeout) *#Call for def every 10 secs*

*#Run*

reactor**.**run()

Koodi määritellä ajamaan funktio joka kymmenes sekunti. Funktiossa doWork ajetaan db.py ja määritetään silmukan aikaväli.

## 3.4 get.sh

Get.sh on yksinkertainen bash-skripti, jolla otetaan yhteys CoAP-palvelimeen. CoAP-palvelimen tuoma tieto tallennetaan tekstitiedostoon paikallisesti, jota db.py hyödyntää. get.sh on tallennettu /usr/local/bin –kansioon ja sille on annettu ajettavan tiedoston oikeudet – sudo chmod x+r get.sh

get.sh ajetaan tietokantapalvelimella aina db.py –skriptin silmukassa.

1. #/bin/bash
3. sudo coap get coap://172.28.170.197/Pir > /home/pidb/projekti/iot-projekti/get.txt

Tiedosto kokonaisuudessaan.

## 3.5 CoAPthon

Projektissa käytettiin CoAPthon-sovellusta, joka löytyy osoitteesta <https://github.com/Tanganelli/CoAPthon>. CoAPthon toimii serverinä Raspberry-Pi:llä, joka lukee PIR-sensorin tilan ja kertoo sen eteenpäin CoAP-protokollaa-käyttäen. Projektin kannalta oleelliset muutokset löytyvät alapuolelta.

### 3.5.1 coapserver.py

4 from Pir import Pir

Importattiin Pir-luokka, josta haetaan serverin käyttämät tiedot.

13 self.add\_resource('Pir/', Pir())

Kerrotaan serverille, että käyttää importattua Pir-resurssia.

23 ip = "172.28.170.197"

24 port = 5683

Kerrotaan serverille että käyttää Raspberry-Pi:n omaa IP-osoitetta ja porttia.

## 3.5.2 Pir.py

CoAPthon-sovelluksen asennusta muutettiin seuraavalla tavalla. Tiedosto kokonaisuudessaan:

**from** coapthon.resources.resource **import** Resource

3. **class** Pir(Resource):
4. **def** \_\_init\_\_(self, name="Pir\_lukema", obs=True):
5. super(Pir, self).\_\_init\_\_(name, visible=True, observable=True, allow\_children=True)
6. self.payload = "PIR\_LUKEMA"
8. **def** render\_GET(self, request):
9. file = open('lukema.txt','r')
10. self.payload = file.read()
11. **return** self
13. **def** render\_PUT(self, request):
14. self.payload = request.payload
15. **return** payload
17. **def** render\_POST(self, request):
18. res = BasicResource()
19. res.location\_query = request.query
20. res.payload = request.payload
21. **return** res
23. **def** render\_DELETE(self, request):
24. **return** True

Pir.py -tiedosto luotiin ja se määritettiin lukemaan Pir-sensorin tuottamaa tiedostoa, johon sensorin keräämä tieto tallennettiin.

Pir-sensori tallentaa keräämänsä tiedon tiedostoon “lukema.txt”, jonka funktion “render\_GET” lukee ja palauttaa.