Haaga-Helia
ammattikorkeakoulu Oy

Yksin asuvan fyysisen aktiivisuuden seuranta

Heidi Baruti

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2018

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Heidi Baruti

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Yksin asuvan fyysisen aktiivisuuden seuranta

Sivu- ja liitesivumäärä
32 + 28

Suomessa kotihoidon kysynnän ja hoitajien vähäisyyden vuoksi kotihoitotyötä tulee tehdä mahdollisimman tehokkaasti. Monet toimintakyvyltään heikot asuvat yksin kotonaan eivätkä kaukana asuvat omaiset tiedä henkilön toimintakyvyn tilanteesta.

Aktiivisuuden seurantalaitteella voidaan seurata henkilön toimintakyvyn tasoa. Aktiivisuuden seuraamiseen käytetään mm. aktiivisuusrannekkeita, turvarannekkeita ja valvontajärjestelmiä. Rannekkeet eivät sovellu kotihoidon asiakkaille, sillä ne tulee muistaa laittaa ranteeseen ja ladata. Niiden käyttöönotto voi olla vaikeaa henkilölle, joka ei ole tottunut käyttämään älylaitteita. Valvontajärjestelmiä voidaan tehdä esimerkiksi erilaisilla sensoreilla, jotka vastaanottavat informaatiota ympäristöstään.

Projektin tavoitteena on toteuttaa aktiivisuuden seurantalaite, joka on yksinkertainen, helppokäyttöinen ja suhteellisen halpa. Projektin tulos on konkreettinen laitteisto sekä ohjelmisto ja rajauksena on järjestelmän toimivuuden testaaminen.

Projekti toteutetaan aikavälillä 1.3. – 7.5.2018. Työtä tehdään täysipäiväisesti 5 päivää viikossa etätyönä käyttäen ketterän ohjelmistokehityksen toimintaperiaatteita yleisellä tasolla.

Projektin tuloksena syntyy IoT -sensorijärjestelmä, jonka tekoon käytetään Raspberry Pi 3 -tietokonetta kosketusnäytöllä, Arduino Pro Mini -kehitysalustaa, RF -vastaanottomoduulia sekä neljää liike- ja neljää ovi/ikkunatunnistinta. Arduino ohjelmoidaan Arduino -ohjelmointikielellä, muu ohjelmisto kirjoitetaan Pythonilla. Projektissa käytetään MySQL -tietokantaa.

Asiasanat
IoT, monitorointi, sensorit, Arduino, Raspberry Pi

Sisällys

1 Johdanto

Projektissa tuotetaan tuote Domax Oy:lle. Domax Oy on vuonna 1989 perustettu ohjelmistojen suunnittelu- ja valmistusyritys (YTJ). Projektissa luodaan yksinkertainen sensorijärjestelmä, jota voidaan halutessa laajentaa. Laitteisto rakennetaan mahdollisimman pitkälle valmiista olemassa olevista tavaroista ja sovellus kasataan internetistä löytyvistä malliratkaisuista muokkaamalla ja yhdistämällä ne yhdeksi kokonaisuudeksi. Asennuksen helppouden kannalta on tärkeää, että lopputulos on langaton. Projekti toteutetaan noin kymmenessä viikossa ja sen rajauksena on järjestelmän toimivuuden testaaminen.

Ratkaisun avulla pyritään seuraamaan sekä ylläpitämään henkilön aktiivisuutta passiivisin ja interaktiivisin keinoin, auttamaan kotihoitoa määrittelemään hoidon määrän tarve asiakkaalle ja vähentämään aktiivisuuteen liittyvää kirjaamisen määrää sekä tuomaan omaisille mielenrauhaa henkilön toimintakyvyn tilasta.

Opinnäytetyössä käsitellään aiheeseen liittyen teoreettinen ja toiminnallinen osa. Teoreettisessa osassa kerrotaan aiheeseen liittyvät taustatiedot sekä tehdään pienimuotoinen kartoitus olemassa olevista teknologioista. Toiminnallisessa osassa kerrotaan, mitä teknologioita projektiin on valittu ja miten projekti on teknillisesti suunniteltu ja ratkaistu. Lisäksi opinnäytetyön lopussa käydään läpi projektin tulokset ja pohdinta opinnäytetyöprosessista sekä opinnäytetyön tekijän ammatillisesta kasvusta.

Projektin tuotoksen asiakkaiksi on ajateltu kotihoidon asiakaskuntaa. Helsingin Seutu (2018) määrittelee kotihoidon asiakkaiksi vanhukset, kehitysvammaiset sekä pitkäaikaissairaat. Suomessa kotihoitajien mukaan kirjaamiseen kuluu liikaa työaikaa, ja lisäksi Suomen kotihoitajien määrä on Pohjoismaiden matalin (Kröger, Van Aerschot & Puhenparambil 2018, 16-20). Aktiivisuuden seurantalaitteen avulla pystytään vähentämään aktiivisuuden seurantaan liittyvää kirjaustyötä vapauttaen hoitajien aikaa työn muihin osiin.

Aktiivisuuden seurantaan on olemassa valmiita ratkaisuja, kuten rannekkeita ja IoT -järjestelmiä. Shovic (2016, luku 1-2) kuvaa esineiden Internet -ratkaisuja laitteiksi, jotka kommunikoivat toistensa tai serverin kanssa, joka on yhteydessä Internetiin. Erilaiset sensorijärjestelmät ovat esimerkkejä aktiivisuuden seurantaan käytettävistä IoT -ratkaisuista.

Puettavat sensorit, kuten rannekkeet, eivät ole sopivia yksin asuvien toimintakyvyltään heikkojen henkilöiden, kuten vanhusten tai pitkäaikaissairaiden aktiivisuuden seurantaan, sillä ne eivät ole hyödyllisiä, jos henkilö unohtaa ladata sen. Projektin tuotos pyrkii vas-

taamaan puutteeseen tarjoamalla ratkaisun, missä seurantalaitetta ei tarvitse muistaa pukea, vaan aktiivisuutta seurataan asuntoon asennettavilla langattomilla sensoreilla.

Aholan (2010) mukaan vanhuksille tarkoitetuissa seurantateknologiaratkaisuissa on tärkeää, että vanhukset saataisiin aktiivisiksi käyttäjiksi heidän omasta tahdostaan. Tämän vuoksi projektissa syntyvä sensorijärjestelmä toteutetaan mahdollisimman yksinkertaisesti ja vain muutamalla erityyppisellä sensorilla. Seurantajärjestelmän käyttäjän ei haluta tuntevan oloaan jatkuvasti seuratuksi. Järjestelmästä tehdään niin helppokäyttöinen, että sellainenkin käyttäjä, jolle älylaitteet eivät ole tuttuja, osaa sitä käyttää.

Aktiivisuutta seurataan eri sensoriratkaisuissa erilaisin sensorein. Liiketunnistimet voivat toimia PIR – tai HF -teknologian avulla. PIR tarkoittaa passiivista infrapunaa. Adafruitin (2014) mukaan PIR -liiketunnistin tunnistaa liikkeen, jos liikkuva objekti on lämmin. HF on lyhenne sanoista high frequency, eli korkeataajuus. HF -liiketunnistimen tunnistavan liikkeen taajuuserojen avulla, sillä se lähettää elektromagneettisia aaltoja, jotka heijastuvat liikkuvasta objektista korkeampi- tai matalampitaajuisina (HQ Designs). Muita aktiivisuuden seurantaan käytettäviä sensoreita ovat esimerkiksi lämpötilaa ja painoa mittaavat sensorit.

Sensorit vaativat järjestelmän toimiakseen. Eräs helppokäyttöinen ja edullinen ratkaisu on käsitellä sensorien signaalit Arduino -kehitysalustalla. Arduino on pienikokoinen fyysinen alusta mikrokontrollerille, joka ymmärtää sensoritietoa. Se on laajasti käytetty, jonka vuoksi siihen liittyvää malliratkaisukoodia on Internetissä runsaasti tarjolla. Arduinoa voi ohjelmoida monella eri kehitysympäristöllä, eli IDE:llä, mutta sillä on myös oma Arduino IDE. Arduinoissa on Atmelin 8 -bittinen AVR -mikrokontrolleri, jossa on kolmea erilaista muistia, joita käytetään esimerkiksi konfigurointitiedon ja Arduino -ohjelman tallentamiseen. Sensorien signaalien vastaanottamista varten Arduinoon voidaan lisätä RF -vastaanottomoduuli. RF on lyhenne englannin kielen sanoista radio frequency, radiotaajuus. (Hughes 2016, luku 1, 4, 6 & 9.)

Arduinon avulla sensoreista saadaan luettua tieto, mutta jos tietoa halutaan tallentaa tietokantaan, järjestelmään tulee lisätä tietokone. Shovic (2016, luku 1) kertoo Raspberry Pi:n soveltuvan IoT -projekteihin, koska siinä on useita ajureita sensoreille ja laitteille. Hän kuvailee Raspberry Pi:tä pienikokoiseksi, yhden piirilevyn tietokoneeksi.

Projektin toteuttamiseen valittiin neljä PIR -liiketunnistinta ja sama määrä ovi/ikkunakytkimiä, Arduino Pro Mini, 433MHz:n RF -vastaanottomoduuli ja Raspberry Pi 3 kosketusnäytöllä. Opinnäytetyön toiminnallisessa osassa kuvataan, kuinka sensorien

protokolla on selvitetty ja miten fyysinen laitteisto koottu, miten tietokanta on suunniteltu ja toteutettu, sekä minkälainen ohjelmisto laitteeseen tarvitaan.

2 Projektin tausta

Projektissa syntyvä tuote on IoT -tuote. Internet of Things, eli esineiden Internet viittaa laitteisiin, jotka kommunikoivat toistensa tai serverin kanssa, joka on yhteydessä Internetiin. Nämä laitteet voivat tehdä havaintoja ympäristöstään ja ne voivat olla myös toimilaitteita, eli ne voivat tehdä asioita, kuten laittaa valot päälle tai lukita ovia. IoT:tä voidaan käyttää sensoriverkkojen ja -ryhmien rakentamiseen. Tietokoneen kyky ymmärtää sen ympäristö sensorien avulla mahdollistaa kerättyjen tietojen lähettämisen, analysoinnin, tai toiminnan Internetiin. (Shovic 2016, luku 1-2.)

Projektissa syntyvällä tuotteella pyritään yksin asuvan henkilön aktiivisuuden seuraamiseen ja ylläpitämiseen interaktiivisin keinoin. Tuotteen avulla henkilön omaisille halutaan tuoda mielenrauha siten, että omaiset näkevät lähes reaaliajassa henkilön aktiivisuusmerkinnät ja pystyvät kaavion avulla helposti näkemään, miten aktiivisuusmerkintöjen määrä sijoittuu henkilölle laadittuun vertailuarvoon nähden. Laitosnäkymä, joka sopii esimerkiksi kotihoidolle, auttaa palvelutarpeen uudelleenmäärittämisessä. Näkymästä on helppo huomata, jos henkilön aktiivisuus on laskenut tai noussut.

2.1 Kotihoito Suomessa

Kotihoito tarkoittaa kotipalvelua ja kotisairaanhoitoa. Kotipalvelu tarkoittaa pääasiassa kotiavustajien sekä lähi- ja kodinhoitajien antamaa apua päivittäisiin tehtäviin, ja sitä voi saada henkilö, jonka toimintakyky on heikko. (Sosiaali- ja terveysministeriö.) Helsingin Seutu (2018) määrittelee kotihoidon asiakkaiksi vanhukset, kehitysvammaiset sekä pitkäaikaissairaat.

Savela (2016) kertoo, että Suomessa pitkäaikaisvuodeosastoja suljetaan ja niistä kehitetään kuntoutusyksiköitä, jotta saadaan potilaat kuntoutettua ja lähetettyä takaisin kotiin. Savela mainitsee, että Marja Heikkilän, Keski-Suomen SOTE2020 -hankkeen hankepäällikön, mukaan tavoitteena on saada ihmiset asumaan kotonaan mahdollisimman pitkään. On tärkeää, että yksin asuvan hoidon määrän tarve osataan arvioida oikein, jotta henkilö kykenee asumaan kotonaan. Terveyden ja hyvinvoinnin laitoksen (2017, 4) mukaan uudelle kotihoidon asiakkaalle arvioidaan ensimmäisenä palvelutarve, jonka tekemiseen voi kulua useampikin tunti. Hoitotarve ei kuitenkaan välttämättä pysy samana pitkään, jonka vuoksi heikkenevä toimintakyky saattaa heiketä nopealla tahdilla.

Nordcare2 -tutkimus paljastaa, että Suomessa kotihoitajien määrä on Pohjoismaiden matalin ja heidän työpaineensa korkein. Kotihoidossa hallinnolliset tehtävät ja kirjaamisen määrä ovat nousussa, ja hoitajat kokevat, että kirjaamiseen kuluu liikaa työaikaa. (Kröger

ym. 2018, 16-20.) Kun Suomessa pyritään siihen, että ihmiset pystyvät asumaan kotonaan mahdollisimman pitkään, kotona asuvien vanhusten määrä lisääntyy ja näin ollen lisääntyy myös kotihoidon tarpeen määrä. Kun kotihoitajien määrä on matala, on erittäin tärkeää, että hoitoa osataan tehdä mahdollisimman tehokkaasti. Korkean työpaineen vuoksi monet hoitajat saattavat miettiä alan vaihtoa, joka kasvattaa pulaa kotihoitajista. Tämän vuoksi olisi tärkeää saada työ tehdyksi tehokkaasti, jotta hoitajien työpainetta saataisiin myös alennettua. Työtehokkuutta voidaan parantaa hyödyntämällä aktiivisuuden seurantaan käytettävää teknologiaa.

2.2 Olemassa olevat teknologiat aktiivisuuden seurantaan

Aktiivisuutta voidaan seurata aktiivisuusrannekkeilla. Ranneke on puettava sensori, joka tallentaa tietoa henkilöstä, joka sitä pitää. Esimerkiksi Polar Electro (2018) valmistaa Polar Loop 2 -nimistä aktiivisuusranneketta, joka tallentaa ladattuna ja ranteeseen puettuna tietoa siitä, kuinka paljon henkilö istuu, seisoo, liikkuu ja nukkuu. Polar kertoo lisäksi sivuillaan, että ranneke hälyttää, jos henkilö on istunut liian kauan paikallaan. Tämä on hyvä ominaisuus henkilölle, joka istuu päivän aikana pitkiä aikoja kerrallaan. Aktiivisuusrannekkeet ovat monille tuttuja kuntoilun maailmasta.

Monille toimintakyvyltään heikoille, yksin asuville henkilöille suositellaan turvaranneketta aktiivisuusrannekkeen sijaan. Monet turvarannekkeet eivät sisällä varsinaista sensoria, joka mittaa henkilön aktiivisuutta, vaan rannekkeessa on yksi nappi, jota henkilön tulee painaa hätätilanteessa. Hätätilanne voi olla esimerkiksi henkilön kaatuminen. Nappia painamalla rannekkeesta lähtee hälytys esimerkiksi kotihoidolle, joka saapuu auttamaan henkilön ylös. Tässä ratkaisussa on tärkeää, että henkilö kykenee painamaan nappia kaatumisen jälkeen. On olemassa myös turvarannekkeita, joissa hälytys lähtee automaattisesti, jos henkilön aktiivisuustaso poikkeaa normaalista. Tällainen ratkaisu sisältää sensorin, joka takaa paremmin sen, että apu saapuu paikalle.

