KANDIDAATINTYÖ

Jori-Pekka Rautava

SÄHKÖTEKNIIKAN TUTKINO-OHJELMA
2017

KANDIDAATINTYÖ

CNC-jyrsimen tärinän mittaus värinäsensorilla

Jori-Pekka Rautava

Ohjaaja: Jari Marjakangas

SÄHKÖTEKNIIKAN TUTKINTO-OHJELMA
2017

Rautava J-P. (2017) CNC-jyrsimen tärinän mittaus värinäsensorilla. Oulun yliopisto, sähkötekniikan koulutusohjelma. Kandidaatintyö 34 s.

TIIVISTELMÄ

Tässä työssä tehdään ensin kirjallisuuskatsaus LTE-tekniikan perusteisiin sekä 5G-tekniikan käyttöön ja mahdollisuuksiin IoT-sovellutuksissa. Teoriaosuudessa tarkastellaan mistä osista LTE-tekniikka koostuu ja mitkä ovat keskeisiä asioita LTE-tekniikan mahdollistamassa suuremmassa datan siirtonopeudessa verrattuna 3G tekniikkaan. 5G-tekniikkaan liittyen esitellään 5G:n perusideat ja miksi se mahdollistaa entistä suuremmat datan siirtonopeudet ja pienemmät latenssit kuin 4G-tekniikka.

Tämän työn lopussa on esitelty tekemäni IoT-sovellus, jossa CNC-jyrsimestä voidaan mitata tärinäarvoja käyttäen ADIS16227 värinäsensoria, joka on liitetty Raspberry Pi –tietokoneeseen. Lopuksi pohditaan tämän sovellutuksen rajoitteita ja käyttökohteita.

Asiasanat: Internet of Things, 5GTN, Raspberry Pi, värinäsensori

Rautava J-P. (2017) Vibration measurement of CNC milling maching with vibration sensor. University of Oulu, Degree Programme in Electrical Engineering. Bachelor's Thesis, 34 p.

ABSTRACT

In the beginning of this thesis I make a literature overview of the basics of LTE technology and the possibilities of 5G technology and IoT applications. In the theory part I review what makes LTE technology and what are the key principles that allow greater data rates compared to 3G. The very basics of 5G technology is introduced. I also review why 5G technology allows greater data rates and lower latency compared to 4G technology.

In the end of this thesis I present IoT application I have made that allows to examine vibration from CNC milling machine. For measuring the vibration ADIS16227 vibration sensor is connected with Raspberry Pi computer. Finally, I consider what can be done with this IoT application and what constraints it may have.

Key words: Internet of Things, 5GTN, Raspberry Pi, vibration sensor

SISÄLLYS

LYHENTEIDEN JA MERKKIEN SELITYKSET

3GPP	Third generation partnership project
5GTN	5th Generation Test Network
CE	Chip Enable
CNC	Computer Numerical Control
CWC	Center for Wireless Communications
DDoS	Distributed Denial of Service
FDD	Frequency Division Duplexing
GPIO	General Purpose Input/Output
GSM	Global System for Mobile communications
HSPA	High Speed Packet Access
IoT	Internet of Things
ISM	Industrial, Scientific and Medical, radio bands
LoRa	Long Range, long range and low power wireless technology
LTE	Long Term Evolution
LPWAN	Low-Power Wide-Area Network
MEC	Mobile-Edge Computing
MIMO	Multiple-Input-Multiple-Output
MISO	Multiple-Input-Single-Output
NB-IoT	Narrowband Internet of Things
OFDM	Orthogonal Frequency-Division Multiplexing
RAT	Radio Access Technology
RF	Radio Frequency
SISO	Single-Input-Single-Output
SW-SIM	Software-Subscriber Identity Module
TDD	Time Division Duplexing
VPN	Virtual Private Network
WAN	Wide Area Network
WLAN	Wireless Local Area Network

kbps	Kilobittiä sekunnissa
Gbps	Gigabittiä sekunnissa
Mbps	Megabittiä sekunnissa
Tbps	Terabittiä sekunnissa
MHz	Megahertsi
MB	Megatavu
TB	Teratavu
mW	Milliwatti
mA	Milliampeeri
km/h	Kilometriä tunnissa

1. JOHDANTO

Tässä työssä on tarkoituksena tutkia ja kehittää sensorimittausmenetelmä Oulun yliopiston FabLabilla sijaitsevan CNC-jyrsimen (Computer Numerical Control) tärinän mittaamiseksi sensorin avulla. Tutkimuksen tarkoituksena on saada sensorilta mittausdataa laitteen toiminnasta laitteen tärinän avulla ja korjata virheelliset käyttöasetukset tai kuluneet komponentit ennen laitteen vioittumista. Raspberry Pi-tietokoneella luetaan sensorin mittausdata, joka siirretään WLAN-yhteydellä langattomasti 5G-testiverkkoa käyttäen FabLabin Azure-pilvipalveluun, jossa tehdään analyysi saadusta mittausdatasta. Azure on Microsoftin tarjoama pilvipalvelu, jota voidaan käyttää alustana erilaisille langattomuutta vaativille sovelluksille. Tämän työn teknisessä toteutuksessa on keskitytty vain sensorin mittausdatan keräämiseen, datan siirto 5G-testiverkkoa käyttäen ei kuulu tähän työhön.