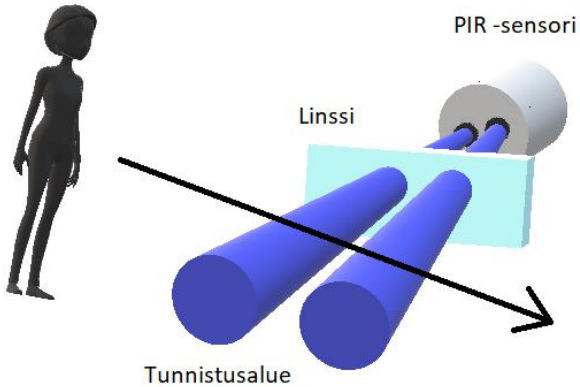
Rannekeratkaisu on melko yleinen, mutta ei välttämättä järkevin vaihtoehto toimintakyvyltään heikolle yksin asuvalle henkilölle. Rannekeratkaisujen toimivuus riippuu rannekkeen pitäjästä, sillä henkilön pitää muistaa pukea ranneke ja etenkin aktiivisuusrannekkeita täytyy muistaa ladata lähes joka päivä. Aktiivisuuden seurannan tarkkuus kärsii automaattisesti niinä aikoina, kun henkilö lataa ranneketta, sillä se ei mittaa silloin aktiivisuutta. Joidenkin aktiivisuusrannekkeiden käyttöönotto tai käyttö voi olla lisäksi haastavaa. Tämän vuoksi aktiivisuusrannekkeet eivät välttämättä sovi henkilölle, joka ei ole tottunut käyttämään älylaitteita. Toimintakyvyltään heikoille yksin asuville henkilöille paras ratkaisu saattaa olla sellainen, jossa laitteen toimivuus ei riipu heistä itsestään. Rannekkeita parempi

ratkaisu voi sen vuoksi olla sensorijärjestelmä. Sensorit tallentavat tietoa ympäristöstään, eivätkä välttämättä vaadi yksin asuvalta henkilöltä ylimääräistä toimintaa.

2.2.1 Liiketunnistin aktiivisuuden seurantaan ja RF -vastaanottomoduuli

Sensori muuttaa ei-sähköistä tietoa, kuten liikettä, sähköiseen, mikrokontrollerin ymmärtämään muotoon. Sensorit voivat tunnistaa esimerkiksi liikkeen, valon, kosteuden tai lämmön ja niiden mittaustieto on joko analogista tai digitaalista. Analogisen tiedon voi muuttaa A/D -muuntimella digitaaliseksi, eli nolliksi ja ykkösiksi. Sensorit voivat olla passiivisia, jolloin ne vastaanottavat informaatiota, tai aktiivisia, jolloin ne lähettävät esimerkiksi radiosignaaleja ja odottavat vastetta. Ne toimivat yleensä paristoilla ja niiden virrankulutus on pientä, mutta virrankulutuksen vähentämiseksi entisestään sensorit voidaan ohjelmoida tekemään mittauksia tietyin väliajoin. Langattomissa sensorimalleissa on langattoman verkon radiopiiri, jota käytetään tiedon viemiseen sensorista eteenpäin. Sensorit voivat ilmoittaa tietoliikenneverkolle olevansa aktiivisia radiopiirinsä avulla automaattisesti. (Collin & Saarelainen 2016, III osa, luku 9-10.)

Liike- ja läsnäolotunnistimet tunnistavat liikkeen tietyllä alueella. Läsnäolotunnistimet tunnistavat pienemmän liikkeen kuin liiketunnistimet. Läsnäolotunnistimen yleisin asennuspaikka on tilan katossa, kun taas liiketunnistimen voi asentaa myös seinään tai tilan kulmaan. (ABB, 4.) Jotkin sisätiloissa käytettävät liiketunnistimet käyttävät PIR -, eli passiivinen infrapuna -teknologiaa. PIR -sensorissa on kaksi infrapunalle herkkää aukkoa (kuva 1). Sensori tunnistaa liikkeen, jos liikkuva objekti on lämmin.



Kuva 1 PIR -sensorin toiminta (Chank 2017)

Sensori tunnistaa liikkeen esimerkiksi ihmisestä ruumiinlämmön vuoksi. Lämpimän objektin liikkuessa sensorin ohi se piilottaa ensin toisen puolen sensorista, mikä saa aikaan positiivisen differentiaalisen muutoksen sensorin kahden puolen välillä, ja objektin lähdet-

työ alueelta tapahtuu negatiivinen differentiaalimuutos. Sensori toimii näiden muutospulssien varassa. (Adafruit 2014.)

PIR -sensorien lisäksi on olemassa HF -sensoreita, jotka puolestaan lähettävät elektromagneettisia aaltoja 5.8 GHz:n taajuudella. HF on lyhenne sanoista high frequency, eli korkeataajuus. Kun tilassa ei tapahdu liikettä, signaali heijastuu saman taajuisena, mutta jos tilassa liikutaan, signaali heijastuu matalampana tai korkeampana liikkuvasta objektista, jolloin sensori tunnistaa liikkeen. (HQ Designs.)

Karppilan (2014) rakentama liikkeen ja lämmön monitorointilaite pohjautuu Arduinoon ja sensoreihin. Hughes (2016, luku 1 & 9) kuvailee Arduinoa mikrokontrollerialustaksi, joka soveltuu käytettäväksi sensorien kanssa sen vuoksi, että sensorit muuttavat keräämäänsä tietoa mikrokontrollerin ymmärtämään muotoon. Arduinolla on monia erilaisia levytyyppejä, joista Karppila on valinnut laitteeseensa Yúnin.

Sensorit liitetään Arduino -levyn analogisiin tai digitaalisiin pinneihin. Laitteeseen tarvittavien pinnien määrä on yksi levytyypin valintaan vaikuttava tekijä. Sensoreilla voidaan mitata esimerkiksi lämpötilaa, liikettä, kosteutta tai painoa. Hughes (2016, luku 1 & 9.) Karppila on valinnut mitattaviksi liikkeen ja lämpötilan, ja lopullista laitetta voidaan käyttää esimerkiksi lämpötilan tarkkailuun tai kulunvalvontaan, jolloin laite toimii kodin turvalaitteena. Karppila ei käytä laitettaan aktiivisuuden seurantaan, mutta hän on julkaissut laitteeseensa liittyvät koodit GitHub -palvelussa, jonka vuoksi laitetta voi jatkokehittää haluamallaan tavalla, esimerkiksi aktiivisuuden seurantalaitteeksi. (Karppila 2014.)

Sensorien signaalit voidaan vastaanottaa RF -vastaanottomoduulilla. Useimmissa malleissa ei ole sisäänrakennettua antennia, vaan niihin tulee lisätä erillinen antenni. RF on lyhenne englannin kielen sanoista radio frequency, eli radiotaajuus. Hughesin (2016, luku 9) mukaan RF -moduuleilla on joko 315MHz:n tai 433MHz:n taajuus, ne koostuvat erillisestä lähettimestä ja vastaanottimesta, ja niillä on jopa 150 metrin kantama. Eri mallien kantamissa on eroja, ja jos niillä ei ole sisäänrakennettua antennia, myös antenni vaikuttaa RF -vastaanottomoduulin toimintaan.

2.2.2 Arduino ja Arduino IDE

Arduino on helppokäyttöinen, suhteellisen halpa ohjelmoitava laite. Arduino on käytännössä fyysinen alusta AVR -mikrokontrollerille. Useimmat Arduinot käyttävät Atmelin 8-bittistä AVR -mikrokontrolleria, jossa on esiasennettuna käynnistyslataimen laiteohjelmisto. Arduinolla on monia eri levytyyppejä, joista voidaan valita projektiin sopiva levytyypin

koon ja pinnien mukaan. Arduinosta puhuttaessa pinnit tarkoittavat Arduino -levyn input/output liitoskohtia. (Hughes 2016, luku 1 & 4.)

Arduino soveltuu käytettäväksi erilaisten sensorien kanssa, sillä sensorit muuttavat ympäristöstään hankkimaa tietoa mikrokontrollerin ymmärtämään muotoon. Arduinolla voidaan tehdä todella monia asioita, esimerkiksi erilaisia monitorointilaitteita, pieniä robotteja ja automaatiolaitteita, pitäen mielessä sen rajoitukset. Arduinon rajoituksia ovat sen muistin koko ja nopeus. Arduinoja ei ole suunniteltu käyttämään ulkoista muistia mikrokontrollerin muistin lisäksi, ja käyttäjän oletetaan ajavan suhteellisen lyhyttä ohjelmaa. Nopeus ei ole suuri, koska oletetaan, että laitteiden, joita Arduinon avulla rakennetaan, ei tarvitse toimia kovin suurella nopeudella. Esimerkiksi lämpötilan tarkkailuun rakennetun laitteen ei tarvitse mitata lämpötilaa useammin kuin kerran kymmenessä sekunnissa. Yhtenä rajoitteena on myös se, ettei mikrokontrollerin ja ulkoisen maailman välillä ole puskurointia, jonka vuoksi mikropiirin voi käräyttää toimintakelvottomaksi, jos laitteeseen syöttää enemmän sähkövirtaa kuin se pystyy käsittelemään. (Hughes 2016, luku 1 & 9.)

Arduino on tavaramerkitty, jonka vuoksi vain viralliset Arduinot voivat käyttää nimenään Arduinoa. Avoimen lähdekoodinsa ansiosta kuka tahansa voi kopioida sen kaaviot ja käynnistyslataimen koodin, mutta kopioiden tulee käyttää eri nimeä. Usein käytetään -duino tai -ino -loppuisia tuotenimiä, kuten Freeduino ja Diavolino. Erilaiset ”duinot” ovat Arduinon kanssa yhteensopivia joko laitteistoltaan tai ohjelmistoltaan. Arduino -yhteensopivia laitteita voidaan ohjelmoida Arduino IDE:llä samoin kuin virallisia Arduinoja, valitsemalla yhteensopiva Arduino -levytyyppi IDE:n pudotusvalikosta. (Hughes 2016, luku 1.)

Arduino käyttää kääntäjää ja muita työkaluja luodakseen suoritettavia ohjelmia, jotka voidaan siirtää toiselle tietokoneelle. Ohjelmat kehitetään toisen järjestelmän avuin, esimerkiksi Linuxilla, josta ohjelmat siirretään tai ladataan Arduinon AVR mikrokontrollerille. Ohjelmien siirtäminen ja lataaminen mikrokontrollerille onnistuu Arduinossa helposti sen käynnistyslataimen ansiosta, joka on pieni esiladattu laiteohjelma. Käynnistyslataimelle on varattu tila mikrokontrollerin flash -muistissa. Jos mikrokontrolleri on määritetty käyttämään käynnistyslataajaa eikä sitä ole korvattu muulla ohjelmalla, se ajetaan aina, kun Arduino käynnistetään. AVR -mikrokontrolleri kykenee lataamaan ohjelmakoodin flash -muistiin käynnistyslataimen ja sarjaliitännän kautta. (Hughes, luku 5.)

Arduino IDE on Arduinon kehitysympäristö, joka käyttää ohjelmia, joita kutsutaan sketcheiksi, eli luonnoksiksi. Arduino IDE:n kanssa toimitetaan Arduinon runtime AVR-GCC -kirjasto. GCC, GNU Compiler Collection, perustuu konseptille, jossa hyödynnetään eri

käyttöliittymien tiettyjen kielien symbolisia prosessoreita ja välitetään tuloksena syntyvä välikoodi tietyn kohdealustan backendille. AVR-GCC on kääntäjä, joka on rakennettu GCC:n pohjalta ja joka on määritetty luomaan objektikoodia AVR -mikrokontrollereille. AVR-GCC hyväksyy komentoriviargumentteja, switchejä. Koodin laatimiseen kohdeprosessorille tarvitaan vain muutama switch. Arduino IDE:n käynnistäessä sen näytölle tulee oletuksena seuraava koodi.

```
void setup() {
}
void loop() {
}
```

Setup -kohtaan kirjoitetaan asennuskoodi, ja loop -kohtaan pääkoodi, jota ajetaan jatkuvasti. Arduino IDE käyttää tyypillisesti C++ ja C -ohjelmointikieliä. Arduino IDE ja sen käynnistyslatain ovat ne ominaisuudet, jotka tekevät ohjelmoinnista helppoa kokemattomille ohjelmoijille, mutta ohjelmointi sujuu myös muiden kehitysympäristöjen ja komentorivin kautta. Yksi esimerkki on PlatformIO, komentorivityökalu, joka perustuu Python -ohjelmointikieleen. PlatformIO:lla on ennalta määritetyt kehykset erilaisille levytyypeille ja kyky integroida IDE:n tai sen kaltaisen editorin kanssa. Muita usein käytettyjä työkaluja Arduinon ohjelmoimiseen ovat esimerkiksi Eclipse, Sublime Text ja Visual Studio. (Hughes 2016, luku 1, 5 & 6.)

2.2.3 Atmel 8 -bittinen AVR -mikrokontrolleri

Atmelin 8 -bittiset AVR -mikrokontrollerit käyttävät yhteistä keskusyksikköä ja sisäisen tietoväylän ympärille rakennettua modulaarista sisäistä arkkitehtuuria. Ne koostuvat AVR -prosessorista, erilaisista tulo- ja lähtösignaaleista, ajastuksesta, analogi-digitaalimuunnoksista, laskurista/ajastimesta sekä sarjaliitännän toiminnoista. Niissä voi olla myös muita toimintoja. Atmelilla on monenlaisia AVR -mikrokontrollereita, joita voidaan valita sen mukaan, mitä projektiin tarvitaan. Eri mikrokontrollereissa on eri määrä pinnejä. Projektin tarpeiden mukaan voidaan valita mikrokontrolleri, jossa on sopiva määrä pinnejä, jotta piirilevyn tilaa voidaan käyttää mahdollisimman tehokkaasti. (Hughes 2016, luku 2.)

AVR -laitteissa on kolmentyyppistä muistia, EEPROM, Flash ja SRAM. EEPROM on lyhenne Electrically Erasable Programmable Read Only Memorysta, joka vapaasti käännettynä tarkoittaa ohjelmoitavaa lukumuistia, jonka voi sähköisesti tyhjentää. EEPROM:ia käytetään kirjoittaessa pieniä määriä konfigurointitietoa IoT -laitteelle. Tämä konfigurointi-

tieto luetaan, kun laite käynnistetään. EEPROM:iin tallennetaan tietoa, jonka pitää pysyä tallennettuna ohjelmistoversioiden ja tehosyklien välillä. Flash -muistiin tallennetaan ohjelmakoodi. Tämä tarkoittaa Arduinolla koodia, joka kirjoitetaan esimerkiksi Arduino IDE:llä. SRAM -muistia käytetään tilapäisen datan tallentamiseen. Vain SRAM -muisti menettää datan, jos AVR -prosessori menettää virran. (Shovic 2016, luku 1; Hughes 2016, luku 2.)

2.2.4 Raspberry Pi

Raspberry Pi on Raspberry Pi Foundationin kehittämä suhteellisen halpa ja pienikokoinen tietokone, jossa on vain yksi piirilevy. Se käyttää useita eri käyttöjärjestelmiä, joista yleisin on Ubuntu Linuxin Raspian -julkaisu. Raspberry Pi:ssä on useita ajureita sensoreille ja laitteille, jonka vuoksi se soveltuu mainiosti IoT -projekteihin. Siinä on myös yleinen asynkroninen vastaanotin/lähetin, jota käytetään sarjaviestintään eri laitteiden välillä. (Shovic 2016, luku 1; Upton, Duntemann, Roberts, Mamtora & Everard 2016, luku 1).

Raspberry Pi:ssä on 40 GPIO pinniä. GPIO -lyhenne tulee englannin kielen sanoista general-purpose input/output ja tarkoittaa input/output -pinnejä, jotka on tarkoitettu yleiseen käyttöön. Näiden pinnien kautta Raspberry Pi voi ohjata monia laiteita, kuten lamppuja, ovikelloja tai robotteja. Pinnit ovat kahdessa rivissä, ja niiden tarkoitukset voi tarkistaa GPIO -kuvasta, jossa ne on selitetty. Raspberry Pi pystyy kommunikoimaan myös langattomasti Bluetoothin tai Wi-Fin kautta. Raspberry Pi:n moniin ominaisuuksiin kuuluvat esimerkiksi useat USB -portit ja Ethernet -portti. Pi 3 -mallissa on lisänä sisäänrakennettu Wi-Fi ja Bluetooth. Wi-Fi tai Ethernet -portti tulevat usein käyttöön, sillä niiden avulla voidaan muun muassa päivittää käyttöjärjestelmää, sekä ladata ja asentaa ohjelmia. (Upton ym. 2016, luku 1.)

Raspberry Pi:n keskusyksikössä on ARM -prosessori. ARM, Advanced RISC Machine, viittaa RISC -malliseen mikroprosessoriarkkitehtuuriin. RISC, reduced instruction set computing, tarkoittaa prosessoriarkkitehtuuria, jossa yksinkertaiset käskyt kulkevat hyvin nopeasti. (Upton ym. 2016, luku 4.)

Shovicin (2016, luku 2) mukaan Python -ohjelmointikieli on laajasti käytetty Raspberry Pi:n ohjelmoijien joukossa. Raspberry Pi:n GPIO pinnejä voidaan ohjelmoida skripteillä. Ohjelmoinnin suositeltava kieli on Python, joka tulee Raspberry Pi:n Raspbian -käyttöjärjestelmässä mukana. Python -ohjelmia voi kirjoittaa esimerkiksi nano -tekstieditorilla. (Upton ym. 2016, luku 12.)

3 Järjestelmän toteuttaminen

Projektin lopputuloksen kaltaista laajempaa yksin asuville vanhuksille suunnattua aktiivisuuden seurantajärjestelmää on koitettu jo melkein 10 vuotta sitten Mikkelin ammattikorkeakoulun Mobiilihoiva -nimisessä projektissa. Kokeilussa 40:n yksin asuvan vanhuksen kotiin asennettiin ainakin liiketunnistimia eri huoneisiin, tunnistimet uuniin ja mikroon tarkistamaan onko niitä käytetty, sekä tunnistin sänkyyn. Näistä tunnistimista tieto kulki kotihoidon työntekijöille näkymään erilaisina kaavioina, joita kotihoidon työntekijä tulkitsi. (Ahola 2010.)

Tässä projektissa, toisin kuin Mobiilihoivassa, halutaan pitää tunnistimien määrä pienenä, sillä pyritään laitteiston yksinkertaisuuteen. Projektissa pyritään minimituotteeseen, jonka laajennusmahdollisuudet ovat suuret lukuisten olemassa olevien tunnistimien johdosta. Projektin sensorien minimoimisella pyritään tuomaan sen käyttäjälle turvallisempi olo sen sijaan, että hän tuntee jokaisen liikkeensä seuratuksi.

Aholan (2010) mukaan on tärkeää, että yksin asuvat vanhukset olisivat näiden ratkaisujen keskiössä niin, että heidät saataisiin aktiivisesti ja omatahtoisesti käyttämään seurantateknologioita. Tämän vuoksi tässä projektissa toteutettava laitteisto ja ohjelmisto on ajateltu niin, että se on mahdollisimman helppo ja yksinkertainen nimenomaan yksin asuvalle henkilölle. Lähinäkymään toteutettavassa interaktiivisuudessa tavoitteena on saada yksin asuva henkilö aktiiviseksi tuotteen käyttäjäksi, ja samalla tällä interaktiivisella keinolla pyritään ylläpitämään henkilön fyysistä aktiivisuutta.

Projektissa toteutettavan langattoman IoT -tuotteen avulla voidaan verrata päivän aktiivisuustasoa henkilön vertailuarvoon. Kaavion ansiosta etänäkymässä omaisten on helppoa huomata heikkenevä toimintakyky.

Tuotoksella pyritään myös vähentämään kotihoidon kirjaamisen työmäärää aktiivisuuden seuraamisen suhteen. Laitosnäkymässä kotihoito voi nähdä, mikä yksin asuvan fyysisen aktiivisuuden merkintöjen määrä on kunakin päivänä, ja he voivat verrata sitä henkilön merkintöjen vertailuarvoon. Tuotteen avulla kotihoidon on helppo nähdä, jos esimerkiksi henkilön fyysinen aktiivisuus laskee nopeasti, ja sen pohjalta he voivat tehdä tilanteeseen sopivat toimenpiteet, kuten palvelutarpeen uudelleenmäärittäminen.

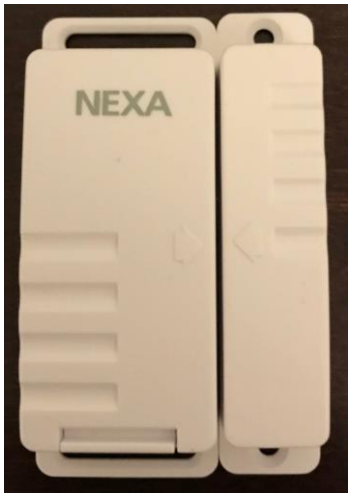
Projektia lähdetään toteuttamaan valitsemalla projektiin sopivat teknologiat, jonka jälkeen lähdetään rakentamaan laitteistoa ja siihen tarvittavaa ohjelmaa. Tämän jälkeen suunnitellaan ja luodaan tietokanta, sekä tarvittava ohjelmisto tiedon tietokantaan lisäämiseen.

Lopuksi luodaan lähi-, etä- ja laitosnäkymät. Opinnäytetyöntekijä tekee projektia etätyönä viitenä päivänä viikossa noin kymmenen viikon ajan. Projektin toteuttamiseen liittyvät päätökset tehdään yhdessä toimeksiantajan kanssa. Toimeksiantajaan pidetään yhteyttä sähköpostin avulla, ja projektin aikana järjestetään muutamia lähipäiviä, joiden aikana käsitellään projektiin liittyviä ongelmatilanteita. Toimeksiantaja suunnittelee ja toteuttaa järjestelmän palvelinpuolen.

3.1 Teknologioiden valinta

Projektin tietokoneena käytetään Raspberry Pi 3:a kosketusnäytöllä. Kosketusnäytön ajateltiin olevan hyvä ja yksinkertainen ratkaisu lähinäkymän näyttämiselle. Näin yksin asuva henkilö ei tarvitse erillistä näyttöä näkymän vuoksi. Raspberry Pi 3 -tietokone sijaitsee kosketusnäytön kääntöpuolella olevassa pienessä kotelossa, joka toimii samalla kosketusnäytön jalkana niin, että näyttö kallistuu käyttäjään päin.

Projektissa käytetään neljää langatonta liiketunnistinta, sekä neljää ovi/ikkunatunnistinta. Näiden uskotaan olevan maksimimäärä kattavaan aktiivisuuden seuraamiseen. Langattomat tunnistimet käyttävät virtalähteenä paristoja, joiden tulisi pienen virrankulutuksen vuoksi kestää muutaman vuoden. Projektissa käytetään NEXA -merkkisiä langattomia ovi/ikkunatunnistimia (kuva 2), jotka maksavat n. 15€/kpl, sekä langattomia NEXA PIR -liiketunnistimia (kuva 3), jotka maksavat n. 25€/kpl.



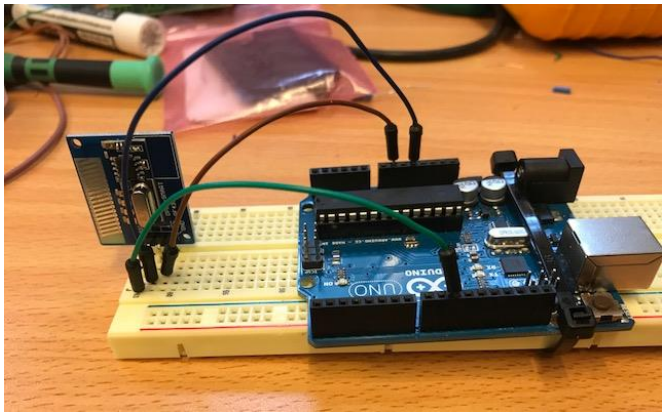
Kuva 2 NEXA ovi/ikkunakytkin

Kuva 3 NEXA PIR -liiketunnistin

Arduino on suunniteltu helppokäyttöiseksi ja edulliseksi ratkaisuksi, johon voi liittää sensoreita (Hughes 2016, luku 1). Tämän vuoksi projektiin valitaan Arduino lukemaan sensoritietoa. Projektissa käytetään Arduino Pro Miniä (8MHz, 3.3V). Arduino Pro Mini on tarpeeksi pieni siihen, että se mahtuu RF -vastaanottomoduulin kanssa Raspberry Pi 3:n kosketusnäytön kotelon sisälle. Arduino Pro Miniin liitetty RF -vastaanottomoduuli vastaanottaa sensorien signaaleja.

Projektin aikana testattiin Arduino Pro Minin lisäksi myös Arduino UNOa sekä Arduino Microa, jotka eivät soveltuneet projektiin yhtä hyvin kuin Arduino Pro Mini korkeamman tehonsa (5V) vuoksi. Testauksen tulosten perusteella matalampitehoinen Arduino kuulee lähimmät signaalit paremmin. 3.3V Arduino soveltuu projektiin paremmin myös sen vuoksi, että sen sarjaliikenteen kytkeminen Raspberry Pi:n sarjaliikenteeseen onnistuu yksinkertaisella kytkennällä ilman regulaattoria tai jännitejakajaa, koska Raspberry on myös 3.3V. UNO -malli ei lisäksi mahdu kosketusnäytön kotelon sisään suuremman kokonsa vuoksi. Arduino Pro Minissä ei ole USB -porttia, jota tarvitaan Arduinon liittämiseen tietokoneeseen, jossa se ohjelmoidaan. Tätä varten Arduinoon tulee liittää USB-sarjaportti -adapteri.

Projektissa testattiin myös erilaisia RF -moduuleja. Eräs kokeilussa ollut RF -moduuli on Wenshing -merkkinen RWS-371 -sarjan 433MHz langaton vastaanotinmoduuli. Tähän vastaanotinmoduuliin tulee lisätä antenni, jotta se pystyy vastaanottamaan sensorien signaaleja. Testauksessa kävi ilmi, ettei tämä vastaanotin pystynyt vastaanottamaan sensorien tietoja kuin muutaman metrin etäisyydellä. Projektin aikana testattiin erilaisia Arduinon mallien ja RF -moduulien yhdistelmiä koekytkentälevyllä, esimerkiksi Arduino UNOa ja OPEN-SMART vastaanottomoduulia (kuva 4).

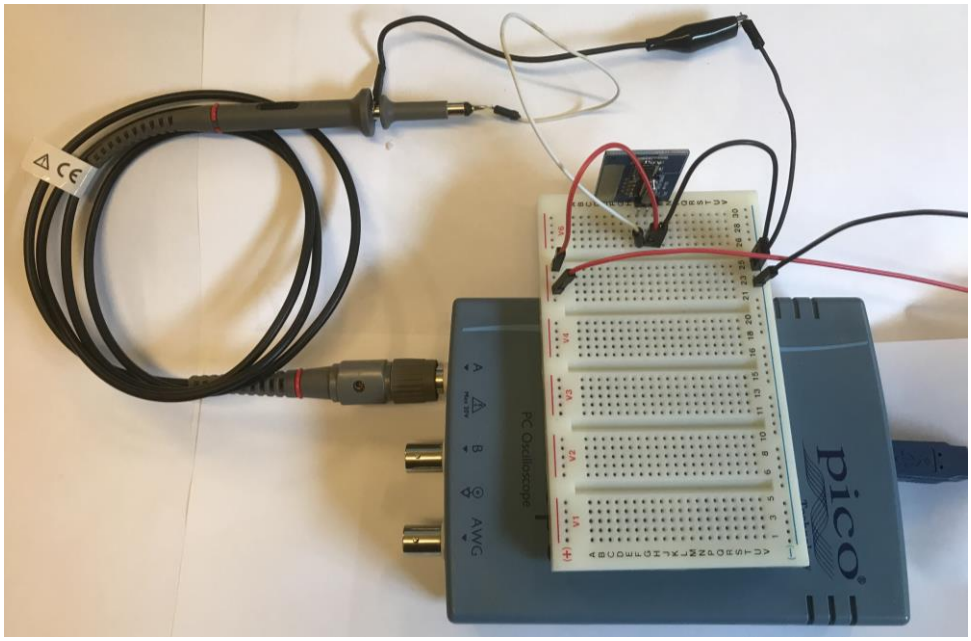


Kuva 4 Arduino UNO ja OPEN-SMART langaton vastaanottomoduuli

Testauksen pohjalta projektiin valittiin OPEN-SMART 433MHz langaton vastaanottomoduuli, jossa on sisäänrakennettu antenni. Kyseinen vastaanottomoduuli sopii projektiin pienen kokonsa, sisäänrakennetun antenninsa ja hyvän kantaman vuoksi. Tämä vastaanottomoduuli kuuli ainoana testatuista sensorien signaalit seinien läpi.

3.2 Protokollan selvittäminen

Projektissa käytettävien sensorien protokolla selvitetään mittaamalla oskilloskoopilla yksi tunnistimen viesti, eli jännitepulssi. Protokolla selvitetään, jotta tiedetään, minkälaista tietoa RF -moduuli vastaanottaa sensoreista. Protokollan selvittämiseen käytetään Pico Technologyn digitaalista oskilloskooppia. Mittausta varten RF -vastaanottomoduulille syötetään 3,3V erillisestä virtalähteestä, esimerkiksi Arduinosta, ja oskilloskoopin maa kytketään samaan maahan RF -vastaanottomoduulin kanssa (kuva 5).



Kuva 5 Oskilloskoopin yhdistäminen RF -moduuliin

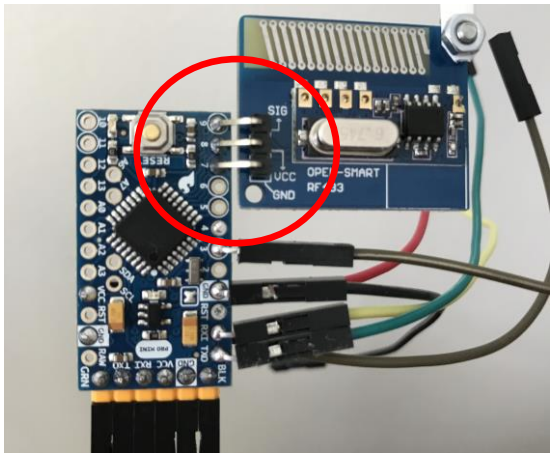
Mittaus otetaan RF -moduulin output -pinnistä, joka on merkitty kirjaimin SIG. Oskilloskooppi yhdistetään USB -kaapelilla tietokoneeseen, jotta mittaustulokset voidaan katsoa tietokoneen näytöltä.

Tunnistimen mitattu viesti (liite 1) alkaa synkronointipulssilla, joka on noin kolmen millisekunnin LOW, jota seuraa 52 ykköstä ja nollaa. Näistä ykkösistä ja nollista kaksi muodostaa yhden bitin, eli 10 on 0 ja 01 on 1. 3.3V Arduinossa HIGH eli 1 vaatii yli 2.4V ja LOW eli 0 alle 0,8V. Näin 52:sta merkistä tulee 26:n merkin binääriluku, jossa on laitteen tunnistenumero, ryhmäkoodi, sekä ON/OFF -tieto. Näistä tiedoista projektin kannalta oleellisia ovat sensorin tunnistenumero ja ON/OFF -tieto, joita tarvitaan Arduino-ohjelmassa.

3.3 Arduinon ja RF -vastaanottomoduulin liittäminen toisiinsa

Arduino Pro Mini liitetään Raspberry Pi:hin USB -kaapelilla USB-sarjaportti -adapterin avulla, jotta se voidaan ohjelmoida. Arduinon ohjelman toteuttamisessa hyödynnetään Internetistä löytyvää malliratkaisua, jota muokataan Arduino IDE:llä. Internetistä löytynyt mallikoodi toimii projektiin valittujen NEXA -merkkisten ovi/ikkunatunnistimien ja liiketunnistimien kanssa. Arduinon ohjelman tarkoituksena on käsitellä informaatiota, jota Arduino vastaanottaa RF -moduulin avulla sensoreista.

Ensin Arduinoon kytketään RF -vastaanottomoduuli vastaanottamaan liike- ja ovi/ikkunatunnistimien signaaleja, jotta sensorien digitaalinen pulssitieto saadaan Arduinossa vastaanotettua. RF -vastaanottomoduuli liitetään Arduinon 7, 8, ja 9 pinneihin (kuva 6).