Tämän kandidaatintyön tarkoitus on pidentää FabLabilla olevien laitteiden käyttöikää, koska tässä työssä tehtävä sensorin mittausmenetelmä voidaan toistaa myös muissa laitteissa parametreja muuttamalla sensorilta saaduista mittausarvoista riippuen ja tällä tavoin saadaan kaikki FabLabin laitteet sensorimittauksen piiriin pienellä vaivalla.

Teorian osuus käsittelee enemmän LTE-tekniikkaa (Long Term Evolution) ja 5G-tekniikaa, johon siirrytään lähitulevaisuudessa. Aluksi tässä työssä oli tarkoitus käsitellä myös sensorin liittämistä 5G-testiverkkoon, mutta se olisi laajentanut työn kokonaisuutta liikaa suhteutettuna kandidaatintyön tarkoitukseen.

2. VERKKOTEKNIIKOIDEN TEORIAA

2.1. 5GTN

5GTN (5th generation test network) on Oulun yliopistolla CWC:n (Centre for Wireless Communications) ylläpitämä testiverkko 5G-teknologioiden kehitykseen. Testiverkko tukee 5G-laitteiden käyttöä, ylempiä taajuusalueita sekä kognitiivisia hallintatyökaluja ja testaustyökaluja uusille ratkaisuille. Testiverkossa on mahdollista testata pienten solujen vaikutusta tehokkuuteen ja toteuttaa sitä kautta erilaisia IoT-sovelluksia (Internet of Things, esineiden Internet) [1].

5G-verkon kantavana ideana oleva solukokojen pieneminen voi avata uusia mahdollisuuksia mikro-operaattoreille. Solukoot voivat pienentyä, jopa vain kerrostalon kokoisiksi. Mikro-operaattorit voisivat huolehtia pienten solujen tukiasemien ylläpidosta ja asennuksista.

5GTN on jaettu kahteen osaan: VTT:llä on suljettu verkko, jossa yritykset voivat testata teknologioidensa toimivuutta, työkaluja ja sovelluksia kontrolloidussa ympäristössä. Avoimessa verkossa Oulun yliopistolla voidaan tutkia ratkaisuja esimerkiksi tilanteisiin, joissa on lukuisia päätelaitteita. Verkko on täysin toimiva operaattoriverkko, joten sen tarkoituksena on varmistaa pieniin soluihin pohjautuvien, niin kutsuttujen mikro-operaattoreiden, toimintamalleja oikeassa toimintaympäristössä. Tulevaisuudessa testiverkko tulee kattamaan eri alueita Oulussa. Testiverkossa on myös mahdollista testata ohjelmia, palveluja ja työkaluja oikeassa 5G-verkossa, joten myös vaativia IoT ratkaisuja on mahdollista testata tässä ympäristössä [1].

5G-testiverkon kehittämisessä on tällä hetkellä mukana jo lähes 30 eri yritystä, joten kehitys on laaja-alaista tietoliikennetekniikan kaikilla osa-alueilla. 5G testiverkossa pystytään myös testaamaan reunalaskentatekniikkaa (MEC, Mobile-Edge Computing), joka tarkoittaa, että dataa käsitellään heti linkkitornin läheisyydessä, sen sijaan että se lähetettäisiin ensin serverille ja sen jälkeen serveriltä takaisin käyttäjälle. Tällä estetään latenssin kasvaminen ja verkon turha kuormittuminen edes takaisessa liikenteessä.

2.2. LTE tekniikka

LTE (Long Term Evolution) on vielä melko nuori verkkoteknologia, mutta se luo vahvan perustan 5G-tekniikan kehittämiselle. Isoimpana mahdollistajana tähänastisessa LTE:n kehityksessä ovat olleet uudet radioliityntätekniikat (RAT, Radio Access Technology), näitä liityntäteknologioita ovat muun muassa LTE, WLAN (Wireless Local Area Network) sekä Bluetooth, WLAN-tekniikasta käytetään myös yleisesti nimitystä Wi-Fi, joka on amerikkalaisten käyttämä nimi samasta tekniikasta. LTE esiteltiin 2009 ja sen jälkeen se on kehittynyt valtavasti muun muassa tiedonsiirtonopeudessa, kapasiteetissa, kaistojen ja laitteen sijainnin joustavuudessa sekä sovellusten kantamassa [2]. Sijainnin joustavuus tarkoittaa, että nykyään ei tarvita enää välttämättä erikseen GPS-sirua laitteeseen sijainnin määrittämiseksi. Sen sijaan laitteen sijainti saadaan, kun se yhdistetään useampaan tukiasemaan yhtä aikaa ja tällä tavoin voidaan määrittää sen sijainti signaalin kulkuajan viiveeseen perustuen. Yhteys useaan tukiasemaan yhtäaikaisesti mahdollistaa myös suuremmat käyttönopeudet, LTE-signaalia voidaan käyttää jopa 350 *km/h* nopeudessa signaalin katkeamatta.