Kuva 6 RF -vastaanottomoduulin liitäntä Arduino Pro Miniin

Arduino -ohjelmassa (liite 2) pinnit 7 ja 8 määritellään outputeiksi pinMode() -funktiolla. Pinnille 7 annetaan arvo LOW ja pinnille 8 arvo HIGH digitalWrite() -funktion avulla seuraavasti:

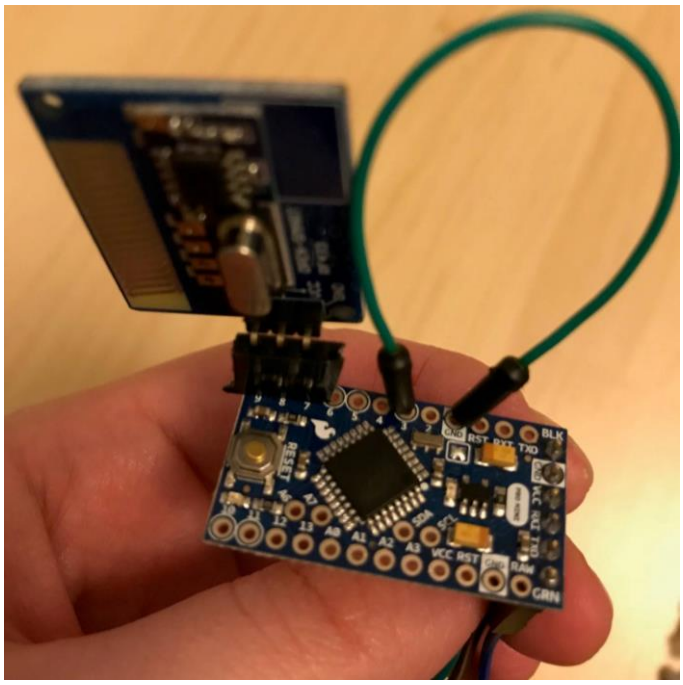
```
pinMode(7, OUTPUT);
pinMode(8, OUTPUT);
digitalWrite(7, LOW);
digitalWrite(8, HIGH);
pinMode(9, INPUT);
```

Kun pinni määritellään outputiksi ja sille annetaan arvoksi LOW, pinni on nollassa voltissa. Jos pinni määritellään outputiksi ja sen arvoksi HIGH, pinni on 3.3:ssa voltissa. Pinni on 3.3:ssa voltissa, koska projektissa käytettävä Arduino Pro Mini on 3.3V. Jotkin Arduino -levytyypit ovat 5V, jolloin pinni olisi viidessä voltissa HIGH asetuksella. Pinni 7 on siis nollassa voltissa ja pinni 8 3.3:ssa voltissa. RF -vastaanottomoduulin VCC -, eli syöttöjännitepinni, liitetään Arduinon pinniin 8 ja ground-, eli maapinni, Arduinon pinniin 7.

RF -vastaanottomoduulin signaalipinni liitetään Arduinon pinniin 9, joka määritellään inputiksi. Pinni 9 vastaanottaa vastaanottomoduulin signaalin, joka on pulssitietoa, eli pinnin 9 tila vaihtelee HIGH ja LOW tilassa. NEXA:n protokollan mukaan HIGH kestää aina saman aikaa, mutta LOW:n kesto vaihtelee. Tämän perusteella saadaan vastaanotetun lähetyksen tietosisältö selvitettyä. LOW -pulssin pituus mitataan Arduinon PulseIn() -funktiolla. NEXA:n protokollan kuvauksen mukaisia pulssipituuksia tarkastelemalla saadaan selville signaalia lähettävän sensorin laitenumero. Jos laitenumero täsmää Arduinoon tallennetun sensorin laitenumeroa, Arduino lähettää siitä sarjaliikenteen avulla tiedon Raspberry Pi:lle.

3.4 Sensoritiedon tallentaminen ja poistaminen Arduinosta

Sensorien tunnistenumerot voidaan tallentaa Arduinon ohjelmaan käyttämällä liitäntäjohtoa, jonka toinen pää kytketään Arduinon piirilevyn ground -pinniin, jota merkitään piirilevyssä kirjaimin GND, ja toinen pää pinniin 3 (kuva 7).



Kuva 7 Sensorien tallennus Arduinolle

Tämä liitäntä aiheuttaa sen, että Arduino käy läpi seuraavan koodin, jossa se pyrkii nauhoittamaan sensoreita.

```
 if (digitalRead(3)==0)  //start pin 3
 {
 Serial.println("Start learning...");
 Serial.flush();
 delay(1000);
 while(micros() < 10*1000000) {
  //TXLED1;
  exists = false;
  if(NexaReceive(Sendercode,on,group,channel) == 0)
   continue;
  if(on && !group) {
   for(int i=0; i<8; i++) {
    if(knownSenders[i].sender == Sendercode && knownSenders[i].channel == channel)
{
     Serial.print("Remote already registered, "); Serial.print(i); Serial.print(": ");
     Serial.print(Sendercode); Serial.print("|"); Serial.println(channel);
     exists = true;
     break;
    }
   }
   if(!exists) {
    for(int i=0; i<8; i++) {
     if(knownSenders[i].sender == 0 && knownSenders[i].channel == 0) {
      Serial.print("Will register remote, "); Serial.print(i); Serial.print(": ");
      Serial.print(Sendercode); Serial.print("|"); Serial.println(channel);
      knownSenders[i].sender = Sendercode; knownSenders[i].channel = channel;
      Serial.println(Sendercode,HEX);
      for(int j=0; j<9; j++) {
       EEPROM.write((i*6)+j, (Sendercode >> (j*8) & 0xFF));
      }
      EEPROM.write((i*6)+5, channel);
```

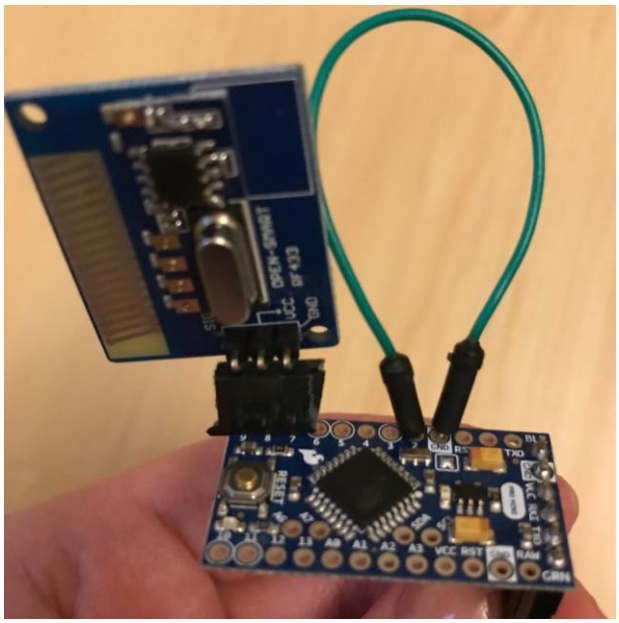
```
          break;
        }
      }
    }
  }
  if(!on && group) { //CLEAR!!
    Serial.println("Clearing stored remotes.");
    for(int i=0; i<49; i++)
      EEPROM.write(i,0);
    for(int i=0; i<8; i++) {
      knownSenders[i].sender = 0;
      knownSenders[i].channel = 0;
    }
  }
}
Serial.println("Stop learning.");

} //end pin 3
```

Käytännössä nauhoitus tapahtuu niin, että avataan Arduino IDE:n sarjamonitori, liitäntäjohto kytketään ground- ja 3 -pinniin, odotetaan sarjamonitorin näyttävän tekstiä ”Start learning”, jonka jälkeen sensoreita aktivoidaan yksi kerrallaan. Sensorit saadaan nauhoitettua Arduinon EEPROM -muistiin avaamalla ovi/ikkunasensoreita yksi kerrallaan ja heilauttamalla kättä liiketunnistimien edessä. Sarjamonitori ilmoittaa laitteen tunnistenumeron, jos tallennus on onnistunut. Tässä projektissa nauhoitetaan neljä PIR -liiketunnistinta ja neljä ikkuna/ovitunnistinta.

Ohjelmassa on seuraava koodi sen varalta, että ohjelman kanssa käytettäviin sensoreihin halutaan tehdä muutoksia.

```
if(digitalRead(2)==0)
{ //CLEAR!!
    Serial.println("Clearing stored remotes.");
    for(int i=0; i<49; i++)
      EEPROM.write(i,0);
    for(int i=0; i<8; i++)
      {
        knownSenders[i].sender = 0;
        knownSenders[i].channel = 0;
      }
}
```

Koodin avulla tyhjennetään Arduinon EEPROM -muisti ja näin ollen tiedot sensoreista. Tällöin ohjelman sensoritiedot nollaantuvat, jonka jälkeen muistiin voidaan jälleen tallentaa muita sensoreita. Muistin tyhjentäminen toimii laittamalla liitäntäjohto ground- ja 2 -pinniin (kuva 8).

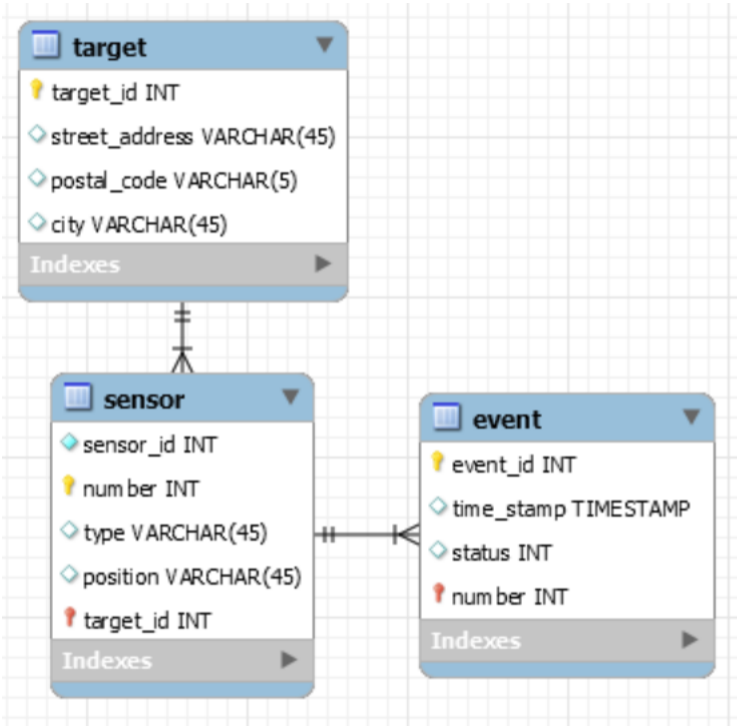


Kuva 8 Arduinon EEPROM -muistin tyhjennys

Sarjamonitori avataan ja siinä tulee lukea teksti ”Clearing stored remotes”, joka kertoo, että muisti on tyhjennetty sensorien tunnistenumeroista.

3.5 Sensoritiedon tallentaminen tietokantaan

Projektissa käytetään MySQL:ää. Raspberry Pi 3:een asennetaan MySQL Raspbian Francen How To Raspberry Pi -sivustolle kirjoittaman ohjeen mukaan. MySQL on tietokannan hallintajärjestelmä (Raspbian France 2018). Tietokannan suunnitteluvaiheessa tehdään relaatiokaavio (kuva 9). Suunnitelmassa oli aluksi erillinen osoitetaulu, jossa oli postinumero ja kaupunki. Tämä erillinen taulu oli suunnitelmassa, koska kyseessä on yhden suhde moneen -relaatio, eli yhdellä kaupungilla on monta postinumeroa, mutta yhdellä postinumerolla vain yksi kaupunki. Projektin aikana todettiin, ettei tällaisesta erillisestä taulusta ole merkittävää hyötyä, joten kaupunki -attribuutti päätettiin sijoittaa suoraan target -taulun attribuutiksi.



Kuva 9 Tietokannan relaatiokaavio

Relaatiokaaviossa tietokantataulujen lisäksi kuvataan eri taulujen relaatiot toisiinsa, sekä pää- ja viiteavaimet. Kuvassa sinisellä taustalla on tietokantataulun nimi ja sen alapuolella taulun attribuutit. Attribuutin vasemmalla puolella oleva keltainen avain kertoo attribuutin olevan taulun pääavain, punainen tarkoittaa viiteavainta. Taulujen väliset viivat kertovat taulujen relaatioista, esimerkiksi yhdellä kohteella (target) voi olla monta sensoria, mutta yhdellä sensorilla vain yksi kohde. Attribuuttien nimien jälkeen riveillä kerrotaan attribuutin tietotyyppi.

Tämän jälkeen lähdetään toteuttamaan tietokantaa Raspberry Pi 3:n komentoriviltä. MySQL:ään kirjautumisen ja oikean tietokannan valinnan jälkeen komentorivillä luodaan tietokantataulut CREATE -komennoilla (liite 3). Tämän jälkeen sensori -tauluun tallennetaan jokaisen kahdeksan sensorin numero, tyyppi ja sijainti insert -lauseella, esimerkiksi INSERT INTO sensor (number, type, position) VALUES (12345678, 'OT', 'Parveke');. Sensorin tyyppiin sijoitetaan arvo OV, eli ovi/ikkunatunnistin, tai LT, eli liiketunnistin.

Jos signaalia lähettävän laitteen tunnistenumero täsmää tallennetun sensorin tunnistenumeroa, Arduino lähettää siitä tiedon sarjaliikenteen avulla Raspberry Pi:lle, joka kirjaa tiedon MySQL -tietokantaan ja lähettää tiedon myös Internetin yli palvelimelle. Raspberry Pi:ssä tiedon kirjaamisesta tietokantaan ja lähettämisestä palvelimelle vastaa Pythonilla kirjoitettu ohjelma (liite 4), joka nimetään tässä projektissa importdata.py:ksi. Importdata.py ja muut projektin ohjelmat kirjoitetaan nano -tekstieditorilla Raspberry Pi 3:ssa.

Arduinon lähettämä data on muodossa #;tunnistenumero;ON tai OFF;X;, eli esimerkiksi #26363922;ON;X;. Importdata.py -ohjelma jakaa tiedon osiin seuraavasti.

```
data = arduino.readline()
  print data
  if data != "":
    pieces = data.split(";")
    tarkiste = pieces[0]
    if tarkiste == "#":
      laite = pieces[1]
      tila = pieces[2]
      loppu = pieces[3]
```

Tieto tulee jakaa, jotta ohjelma osaa asettaa tietokantaan oikeisiin kohtiin oikeat tiedot. Jos Arduinon rivitieto alkaa # -merkillä ja päättyy X -merkkiin, tietokantaan otetaan ohjelmassa yhteyttä, jonka jälkeen tiedot lisätään seuraavasti.

```
curs.execute('INSERT INTO event(number, status) VALUES (%s,%s)',(laite,tila))
   db.commit()
```

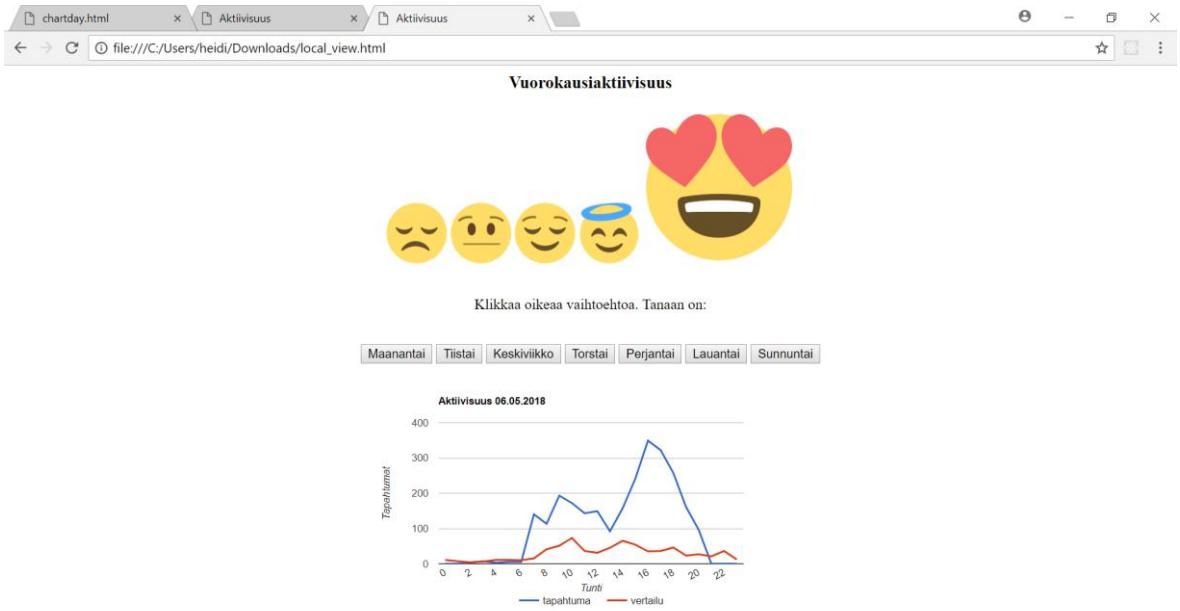
Tiedon tallentuessa tietokantaan MySQL lisää riville automaattisesti päivän ja kellonajan, jolloin tieto on tallennettu. Importdata.py:n tulee olla käynnissä koko ajan, jotta aktiivisuuden seurantajärjestelmä toimii. Tämän vuoksi luodaan seuraava ohjelma importdata.sh.

```
#!/bin/bash
while true; do
  python /home/pi/importdata.py &
  wait $!
  sleep 10
done
exit
```

Importdata.sh varmistaa importdata.py:n käynnistyksen, jos se lähtee pois päältä esimerkiksi silloin, kun Raspberry käynnistetään uudelleen. Importdata.sh tulee lisätä /etc/rc.local -tiedostoon, jotta se ajetaan esimerkiksi käynnistyksen yhteydessä.

3.6 Näkymät

Projektissa luodaan lähinäkymä, etänäkymä ja laitosnäkymä. Näkymät luodaan Python -ohjelmointikielellä. Lähinäkymä (kuva 10) on tarkoitettu henkilölle, jonka kotona sensorijärjestelmä on, etänäkymä esimerkiksi henkilön omaisille ja laitosnäkymä kotihoidolle. Lähinäkymässä näytetään viisi hymiötä, joista yksi näkyy aina suurempikokoisena. Suurempikokoinen hymiö kuvaa sitä, miten henkilön päivän aktiivisuustaso sijoittuu hänen keskiarvoonsa nähden. Hymiön alapuolella on myös pieni kaavio, jossa näytetään aktiivisuustaso.



Kuva 10 Lähinäkymä

Lähinäkymälle tehtiin hyvin yksinkertainen interaktiivinen toiminto. Toiminnossa kysytään, mikä viikonpäivä tänään on, ja pyydetään käyttäjää klikkaamaan oikeaa vaihtoehtoa kosketusnäytöltä. Ohjelma kertoo käyttäjälle, menikö vastaus oikein vai väärin. Jatkokehityksessä on aikomus antaa vastaamisesta käyttäjälle aktiivisuuspiste, joka tallennetaan tietokantaan.

Näkymiä varten luodaan kaksi uutta tietokantataulua, test ja chartday. Taulut lisätään CREATE TABLE -komennoin (liite 5). Test -taululla on attribuutit test_id, hour ja kpl, chartday -taulun attribuutit ovat chartday_id, hour, now ja comparison. Nämä taulut lisätään sen vuoksi, että saadaan aikaan kaavio, jossa on viiva, joka näyttää henkilön päivän aktiivisuusmerkintöjen määrän jokaisen tunnin aikana ja vertauksen vuoksi toinen viiva, joka näyttää henkilön keskiarvon kaikkien päivien aktiivisuusmerkinnöistä. Lähinäkymän ohjelman (liite 6) nimeksi annetaan local_view.py. Local_view.py:ssä tyhjennetään ensin test -tietokantataulu, johon kerätään päivän tapahtumat.

sql="DELETE FROM test"

Sen jälkeen test -taulun hour -attribuutille haetaan event -taulusta aikaleiman tunti, ja kpl -attribuutille tapahtumien määrä kunakin tuntina.

sql="INSERT INTO test(hour, kpl) SELECT HOUR(time_stamp) AS hour, COUNT(*)AS kpl FROM event WHERE time_stamp BETWEEN\'"+strftime("%Y-%m-%d 00:00:00", gmtime())+"\' AND \'"+strftime("%Y-%m-%d 23:59:59",gmtime())+"\' GROUP BY HOUR(time_stamp)"

Tämän jälkeen päivitetään chartday -taulun now -sarake, test -tauluun haetuilla tiedoilla.

sql="UPDATE chartday SET chartday.now = (SELECT test.kpl FROM test WHERE test.hour = chartday.hour)"

Chartday -taulun attribuutin now arvoksi annetaan 0, jos se on tyhjä, eli NULL.

sql="UPDATE chartday SET chartday.now = 0 where chartday.now is NULL"

Lopuksi haetaan päivän tapahtumat chartday -taulusta kaavion piirtämistä varten.

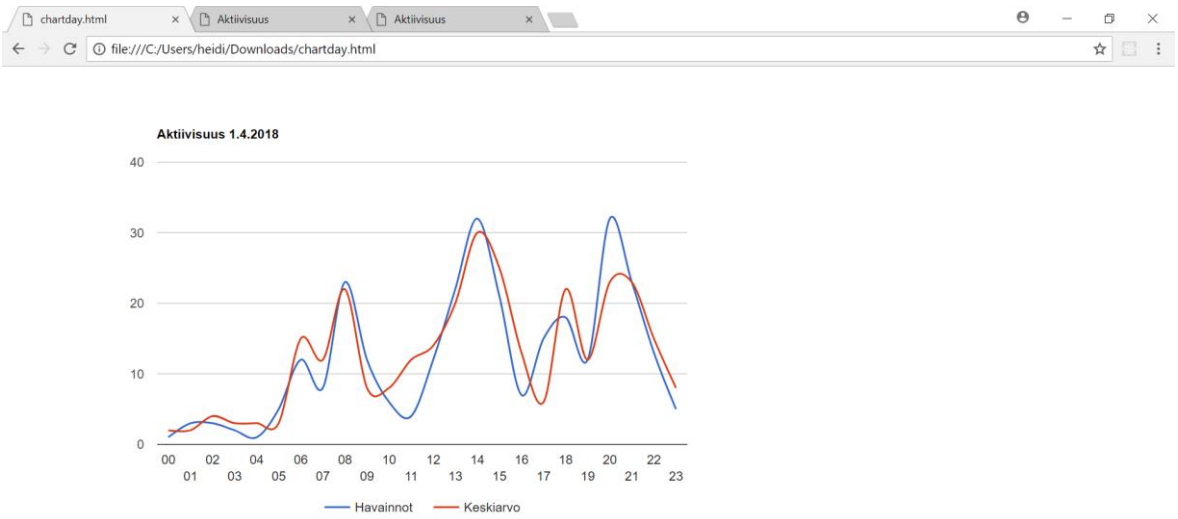
sql="SELECT hour, now, comparison FROM chartday where hour <=\'" + strftime("%H",gmtime())+"\'"

Kaavio tehdään Google Chartilla ja se asetetaan päivittymään minuutin välein, jotta yksin asuva henkilö näkee ajankohtaisen tiedon aktiivisuusmerkintöjensä määrästä. Lähinäky-mää varten luotiin local_view.sh -ohjelma.

```
#!/bin/bash
while true; do
   python /home/pi/local_view.py &
   wait $!
   sleep 10
done
exit
```

Local_view.sh /etc/rc.local -tiedostoon tallennettuna pitää huolen siitä, että local_view.py -ohjelma pysyy päällä. Ohjelmalla lähetetään kaikki tiedot myös palveluntarjoajan palvelimelle.

Etänäkymässä (kuva 11) näytetään samaa kaaviota, kuin lähinäkymässä. Kaaviota näytetään koko näytön kokoisena, sillä etänäkymässä ei ole lähinäkymän hymiöitä eikä interaktiivista toimintoa.



Kuva 11 Etänäkymä

Kaavion koodit ovat muilta osin samat kuin local_view.py:ssä. Etänäkymän python -ohjelma (liite 7) on nimeltään chartday.py. Chartdaylle tehdään myös sh -tiedosto, joka pitää py -tiedoston käynnissä.

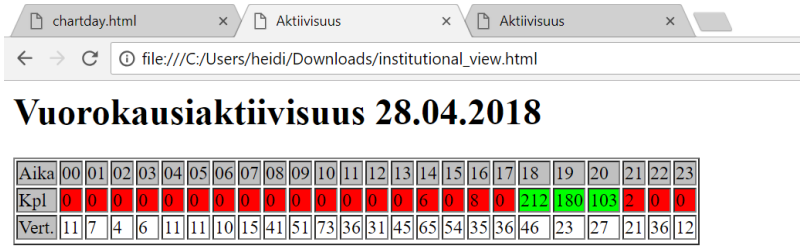
```
#!/bin/bash
while true; do
  python /home/pi/chartday.py &
  wait $!
  sleep 10
done
exit
```

Tämäkin sh -tiedosto lisätään /etc/rc.local -tiedostoon seuraavasti, jotta ohjelmat käynnistyvät heti esimerkiksi uudelleenkäynnistyksen yhteydessä.

/home/pi/importdata.sh &
/home/pi/local_view.sh &
/home/pi/institutional_view.sh &
/home/pi/chartday.sh &

Rc.local -tiedostoon kirjoitetaan siis sh -tiedoston sijainti ja & -merkki rivin loppuun.

Laitosnäkymässä (kuva 12) näytetään tieto taulukkomuotoisena. Python -ohjelman (liite 8) nimi on institutional_view.py. Taulukkoa varten tehdään samat tietokantatoiminnot, kuin edellisissä näkymissä. Käytännössä vain html -tiedosto on erilainen, kun kaavion sijaan tehdään seuraava taulukko.



Kuva 12 Laitosnäkymä

Taulukon ajatellaan olevan sopiva tiedon esitysmuoto esimerkiksi kotihoitajille, jotka värien ansiosta näkevät heti, jos aktiivisuus on ollut alhaalla liian pitkään.

4 Pohdinta

Projektin tavoitteena oli tehdä yksin kotona asuvan aktiivisuuden seurantaan sopiva sensorijärjestelmä, joka on suhteellisen edullinen, helppokäyttöinen ja mahdollisimman pitkälle olemassa olevista ratkaisuista koottu. Edullisuuteen päästiin käyttämällä vain kahdeksaa sensoria, jotka olivat suhteellisen edullisia joihinkin muihin merkkeihin ja malleihin verrattuna. Jos projektista haluaisi vielä edullisemman, esimerkiksi ovi/ikkunatunnistimia voisi käyttää vain kaksi, jolloin säästettäisiin hinnasta vielä noin 30 euroa. Raspberry Pi on yksi halvimmista tietokoneista ja myös Arduinot ovat tunnetusti edullisia. Joissakin muissa markkinoilla olevista sensorijärjestelmissä on monenlaisia sensoreita eri tarkoituksiin, jotka osaltaan tekevät ratkaisuista kalliita.

Tuotoksen helppokäyttöisyyteen päästiin sillä, että sensorijärjestelmässä käytetään langattomia sensoreita, joiden virrankulutus on hyvin vähäistä, joten paristojen vaihtoaika on joitakin vuosia. Yksin asuvan henkilön ei tarvitse tehdä muita toimenpiteitä järjestelmän kanssa, kuin pitää huoli siitä, että kosketusnäytöllinen tietokone on kiinni pistorasiassa. Koko järjestelmä on konkreettisesti vain kahdeksan sensoria ja kosketusnäyttö, sillä tietokone, Arduino ja RF -vastaanottomoduuli sijaitsevat kosketusnäytön kotelossa. Kun tietokoneen laittaa päälle, se avautuu kiosk -moodissa, joka estää laitteen käytön muuhun toimintaan, joten käyttäjällä ei käytännössä tarvitse olla minkäänlaista aiempaa kokemusta tietokoneista. Tuotokseen tarvitsee Bluetooth tai USB -näppäimistön, jotta laitteen käynnistäessä sille voidaan kirjautua sisään. Tuotoksen jatkokehityksessä laite varustetaan varavirtalaitteella, jolloin asennuksen jälkeen kirjautumista ei tarvita.

Arduinoista ja Raspberryistä voi rakentaa todella monenlaisia järjestelmiä erikseen ja yhdessä. Niitä käytetään hyvin paljon, jonka vuoksi esimerkkejä ja malliratkaisuja löytyi Internetistä runsaasti. Myös näkymissä käytetty Google Chart on laajassa käytössä, joten sen tekoon löytyi esimerkkiratkaisuja.

Projekti oli tavoitteena saada valmiiksi kymmenen viikon aikana ja sen rajauksena oli järjestelmän toiminnan testaaminen. Projektissa päästiin molempiin näistä tavoitteista. Järjestelmässä sensorien tieto kulkee Arduinoon RF -vastaanottomoduulin kautta ja siitä Raspberry Pi:hin sarjaliikenteenä, joka lisää tiedot tietokantaan ja palveluntarjoajan palvelimelle, josta tietoa voidaan näyttää näkymissä.

4.1 Projektin ajankohtaisuus, tarpeellisuus ja rajoitteet

Yksin asuvien heikkotoimintakykyisten aktiivisuuden seuranta on ajankohtainen aihe, sillä Nordcare2 -tutkimuksen mukaan Suomen hoitajien taso on matala ja kirjaamisen määrä työssä on liian korkea. Suomessa yleisesti pyritään siihen, että esimerkiksi vanhukset asuisivat mahdollisimman pitkään kotonaan. Tämä johtaa suoraan kotihoidon kysynnän kasvamiseen, jolloin kotihoitajien työ tulisi sujua mahdollisimman tehokkaasti. Kotihoito on jatkuvasti uutisissa, koska hoitajia ei ole tarpeeksi eikä heillä ole tarpeeksi aikaa kaikille asiakkaille. Mölsä (2018) kertoo, että hoivalaitokseen pääsy on vaikeaa, kun laitospaikkoja vähennetään eikä kotihoito pysty reagoimaan näihin muutoksiin nopeasti, joka johtaa yksinäisten ihmisten kotikuolemiin.

Projektin tulos on tarpeellinen, sillä kotihoidon asiakaskunta tarvitsee yksinkertaisemman, heidän toiminnastaan riippumattoman ja edullisemman ratkaisun aktiivisuuden seurantaan, kuin mitä markkinoilla on tällä hetkellä tarjolla. Turvarannekkeet eivät ole riittävä ratkaisu, sillä niiden toiminta on täysin riippuvainen rannekkeen pitäjästä. Jos henkilö esimerkiksi kaatuu, eikä hän ole muistanut laittaa rannekettaan ranteeseen, hänellä ei välttämättä ole muuta vaihtoehtoa, kuin odottaa seuraavaa kertaa, jolloin esimerkiksi kotihoito tai omaiset tulevat katsomaan häntä. Turvarannekkeet mukaan luettuna ne, joissa on aktiivisuutta seuraava sensori, auttavat käytännössä vain tilanteissa, joissa vahinko on jo sattunut. Niiden avulla ei voida päätellä esimerkiksi sitä, onko henkilön toimintakyky heikentynyt.

Aktiivisuusrannekkeiden käyttöönotto tai käyttö voi olla yksin asuvalle henkilölle liian vaikeaa, jolloin ranneke luultavasti jää käyttämättä. Joissakin aktiivisuusrannekkeissa on näkymä, joka näyttää henkilön aktiivisuustasoa sen jälkeen, kun ranneke on liitetty tietokoneeseen ja synkronoitu. Jos henkilö on antanut omaisilleen tilin käyttäjätunnuksen ja salasanan, he voivat sitä kautta seurata henkilön aktiivisuuden ja näin ollen toimintakyvyn tasoa. Näkymän toimivuus saattaa olla riippuvainen siitä, muistaako henkilö liittää rannekketta tietokoneeseen.

Näiden ajatusten pohjalta sensoriratkaisu näyttää järkevimmältä vaihtoehdolta, sillä siitä voi tehdä helposti käyttäjästä riippumattoman. Sensorijärjestelmän avulla pyritään tarkastelemaan henkilön toimintakykyä, jotta sen heikkenemistilanne pystyttäisiin huomaamaan mahdollisimman aikaisin. Toimintakyvyn heikkenemistilanteessa toimintakyky saattaa heiketä hyvin nopealla tahdilla, jonka vuoksi on erittäin tärkeää saada tilanne heti esimerkiksi kotihoidon ja omaisten tietoon, jotta voidaan tehdä tilanteeseen sopivat toimenpiteet. Järjestelmä tarvitsee myös Internet -yhteyden toimiakseen.

Projektin tuotoksesta hyötyy yksin asuva toimintakyvyltään heikko henkilö, hänen omaisensa sekä kotihoito. Toimintakyvyltään heikko, yksin asuva henkilö voi olla esimerkiksi vanhus, kehitysvammainen tai pitkäaikaissairas. Tuotos saattaa auttaa yksin asuvan henkilön aktiivisuuden motivoinnissa, sillä näkymän hymiöt kertovat siitä, kuinka hyvällä tasolla henkilön aktiivisuus on ja henkilön tulisi saada vähintään hymynaama, joka kertoo päivän aktiivisuustason olevan samalla tasolla kuin hänen keskiarvonsa. Henkilö ei kuitenkaan välttämättä välitä siitä, mikä hänen aktiivisuustasonsa on, jolloin motivointia ei tapahdu. Tällöin tuotoksen ainoa hyöty hänelle on se, että mahdollisesti kotihoito ja omaiset osaavat auttaa henkilöä tilanteessa, jossa näyttää siltä, että aktiivisuus hiipuu päivä päivältä, tai jos se on täysin nollassa.

Omaiset, esimerkiksi vanhuksen lapset, saattavat asua kaukana, eivätkä he välttämättä tiedä, mikä henkilön aktiivisuustaso on. Etänäkymällä heille voidaan tuoda mielenrauha henkilön tilasta, sillä näkymän kaaviosta voidaan nähdä, onko henkilön kotona liikuttu esimerkiksi tilanteessa, jossa omaiset eivät saa vastausta yrittäessään soittaa henkilölle. Kotihoito voi hyötyä järjestelmästä palvelutarpeen arvioinnissa sekä siinä, milloin olisi tarve tehdä uusi palvelutarpeen arviointi. Henkilön aktiivisuustason ollessa lähellä nollaa, hoitajat tietävät, että henkilön luona pitää käydä.

Henkilöstä riippumattomalla sensorijärjestelmällä on rajoituksensa. Tuotoksella ei nähdä, kuka kotona liikkuu, eli jos henkilön luona käy usein vieraita, aktiivisuustaso saattaa nousta melko jyrkästi, tai jos henkilö ei ole kotona vaan esimerkiksi käymässä kuntoutuskeskuksessa, aktiivisuustaso voi olla lähellä nollaa. Jos kotihoito seuraa henkilön tilannetta päivätasolla, olisi heillä hyvä olla tieto ainakin säännöllisistä tapahtumista, jolloin henkilö ei ole kotona. Henkilön aktiivisuutta ei seurata kodin ulkopuolella.

4.2 Jatkokehitys

Tuotoksen ylläpitoon kuuluu ohjelmiston päivittäminen sekä sensorien paristojen vaihto muutamien vuosien välein. Tuotosta voi jatkokehittää monella tavalla. Sensorien sijainti lisättiin tietokantaan projektin aikana, mutta tietoa ei hyödynnetä tuotoksessa. Sijaintia voisi hyödyntää näkymissä niin, että esimerkiksi omaiset näkevät onko ulko-ovi avattu, eli onko henkilö lähtenyt asunnosta tai onko keittiössä käyty ja jääkaapin ovea avattu.

Sensorit sijaitessa esimerkiksi eteisessä ja ulko-ovessa, voi ohjelmaa kehittää niin, että nähdään, kumpi sensoreista on antanut aktiivisuusmerkinnän ensin. Tämän tiedon avulla voidaan päätellä, onko henkilö lähtenyt asunnosta ja palannut kotiin. Tämä ominaisuus on

hyödyllinen esimerkiksi muistisairaan henkilön omaisille ja kotihoidolle. Omaisille ja kotihoidolle voidaan myös kehittää ominaisuus, jossa he saavat esimerkiksi sähköpostilla ilmoituksen siitä, jos yksin asuva henkilö ei ole saanut aktiivisuusmerkintöjä suhteellisen pitkään aikaan. Tässä tilanteessa voidaan hyödyntää edellä mainittua ominaisuutta, jossa nähdään, onko henkilö poistunut kotoaan, jonka vuoksi aktiivisuusmerkintöjä ei ole tullut.

Tällä hetkellä aktiivisuutta seurataan vain päivätasolla, joten näkymistä voi jatkokehittää myös versiot, joissa voidaan nähdä aktiivisuus viikko- ja kuukausitasolla. Näistä voi olla apua henkilön toimintakyvyn seuraamisessa, etenkin kaaviosta toimintakyvyn heikkeneminen on helppoa nähdä, jos kuukausinäkymässä kaavion aktiivisuusviiva lähtee laskuun. Projektin tuotoksesta voidaan jatkokehittää myös ylläpitonäkymä, jossa voidaan esimerkiksi hallita sensoreita.

4.3 Oppiminen ja ammatillinen kehittyminen

Projektin tekijän oppimisen kannalta projektissa haluttiin laaja yleiskatsaus siihen, mitä työnkuvaan kuuluu ja tekijälle uutta asiaa projektissa oli elektroniikasta ohjelmointiin. Opinnäytetyön tekijällä oli taustalla vain perustieto elektroniikasta, eikä aiempaa kokemusta eri laitteiden yhdistämisestä ollut. Tekijä opetteli elektroniikkapuolen asioita sitä mukaa, kun ne tulivat projektin teossa ajankohtaisiksi. Toimeksiantaja auttoi elektroniikan kokoamisessa ja sen ymmärtämisessä. Merkittävin uusi asia elektroniikkaosuudessa oli sensorien protokollan selvittäminen oskilloskoopilla.

Tekijä sai projektin johdosta melko laajan kuvan siitä, mitä työnkuvaan saattaa kuulua. Python oli ohjelmointikielenä tekijälle uusi, mutta entinen Java -ohjelmointikielen kokemus auttoi sen ymmärtämisessä ja kirjoittamisessa. Internetistä löytyi paljon esimerkkikoodia, jota muokkaamalla ja soveltamalla saatiin aikaan toimiva kokonaisuus. Python -ohjelmissa luotiin HTML -sivu, jonka teko oli tekijälle ennestään tuttua. HTML -sivun teko Python -ohjelman sisällä oli tekijälle uusi konsepti, ja se syvensi ohjelmoinnin ymmärrystä.

MySQL -tietokannan hallintajärjestelmä sekä tietokantojen luonti, muokkaus ja poisto komentoriviltä olivat ennestään tuttuja. Tietokantakyselyistä osa oli melko vaikeita verrattuna tekijän aiempaan kokemukseen ja se myös avarsi ymmärrystä siitä, miten monipuolisesti kyselyitä voi käyttää. Usean näkymän vuoksi uuden oppimiselle tuli samalla myös kertausta, kun monia samanlaisia asioita, kuten tietokantakyselyjä, tuli tehdä kolmeen eri ohjelmaan.

Projekti oli melko laaja henkilölle, jolle elektroniikka ei ole tuttua. Hyvä puoli projektissa oli ehdottomasti se, että oppii käytännössä tiedon kulun elektroniikkatasolta näkymiin saak-

ka. Aika projektin tekoon oli rajallinen, mutta projektin tavoitteisiin päästiin siitä huolimatta toimeksiantajan avulla. Opinnäytetyön raportin kirjoittamiseen kului enemmän aikaa, kuin osattiin aluksi suunnitella, ja se vaikeutti työtä, että raporttia piti kirjoittaa käytännössä samanaikaisesti tuotoksen teon kanssa. Opinnäytetyön alkuperäisessä suunnitelmassa teoriaosuuden kirjoittamiseen oli varattu vain pari ensimmäistä projektiviikkoa, mutta melko nopeasti huomattiin, ettei aika ole riittävä. Muutosehdotuksen jälkeen sovittiin ohjaajan ja toimeksiantajan kesken, että sekä tietoperustaa, että toiminnallista osuutta kirjoitetaan läpi projektin.

Etätyöskentely sujui melko hyvin. Projektin aikana pidettiin muutamia lähipäiviä toimeksiantajan kanssa, jotta pysyttiin samalla sivulla projektin teosta ja jotta ongelmatilanteissa tekijän oli mahdollista saada apua. Apua oli saatavilla myös etäpäivinä sähköpostin kautta. Yhteenvetona projektin tavoitteisiin päästiin muutaman mutkan kautta, opinnäytetyön tekijälle jäi paljon selkeämpi kuva ohjelmistokehityksestä kokonaisuutena ja myös parempi ymmärrys siitä, miten projekti kannattaa suunnitella ja kuinka paljon aikaa kuhunkin projektin osaan suurin piirtein kuluu.

Lähteet

ABB. Läsnäolo- ja liiketunnistimet. Luettavissa: http://installationsprodukter.se/documents/Esitteet/Lasnaolo_ja_liiketunnistimet_netti.pdf. Luettu: 24.3.2018.

Adafruit 2014. PIR Motion Sensor. Luettavissa: https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/how-pirs-work. Luettu: 25.3.2018.

Ahola, M. 2010. Teknologiaa vanhuksen tueksi vai kotihoitajan korvaajaksi? Yle. Luettavissa: https://yle.fi/aihe/artikkeli/2010/06/29/teknologiaa-vanhuksen-tueksi-vai-kotihoitajan-korvaajaksi. Luettu: 16.3.2018.

Chank, I. 2017. Basic Passive Infrared Motion Sensor With Arduino. Luettavissa: https://www.likecircuit.com/motion-sensor/. Luettu: 1.4.2018.

Collin, J., Saarelainen, A. 2016. Teollinen Internet, III osa, luku 9-10. Talentum. Helsinki. Luettavissa: http://ezproxy.haaga-helia.fi:2048/login?url=https://bisneskirjasto.almatalent.fi/teos/BAFBIXCTEB#. Luettu: 26.3.2018.

Helsingin Seutu 2018. Kotihoito. Luettavissa: https://www.helsinginseutu.fi/hs/selkosivut-fi/apua-arjessa/kotihoito/. Luettu: 3.4.2018.

Hughes, J. M. 2016. Arduino: A Technical Reference, luku 1-2, 4-5, 6, 9. O'Reilly Media, Inc. Sebastopol, CA, USA. Luettavissa: http://ezproxy.haaga-helia.fi:2188/9781491934319. Luettu: 25.3.2018.

HQ Designs. Motion Sensor – PIR & HF explained. Luettavissa: http://hqdesigns.de/en/interior-guide/motion-sensor/. Luettu: 25.3.2018.

Karppila, A. 2014. Arduino-pohjainen laite liikkeen ja lämpötilan monitorointiin. Amk-opinnäytetyö. Haaga-Helia ammattikorkeakoulu. Helsinki. Luettavissa: http://www.theseus.fi/bitstream/handle/10024/81790/teeseitse_monitorointilaite_lopullinen_v2.pdf?sequence=1&isAllowed=y. Luettu: 25.3.2018.

Kröger, T., Van Aerschot, L., Puthenparambil, J-M. 2018. Hoivatyö muutoksessa. Jyväskylän yliopisto. Luettavissa:

https://jyx.jyu.fi/dspace/bitstream/handle/123456789/57183/978-951-39-7372-8.pdf?sequence=1. Luettu: 18.3.2018.

Mölsä, A. 2018. Oikeusoppineet: Vanhusten lisääntynyt kotihoito on johtanut heitteillejättöihin. Yle. Luettavissa: https://yle.fi/aihe/termi/finto/httpwwwysofiontokokop18368/kotihoito. Luettu 4.5.2018.

Polar Electro 2018. Polar Loop 2 Tyylikäs aktiivisuusranneke. Luettavissa: https://www.polar.com/fi/tuotteet/lahde_liikkumaan/fitness_crosstraining/polar_loop_2. Luettu: 25.3.2018.

Raspbian France 2018. How to install a web server on the Raspberry Pi (Apache + PHP + MySQL). Luettavissa: https://howtoraspberrypi.com/how-to-install-web-server-raspberry-pi-lamp/. Luettu: 20.4.2018.

Savela, S. 2016. Enää ei passata sänkyyn – kuntoutus auttaa vanhukset takaisin kotiin. Yle. Luettavissa: https://yle.fi/uutiset/3-8876704. Luettu: 18.3.2018.

Shovic, J. 2016. Raspberry Pi IoT Projects: Prototyping Experiments for Makers, luku 1-2. Apress. Washington, USA. Luettavissa: http://ezproxy.haaga-helia.fi:2188/book/hardware/raspberry-pi/9781484213773. Luettu: 3.4.2018.

Sosiaali- ja terveysministeriö. Kotihoito ja kotipalvelut. Luettavissa: http://stm.fi/kotihoito-kotipalvelut. Luettu: 23.3.2018.

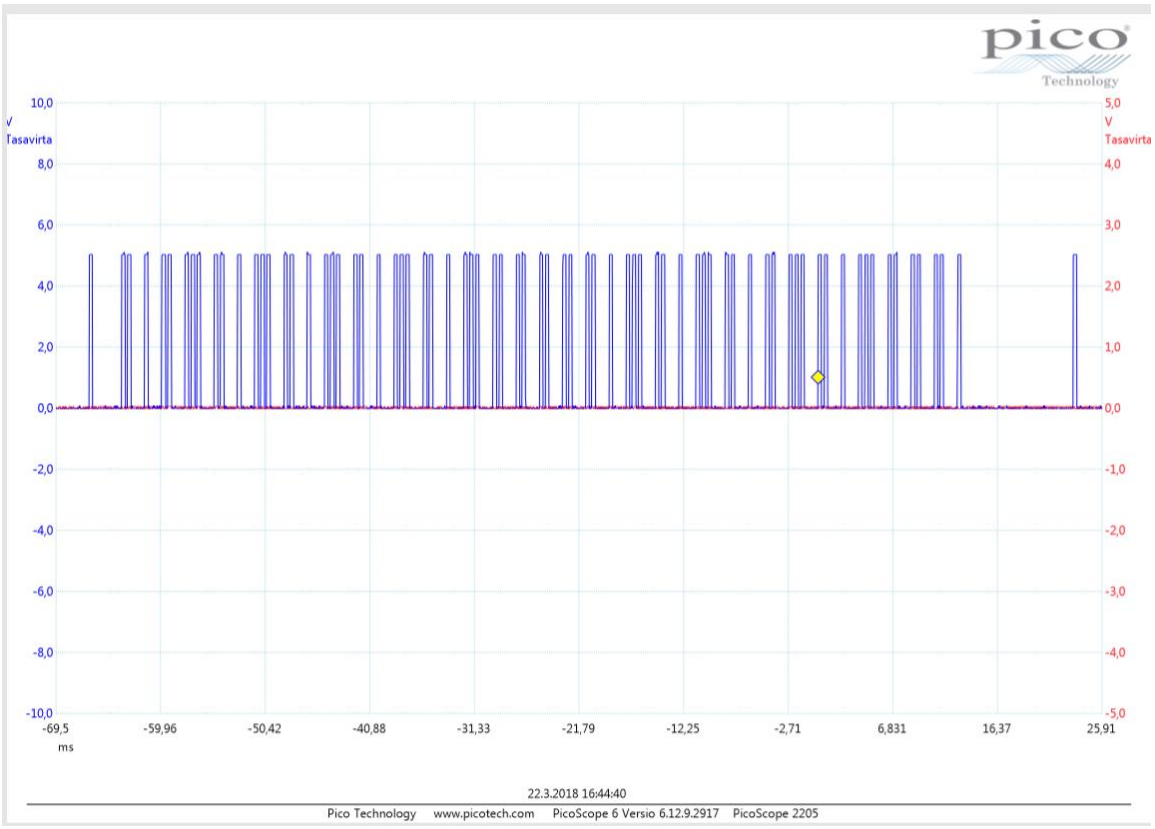
Terveyden ja hyvinvoinnin laitos 2017. Ikäihmisten kotihoidon toimintamalli ja kirjaamisen edellytykset. Luettavissa: https://thl.fi/documents/920442/3225039/iki_toimintaopas.pdf/d63cc653-fcbf-4dac-88e1-d1316beb7d03. Luettu: 23.3.2018.

Upton, E., Duntemann, J., Roberts, R., Mamtora, T. & Everard B. 2016. Learning Computer Architechture with Raspberry Pi, luku 1, 4, 12. John Wiley & Sons, Inc. Indianapolis, USA. Luettavissa: http://ezproxy.haaga-helia.fi:2188/book/hardware/raspberry-pi/9781119183938. Luettu: 3.5.2018.

YTJ. Domax Oy. Luettavissa: https://tietopalvelu.ytj.fi/yritystiedot.aspx?yavain=602934&tarkiste=7EB4FEB01A38466D52A98DE314818EE3564496DD. Luettu: 13.4.2018.

Liitteet

Liite 1. Oskilloskoopilla mitattu sensorin signaali



Liite 2. Arduinon koodi

```
/*
Nexa SelfLearning Receiver
 * Transmitter codes are hard-coded as a fixed number in the Sketch.
 * src: http://www.telldus.com/forum/viewtopic.php?f=12&t=4072
 *
 * The code has been validated to work with the following NEXA transmitters:
 *
 * WBT 912 2ch sender
 * WT2 PRO 2ch Wall sender
 * LMDT 810 Wireless Outdoor motion sensor
 * LMST 606 Wireless Magnetic contact
 * PB3 kit sender
 * Telstick Net
 *
 * Please note that timing varies some between different senders.
 * The Debug Serial port can be used to identifiy Sender codes and also for debug timing issues.
 *
 * Receiver Hardware is a 1 USD 433MHz wireless reciever module bought on Ebay. Search for "433Mhz RF transmitter and receiver arduino" to find a suitable reciever.
 *
 * Receiver functionality is based on original code from
 * Barnaby Gray 12/2008
 * Peter Mead 09/2008
 * "Homeeasy protocol receiver for the new protocol."
 *
 * * The data is encoded on the wire (aerial) as a Manchester code.
 *
 * A latch of 275us high, 2675us low is sent before the data.
 * There is a gap of 10ms between each message.
 *
 * 0 = holding the line high for 275us then low for 275us.
 * 1 = holding the line high for 275us then low for 1225us.
 *
 * The timings seem to vary quite noticably between devices.
 * If this script does not detect your signals try relaxing the timing
```

```
 * conditions.
 * *
 * Each actual bit of data is encoded as two bits on the wire as:
 * Data 0 = Wire 01
 * Data 1 = Wire 10
 *
 * The actual message is 32 bits of data (64 wire bits):
 * bits 0-25: the group code - a 26bit number assigned to controllers.
 * bit 26: group flag
 * bit 27: on/off flag
 * bits 28-31: the device code - a 4bit number.
 *
 * The group flag just seems to be a separate set of addresses you can program devices
 * to and doesn't trigger the dim cycle when sending two ONs.
 */
#include <EEPROM.h>

#define rxPin 9 // Input of 433 MHz receiver
//#define DEBUG
int bb[64];
unsigned long timer;
unsigned long sender_time[8];
int sender_limit[8]= {50,50,50,50,50,50,50,50}; //waiting time
int sender_action[8]={2,2,2,2,2,2,2,2};
int show = 0;

typedef struct {
  unsigned long sender;
  short channel;
} senderChannel;

senderChannel knownSenders[8] = {{0,0},{0,0},{0,0},{0,0},{0,0},{0,0},{0,0},{0,0}};

unsigned long Sendercode = 0; // Here is the unique Transmitter code. Use the Serial
monitor to identify your Transmitter code.

unsigned long NexaReceive(unsigned long &sender, bool &on, bool &group, short
&channel) {
```

```
  int i = 0;
  unsigned long t = 0;
  byte prevBit = 0;
  byte bit = 0;
  unsigned long recievedData = 0;
#ifdef DEBUG
  int t1 = 0; // Latch 1 time only needed for debugging purposes
  int t2 = 0; //latch 2 time only needed for debugging purposes.
#endif

  int rotations = 0;
  // latch 1
  // Latch timing has been loosened to accommodate varieties in the Nexa transmitters.
  while(t < 8000 || t > 13000) {
    t = pulseIn(rxPin, LOW, 13000);
    if(rotations++ > 10000)
      return 0;
  }

  delayMicroseconds(200);
#ifdef DEBUG
  t1 = t; // Save latch timing for debugging purposes
#endif

  rotations = 0;
  // latch 2
  // Latch timing has been loosened to accommodate varieties in the Nexa transmitters.
  while(t < 2200 || t > 2900) {
    t = pulseIn(rxPin, LOW, 2900);
    if(rotations++ > 10000)
      return 0;
  }

  delayMicroseconds(200);



#ifdef DEBUG
  t2 = t; // Save latch timing for debugging purposes
```

```
#endif

  // data collection from receiver circuit
  while (i < 64)
  {
   // do{
      t = pulseIn(rxPin, LOW, 1560);
      delayMicroseconds(200);
    if (t > 200 && t < 400){
      bit = 0;
      bb[i]=0;
    }
    else if (t > 1100 && t < 1560){
      bit = 1;
      bb[i]=1;
    }
    else
     return 0;

    if (i % 2 == 1) {
      if ((prevBit ^ bit) == 0) // must be either 01 or 10, cannot be 00 or 11
        return 0;
      recievedData <<= 1;
      recievedData |= prevBit;
#endif
    }
    prevBit = bit;
    ++i;
  }
#endif

  sender = recievedData >> 6;
  on = (recievedData >> 4) & 0x01;
  group = (recievedData >> 5) & 0x01;
  channel = (short)recievedData & 0xF;
  return recievedData;
}
```

```
void setup()
{
  timer = millis();
  //TXLED1;
  pinMode(rxPin, INPUT); // Input of 433 MHz receiver
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(2, OUTPUT); //EEPROM clearing
  pinMode(3, OUTPUT);  //Learning
  pinMode(4, OUTPUT);
  digitalWrite(7, LOW);
  digitalWrite(8, HIGH);
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(4, HIGH);
  Serial.begin(9600);

// if (digitalRead(4)==0){DEBUG=1;}
  bool on=false, group=false, exists=false;
  short channel;
  /* Read EEPROM */
#ifdef DEBUG
  delay(5000);
  for(int j=0; j<47; j++) {
    Serial.print("Reading from: "); Serial.print(j); Serial.print(" ");
    Serial.print(" 0x"); Serial.println(EEPROM.read(j),HEX);}
#endif
  for(int i=0; i<8; i++) {
    knownSenders[i].sender = ((EEPROM.read((i*6)) << 0) & 0xFF) +
((EEPROM.read((i*6)+1) << 8) & 0xFFFF) +
      ((long(EEPROM.read((i*6)+2)) << 16) & 0xFFFFFF) + ((long(EEPROM.read((i*6)+3))
<< 24) & 0xFFFFFFFF);
    knownSenders[i].channel = EEPROM.read((i*6)+5);
#ifdef DEBUG
    Serial.print("Read senderCode: "); Serial.println(knownSenders[i].sender,HEX);
    Serial.print("Read channel from: "); Serial.print((i*6)+5); Serial.print(" 0x"); Seri-
al.println(EEPROM.read((i*6)+5),HEX);
```

```
#endif
  }
  Serial.println("Nexa Receiver - SelfLearning ");
  Serial.print("Will use Sender|Code: ");// Serial.print(Sendercode); Serial.print(" "); Seri-
al.println(channel);
  //TXLED0;
  for(int i=0; i<8; i++) {
    Serial.print((unsigned long)knownSenders[i].sender);
    Serial.print("|");
    Serial.print(knownSenders[i].channel);
    Serial.print(" ");
    sender_time[i]=0;
  }
  Serial.println();
}


void loop()
{
  unsigned long sender;
  bool on, group, exists=false;
  short channel;

  delay(500);

  if(digitalRead(2)==0)
  { //CLEAR!!
      Serial.println("Clearing stored remotes.");
      for(int i=0; i<49; i++)
        EEPROM.write(i,0);
      for(int i=0; i<8; i++)
        {
          knownSenders[i].sender = 0;
          knownSenders[i].channel = 0;
        }
  }

  if (digitalRead(3)==0)  //start pin 3
```

```
 {
 Serial.println("Start learning...");
 Serial.flush();
 delay(1000);
 while(micros() < 10*1000000) {
  //TXLED1;
  exists = false;
  if(NexaReceive(Sendercode,on,group,channel) == 0)
   continue;
  if(on && !group) {
   for(int i=0; i<8; i++) {
    if(knownSenders[i].sender == Sendercode && knownSenders[i].channel == channel)
{
     Serial.print("Remote already registered, "); Serial.print(i); Serial.print(": ");
     Serial.print(Sendercode); Serial.print("|"); Serial.println(channel);
     exists = true;
     break;
    }
   }
   if(!exists) {
    for(int i=0; i<8; i++) {
     if(knownSenders[i].sender == 0 && knownSenders[i].channel == 0) {
      Serial.print("Will register remote, "); Serial.print(i); Serial.print(": ");
      Serial.print(Sendercode); Serial.print("|"); Serial.println(channel);
      knownSenders[i].sender = Sendercode; knownSenders[i].channel = channel;
      Serial.println(Sendercode,HEX);
      for(int j=0; j<9; j++) {
       EEPROM.write((i*6)+j, (Sendercode >> (j*8) & 0xFF));
      }
      EEPROM.write((i*6)+5, channel);
      break;
     }
    }
   }
  }
  if(!on && group) { //CLEAR!!
   Serial.println("Clearing stored remotes.");
   for(int i=0; i<49; i++)
```

```
      EEPROM.write(i,0);
    for(int i=0; i<8; i++) {
      knownSenders[i].sender = 0;
      knownSenders[i].channel = 0;
    }
   }
  }
  Serial.println("Stop learning.");

  } //end pin 3

  // interpret message
  if (NexaReceive(sender, on, group, channel) != 0) {
   //printResult(sender, group, on, channel); // Print the result on Serial Monitor. Use this to identify your transmitter code.
    testResult(sender, group, on, channel);
  }
}


int testResult(unsigned long sender, bool group, bool on, short channel)
{
  for(int i=0; i<8; i++) {
    if(sender == knownSenders[i].sender && group) {
      if (((millis()-sender_time[i])/1000>sender_limit[i]) || (sender_action[i]!= on))
      {
        sender_time[i]=millis();
        sender_action[i]=on;
        Serial.print("#;");
        Serial.print(sender);
        Serial.print(";");
        Serial.print(on);
        Serial.println(";X;");
      }
      return 1;
    }
    else if(sender == knownSenders[i].sender && channel == knownSenders[i].channel) {
      if (((millis()-sender_time[i])/1000>sender_limit[i]) || (sender_action[i]!= on))
```

```
      {
        sender_time[i]=millis();
        sender_action[i]=on;
        Serial.print("#;");
        Serial.print(sender);
        Serial.print(";");
        Serial.print(on);
        Serial.println(";X;");
      }
      return 1;
    }
  }
  return 0;
}

void printResult(unsigned long sender, bool group, bool on, short channel)
{
  Serial.print("## Sender: ");
  Serial.print(sender);

  group ? Serial.print(" Group Cmd ") : Serial.print("");
  on ? Serial.print(" ON") :  Serial.print(" OFF");

  Serial.print(" Channel: ");
  Serial.println(channel);
}
```

Liite 3. Tietokannan taulujen CREATE -komennot

```
CREATE TABLE target (
target_id INT NOT NULL AUTO_INCREMENT,
street_address VARCHAR(45) NOT NULL,
postal_code VARCHAR(5) NOT NULL,
city VARCHAR(45) NOT NULL,
PRIMARY KEY (target_id)
);

CREATE TABLE sensor (
sensor_id INT NOT NULL AUTO_INCREMENT,
number INT NOT NULL,
type VARCHAR(45),
position VARCHAR(45),
target_id INT,
PRIMARY KEY (sensor_id),
FOREIGN KEY (target_id) REFERENCES target (target_id)
);

CREATE TABLE event (
event_id INT NOT NULL AUTO_INCREMENT,
time_stamp TIMESTAMP NOT NULL,
status INT NOT NULL,
number INT NOT NULL,
PRIMARY KEY (event_id)
);
```

Liite 4. Python -ohjelma importdata.py

```
import serial
import time
import MySQLdb
import urllib2
import requests
arduino = serial.Serial('/dev/ttyUSB0', baudrate=9600)

while 1:
  time.sleep(5)
  data = arduino.readline()
  print data
  if data != "":
    pieces = data.split(";")
    tarkiste = pieces[0]
    if tarkiste == "#":
      laite = pieces[1]
      tila = pieces[2]
      loppu = pieces[3]

      if loppu == "X":
        db = MySQLdb.connect("localhost","kayttaja","salasana","tietokanta")
        curs=db.cursor()
        try:
          curs.execute('INSERT INTO event(number, status) VALUES (%s,%s)',(laite,tila))
          db.commit()
        except:
          db.rollback()
        r = requests.post("[redacted]")
        print(r.status_code)

      if loppu == "A":
        db = MySQLdb.connect("localhost","kayttaja","salasana","tietokanta")
        curs=db.cursor()
        try:
          curs.execute('INSERT INTO sensor(number, type, position) VALUES
(%s,%s,%s)',(laite, tila, sijainti))
```

```
    db.commit()
            except:
    db.rollback()
    r = requests.post("[redacted]")
    print(r.status_code)

db.close()
```

Liite 5. Chartday ja test -tietokantataulujen CREATE TABLE -komennot

```
CREATE TABLE chartday (
chartday_id INT NOT NULL AUTO_INCREMENT,
hour VARCHAR(2),
now INT,
comparison INT,
PRIMARY KEY (chartday_id)
);

CREATE TABLE test (
test_id INT NOT NULL AUTO_INCREMENT,
hour VARCHAR(2),
kpl INT,
PRIMARY KEY (test_id)
);
```

Liite 6. Lähinäkymän local_view.py -ohjelma

```
#!/usr/bin/env python
import os
import time
import MySQLdb
import datetime
from time import gmtime, strftime
while 1:
  print "alkaa"
  db = MySQLdb.connect("localhost","kayttaja","salasana","tietokanta")
  cursor=db.cursor()

#tyhjaa testi nimisen taulun johon keraa paivan tapahtumat
  sql="DELETE FROM test"
  number_of_rows = cursor.execute(sql)
  db.commit()

#hakee paivan tapahtumat ja vie ne testi tauluun
  sql="INSERT INTO test(hour, kpl) SELECT HOUR(time_stamp) AS hour, COUNT(*) AS
kpl FROM event WHERE time_stamp BETWEEN\'"+strftime("%Y-%m-%d 00:00:00",
,gmtime())+"\' AND \'"+strftime("%Y-%m-%d 23:59:59",gmtime())+"\' GROUP BY
HOUR(time_stamp)"
  number_of_rows = cursor.execute(sql)
  db.commit()

#paivittaa chartday taulun now sarakkeeseen testitauluun haetuilla paivan tapahtumilla
  sql="UPDATE chartday SET chartday.now = (SELECT test.kpl FROM test WHERE
test.hour = chartday.hour)"
  number_of_rows = cursor.execute(sql)
  db.commit()

#paivittaa chartdayn now -sarakkeelle arvon 0 missa arvo on null eli tyhja
  sql="UPDATE chartday SET chartday.now = 0 where chartday.now is NULL"
  number_of_rows = cursor.execute(sql)
  db.commit()

#hakee paivan tapahtumat chartin piirtamista varten
```

```
  sql="SELECT hour, now, comparison FROM chartday where hour <=\'" +
strftime("%H",gmtime())+"\'"
  number_of_rows = cursor.execute(sql)
  db.commit()

  f = open('/var/www/html/local_view.html','w')
  f.write('<html>')
#tassa maaritellaan etta chart ladataan aina 60 sekunnin jalkeen uudeleen
  f.write('<head><META HTTP-EQUIV="refresh" CONTENT="60">')
  f.write('<title>Aktiivisuus</title>')
#tassa kolmerivia joilla pakotetaan chart hakemaan aina uuden tiedoston muuten nayttaa
vanhaa muistista
  f.write('<meta http-equiv="Cache-Control" content="no-cache, no-store, must-
revalidate">')
  f.write('<meta http-equiv="Pragma" content="no-cache">')
  f.write('<meta http-equiv="Expires" content="0">')
  f.write('  <script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>')
  f.write('  <script type="text/javascript">')
  f.write('    google.charts.load(\'current\', {packages:[\'corechart\',\'line\']});')
  f.write('</script>')
  f.write('</head>')
  f.write('<body>')
  f.write('<center><H3>Vuorokausiaktiivisuus</h3>')
  row = cursor.fetchone()

#lasketaan kaikki paivan tapahtumat kyseisena ajankohtana yhteen ja verrataan niita ver-
tailuarvoista yht laskettuun tietoon
  sql="SELECT SUM(now) AS sum_now, SUM(comparison) AS sum_comparison FROM
chartday WHERE hour <=\'"+strftime("%H",gmtime())+"\'"
  number_of_rows = cursor.execute(sql)
  db.commit()
#haetaan kaikki tapahtumat mita siihen kellon lyomaan (hour) mennessa on tapahtunut
  sql="SELECT hour,now,comparison FROM chartday WHERE hour
<=\'"+strftime("%H",gmtime())+"\'"
  number_of_rows = cursor.execute(sql)
  db.commit()
  now = 0
```

```
  comparison = 0
  row2save = 0
  while True:
   row = cursor.fetchone()
   if row == None:
     break
   now=now+row[1]
   comparison=comparison+row[2]
   row2save = row[2]

#   print row2save
   comparison = comparison-row2save
#  db.close()
  print "----"
  image1 = "emoji1.png"
  image2 = "emoji2.png"
  image3 = "emoji3.png"
  image4 = "emoji4.png"
  image5 = "emoji5.png"

  if now < (comparison * 0.5):
     f.write('<img src="'+image1+'" width=170 height=170>''<img src="'+image2+'" width=70
height=70>''<img src="'+image3+'" width=70 height=70>''<img src="'$
  elif now < comparison and now > (comparison * 0.5):
     f.write('<img src="'+image1+'" width=70 height=70>''<img src="'+image2+'" width=170
height=170>''<img src="'+image3+'" width=70 height=70>''<img src="'$
  elif now < (comparison * 1.1)  and now > comparison :
     f.write('<img src="'+image1+'" width=70 height=70>''<img src="'+image2+'" width=70
height=70>''<img src="'+image3+'" width=170 height=170>''<img src="'$
  elif now < (comparison * 120 )and now > (comparison * 1.1):
     f.write('<img src="'+image1+'" width=70 height=70>''<img src="'+image2+'" width=70
height=70>''<img src="'+image3+'" width=70 height=70>''<img src="'+i$
  elif now > (comparison * 120):
     f.write('<img src="'+image1+'" width=70 height=70>''<img src="'+image2+'" width=70
height=70>''<img src="'+image3+'" width=70 height=70>''<img src="'+i$
  db.close()
#interaktiivinen toiminto
  db = MySQLdb.connect("localhost","kayttaja","salasana","tietokanta")
```

```
  cursor=db.cursor()
#tarkistaa monesko paiva nyt on ma=0...su=6
  monesko = datetime.datetime.today().weekday()
  f.write('<p><br/>Klikkaa oikeaa vaihtoehtoa. Tanaan on: </p><br/> ')
  if monesko == 0:
    f.write('<button onclick="myFunction()">Maanantai</button> ')
  else:
    f.write('<button onclick="myFunction1()">Maanantai</button> ')
  if monesko == 1:
    f.write('<button onclick="myFunction()">Tiistai</button> ')
  else:
    f.write('<button onclick="myFunction1()">Tiistai</button> ')
  if monesko == 2:
    f.write('<button onclick="myFunction()">Keskiviikko</button> ')
  else:
    f.write('<button onclick="myFunction1()">Keskiviikko</button> ')
  if monesko == 3:
    f.write('<button onclick="myFunction()">Torstai</button> ')
  else:
    f.write('<button onclick="myFunction1()">Torstai</button> ')
  if monesko == 4:
  f.write('<button onclick="myFunction()">Perjantai</button> ')
  else:
    f.write('<button onclick="myFunction1()">Perjantai</button> ')
  if monesko == 5:
    f.write('<button onclick="myFunction()">Lauantai</button> ')
  else:
    f.write('<button onclick="myFunction1()">Lauantai</button> ')
  if monesko == 6:
    f.write('<button onclick="myFunction()">Sunnuntai</button> ')
  else:
    f.write('<button onclick="myFunction1()">Sunnuntai</button> ')
  f.write('<p id="kysely"></p>')
  f.write('<script>')
  f.write(' function myFunction(){')
  f.write('document.getElementById("kysely").innerHTML = "Oikein";')
  f.write('}')
  f.write('function myFunction1(){')
```

```
  f.write('  document.getElementById("kysely").innerHTML = "Melkein oikein";')
  f.write('}')
  f.write('</script>')
  db.close()

  db = MySQLdb.connect("localhost","kayttaja","salasana","tietokanta")
  cursor=db.cursor()

#tyhjaa test -taulun johon keraa paivan tapahtumat
  sql="DELETE FROM test"
  number_of_rows = cursor.execute(sql)
  db.commit()

#hakee paivan tapahtumat ja vie ne test -tauluun
  sql="INSERT INTO test(hour, kpl) SELECT HOUR(time_stamp) AS hour, COUNT(*) AS
kpl FROM event WHERE time_stamp BETWEEN\'"+strftime("%Y-%m-%d 00:00:00”
,gmtime())+"\' AND \'"+strftime("%Y-%m-%d 23:59:59",gmtime())+"\' GROUP BY
HOUR(time_stamp)"
  number_of_rows = cursor.execute(sql)
  db.commit()

#paivittaa chartday taulun now sarakkeeseen testitauluun haetuilla paivan tapahtumilla
  sql="UPDATE chartday SET chartday.now = (SELECT test.kpl FROM test WHERE
test.hour = chartday.hour)"
  number_of_rows = cursor.execute(sql)
  db.commit()

#tyhjentaa chartday taulusta paivan tapahtumat eli sarakkeen now
  sql="UPDATE chartday SET chartday.now = 0 where chartday.now is NULL"
  number_of_rows = cursor.execute(sql)
  db.commit()

#hakee paivan tapahtumat chartin piirtamista varten
  sql="SELECT hour, now, comparison FROM chartday"
  number_of_rows = cursor.execute(sql)
  db.commit()

#google chart
```

```
  f.write('<div id="container" style="width: 500px; height: 250px; margin: 0 auto"></div>')
  f.write('<script language="JavaScript">')
  f.write('    function drawChart() {')
  f.write('      var data = new google.visualization.DataTable();')
  f.write('      data.addColumn(\'string\', \'tunti\');')
  f.write('      data.addColumn(\'number\', \'tapahtuma\');')
  f.write('      data.addColumn(\'number\', \'vertailu\');')
  f.write('      data.addRows([')
  row = cursor.fetchone()
  f.write('[\'')
  f.write(str(row[0]))
  f.write('\',')
  f.write(str(row[1]))
  f.write(',')
  f.write(str(row[2]))
  while True:
    row = cursor.fetchone()
    if row == None:
      break
    print(row[0])
    print(row[1])
    print(row[2])
    f.write('],')
    f.write('[\'')
f.write(str(row[0]))
    f.write('\',')
    f.write(str(row[1]))
    f.write(',')
    f.write(str(row[2]))

  f.write(']')
  db.close()
  f.write('        ]);')
  f.write('    var options = {')
  f.write('        \'title\': \'Aktiivisuus '+strftime("%d.%m.%Y",gmtime())+'\',')
  f.write('        hAxis: {title: \'Tunti\',},')
  f.write('        vAxis: {title: \'Tapahtumat\',},')
  f.write('        \'width\':500,')
```

```
  f.write('        \'height\':250,')
  f.write('        crosshair: {')
  f.write('          color: \'#000\',')
  f.write('          trigger: \'selection\'},')
  f.write('        legend: { position: \'bottom\' }')
  f.write('      };')
  f.write('    var chart = new
google.visualization.LineChart(document.getElementById(\'container\'));')
  f.write('    chart.draw(data, options);')
  f.write('    }')
  f.write('  google.charts.setOnLoadCallback(drawChart);')
  f.write('  </script>')
  f.write('</center>')
  f.write('</body>')
  f.write('</html>')
  f.close()
  time.sleep(20)
```

Liite 7. Etänäkymän chartday.py -ohjelma

```
#!/usr/bin/python
import os
import time
import MySQLdb
from time import gmtime, strftime
while 1:
  print "alkaa"
  db = MySQLdb.connect("localhost","kayttaja","salasana","tietokanta")
  cursor=db.cursor()

#tyhjaa testi nimisen taulun johon keraa paivan tapahtumat
  sql="DELETE FROM test"
  number_of_rows = cursor.execute(sql)
  db.commit()

#hakee paivan tapahtumat ja vie ne testi tauluun
  sql="INSERT INTO test(hour, kpl) SELECT HOUR(time_stamp) AS hour, COUNT(*) AS
kpl FROM event WHERE time_stamp BETWEEN\'"+strftime("%Y-%m-%d 00:00:00",
gmtime())+"\' AND \'"+strftime("%Y-%m-%d 23:59:59",gmtime())+"\' GROUP BY
HOUR(time_stamp)"
  number_of_rows = cursor.execute(sql)
  db.commit()

#paivittaa chartday taulun now sarakkeeseen testitauluun haetuilla paivan tapahtumilla
  sql="UPDATE chartday SET chartday.now = (SELECT test.kpl FROM test WHERE
test.hour = chartday.hour)"
  number_of_rows = cursor.execute(sql)
  db.commit()

#paivittaa chartday -taulun now -sarakkeelle arvon 0 missa arvo on tyhja eli null
  sql="UPDATE chartday SET chartday.now = 0 where chartday.now is NULL"
  number_of_rows = cursor.execute(sql)
  db.commit()
```

```
#hakee paivan tapahtumat chartin piirtamista varten
  sql="SELECT hour, now, comparison FROM chartday"
  number_of_rows = cursor.execute(sql)
  f = open('/var/www/html/chartday.html','w')
  f.write('<html>')

#tassa maaritellaan etta chart ladataan aina 60 sekunnin jalkeen uudelleen
  f.write('<head><META HTTP-EQUIV="refresh" CONTENT="60">')
  f.write(‘<title>Aktiivisuus</title>’)

#pakotetaan chart hakemaan aina uuden tiedoston ettei nayteta vanhaa muistista
  f.write('<meta http-equiv="Cache-Control" content="no-cache, no-store, must-
revalidate">')
  f.write('<meta http-equiv="Pragma" content="no-cache">')
  f.write('<meta http-equiv="Expires" content="0">')

#google chart
  f.write('<script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>')
  f.write('<script type="text/javascript">')
  f.write('google.charts.load(\'current\', {\'packages\':[\'corechart\']});')
  f.write('google.charts.setOnLoadCallback(drawChart);')
  f.write('function drawChart() {')
  f.write('var data = google.visualization.arrayToDataTable([')
  f.write('[\'Hour\', \'Havainnot\',\'Vertailu\'],')
  row = cursor.fetchone()
  f.write('[\'')
  f.write(str(row[0]))
  f.write('\',')
  f.write(str(row[1]))
f.write(',')
  f.write(str(row[2]))
  while True:
     row = cursor.fetchone()
     if row == None:
        break
     print(row[0])
     print(row[1])
```

```
        print(row[2])
        f.write('],')
        f.write('[\'')
        f.write(str(row[0]))
        f.write('\',')
        f.write(str(row[1]))
        f.write(',')
        f.write(str(row[2]))

    while True:
        row = cursor.fetchone()
        if row == None:
            break
        print(row[0])
        print(row[1])
        print(row[2])
        f.write('],')
        f.write('[\'')
        f.write(str(row[0]))
        f.write('\',')
        f.write(str(row[1]))
        f.write(',')
        f.write(str(row[2]))
    f.write(']')
    db.close()
    f.write(']);')
    f.write('var options = {')
    f.write('\'title\': \'Aktiivisuus '+strftime("%d.%m.%Y",gmtime())+'\',')
    f.write('hAxis: {title: \'Tunti\',},')
    f.write('vAxis: {title: \'Tapahtumat\',},')
    f.write('\'width\':900,')
    f.write('\'height\':500,')
    f.write('crosshair: {')
    f.write('color: \'#000\',')
    f.write('trigger: \'selection\'},')
    f.write('legend: { position: \'bottom\' }')
    f.write('};')
    f.write('var chart = new google.visualization.LineChart(document.getElemen$
```

```
f.write('chart.draw(data, options);')
f.write('}')
f.write('google.charts.setOnLoadCallback(drawChart);')
f.write('</script>')
f.write('</body>')
f.write('</html>')
f.close()
time.sleep(20)
```

Liite 8. Laitosnäkymän institutional_view.py -ohjelma

```
#!/usr/bin/env python
import os
import time
from datetime import datetime, timedelta
import MySQLdb
from time import gmtime, strftime
while 1:
  db = MySQLdb.connect("localhost","kayttaja","salasana","tietokanta")
  cursor=db.cursor()

#tyhjaa testi nimisen taulun johon keraa paivan tapahtumat
  sql="DELETE FROM test"
  number_of_rows = cursor.execute(sql)
  db.commit()

#hakee paivan tapahtumat ja vie ne testi tauluun
  sql="INSERT INTO test(hour, kpl) SELECT HOUR(time_stamp) AS hour, COUNT(*) AS
kpl FROM event WHERE time_stamp BETWEEN\'"+strftime("%Y-%m-%d
00:00:00",gmtime())+"\' AND \'"+strftime("%Y-%m-%d 23:59:59",gmtime())+"\' GROUP BY
HOUR(time_stamp)"
  number_of_rows = cursor.execute(sql)
  db.commit()

#paivittaa chartday taulun now sarakkeeseen testitauluun haetuilla paivan tapahtumilla
  sql="UPDATE chartday SET chartday.now = (SELECT test.kpl FROM test WHERE
test.hour = chartday.hour)"
  number_of_rows = cursor.execute(sql)
  db.commit()

#paivittaa chartday -taulun now  -sarakkeelle nollan, missa arvo on tyhja eli null
  sql="UPDATE chartday SET chartday.now = 0 where chartday.now is NULL"
  number_of_rows = cursor.execute(sql)
  db.commit()

#hakee paivan tapahtumat chartin piirtamista varten
  sql="SELECT hour, now, comparison FROM chartday"
```

```
  number_of_rows = cursor.execute(sql)
  db.commit()

  f = open('/var/www/html/institutional_view.html','w')
  f.write('<html>')
#tassa maaritellaan etta chart ladataan aina 60 sekunnin jalkeen uudeleen
  f.write('<head><META HTTP-EQUIV="refresh" CONTENT="60">')
  f.write('<title>Aktiivisuus</title>')
#pakotetaan chart hakemaan aina uuden tiedoston muuten nayttaa vanhaa muistista
  f.write('<meta http-equiv="Cache-Control" content="no-cache, no-store, must-
revalidate">')
  f.write('<meta http-equiv="Pragma" content="no-cache">')
  f.write('<meta http-equiv="Expires" content="0">')
  f.write('</head>')
  f.write('<body>')
  f.write('<H1>Vuorokausiaktiivisuus '+strftime("%d.%m.%Y",gmtime())+'</h1>')
  f.write('<table border = "1"><tr><td bgcolor="#C0C0C0">Aika</td>')
  while True:
    row = cursor.fetchone()
    if row == None:
      break
    f.write('<td bgcolor="#C0C0C0">'+str(row[0])+'</td>')
  f.write('</tr><tr><td bgcolor="#C0C0C0">Kpl</td>')

  sql="SELECT hour, now, comparison FROM chartday"
  number_of_rows = cursor.execute(sql)
db.commit()

  while True:
    row = cursor.fetchone()
    if row == None:
      break
    if int(row[1])<int(row[2]):
      f.write('<td bgcolor="#FF000">'+str(row[1])+'</td>')
    else:
      f.write('<td bgcolor="#00FF00">'+str(row[1])+'</td>')

  f.write('</tr><tr><td bgcolor="#C0C0C0">Vert.</>')
```

```
sql="SELECT hour, now, comparison FROM chartday"
number_of_rows = cursor.execute(sql)
db.commit()

while True:
   row = cursor.fetchone()
   if row == None:
 break
   f.write('<td>'+str(row[2])+'</td>')
f.write('</tr><tr>')
f.write('</table>')

db.close()

f.write('</body>')
f.write('</html>')
f.close()
time.sleep(20)
```