Ensimmäiset makrokokoiset LTE-verkossa käyttöönotetut järjestelmät ylsivät tiedonsiirtonopeuksissa parhaimmillaan 300 *Mbps* 20 *MHz* taajuudella, kun käytössä oli jatkuva ja lisensioitu kaista [2]. LTE evoluution myötä tiedonsiirtonopeuksien huiput voidaan ulottaa *Gbps* alueelle, kun antenniteknologiat, monitie-etenemisen koordinointi, sirpaloitumisen estäminen ja lisensoimattomat kaistat saadaan käyttöön. LTE on laajentanut mahdollisuuksiamme valtavasti pelkästään laajakaistaisten matkapuhelinsovellusten ulkopuolelle, esimerkiksi suuret M2M (Machine to Machine, koneiden välinen kommunikaatio) verkot ovat mahdollisia. Tämä ominaisuus mahdollistaa myös tulevaisuudessa IoT -laitteiden määrän eksponentiaalisen kasvun.

Alkujaan 3G-tekniikkaa kehitettäessä pyrittiin löytämään yksi yhteinen tekniikkaa kaikkialle maailmaan. Sen ansiosta syntyi 3GPP (3 generation partnership project), joka toi kaikkialle maailmaan yhteisen 3G-tekniikan (Global System for Mobile telecommunications). LTE-teknologia on myös 3GPP-projektin kehittämää, joten sama yhdenmukaisuustavoite on pidetty mielessä myös sitä kehitettäessä. Kaistojen säännöstelyä hoidetaan nykyään sekä kansallisella että alueellisella tasolla, esimerkiksi Suomessa Viestintävirasto ja Euroopassa CEPT/ECC (The European Conference of Postal and Telecommunications Administrations/European Customer Center, Euroopan kuluttajakeskus). Maailmanlaajuisella tasolla säätely tapahtuu ITU:n (International Telecommunications Union) toimesta. Säännöstelyllä huolehditaan siitä, mitä spektrin kaistoja käytetään mihinkin sovellukseen ja että esimerkiksi lähetinten säteilytehot jäävät tiettyjen rajojen alapuolelle. ITU vaikuttaa myös epäsuorasti tuotteiden standardien määrittelyyn omien sääntöjensä avulla. [2]

Radiotekniikoille tyypillisin ominaisuus on suuri ja nopea signaalin vaihtelevuus radiokanavan olosuhteissa. Vaihtelu voi johtua esimerkiksi taajuusselektiivisestä häipymisestä, laitteiden välisestä etäisyydestä johtuvasta etenemisreitin katoamisesta ja satunnaisista häiriöistä muiden solujen ja tukiasemien lähetysten takia. LTE yrittää käyttää näitä haittoja hyväkseen sen sijaan, että niitä vastaan yritettäisiin taistella vain lisäämällä lähetystehoa. Jotta näitä seikkoja voitaisiin käyttää hyväksi, LTE käyttää

kaistariippuvaista ajoitusta jossa aika- ja taajuuskaistojen resurssit on dynaamisesti jaettu käyttäjien kesken [2]. Tämä tarkoittaa sitä, että erilliset käyttäjät saavat omille radiokanavilleen aina parhaat mahdolliset ominaisuudet käyttöönsä. Resurssien dynaaminen jakaminen on sovitettu tehokkaasti vastaamaan nopeasti vaihtuvia radiokanavan olosuhteita, jotka johtuvat pakettidataan perustuvan tiedonsiirron ominaisuuksista. Tämä mahdollistaa myös muita keskeisiä tekniikoita, joihin LTE perustuu.

Ajoituksia ohjataan jokaisella ajan hetkellä, siirtämällä käyttäjän liikenne eri osiin ja kontrolloimalla mitä osia jaetuista resursseista ja tiedonsiirtonopeuksista voidaan käyttää kulloiseenkin lähetykseen [3]. Tiedonsiirtonopeuden dynaaminen säätäminen vastaamaan kulloisenkin kanavan olosuhteita voidaan nähdä myös yhtenä osana ajoitusten toimintaa. Langattomaan tiedonsiirtoon liittyy aina mahdollisuus virheisiin ja vaikka tiedonsiirtonopeuden säätäminen onnistuisikin täysin ongelmitta, on silti aina tietty mahdollisuus virheisiin tiedonsiirrossa. LTE hallitsee tiedonsiirtovirheet käyttämällä *fast hybrid-ARQ with soft combining* nimistä tekniikkaa [2, 3]. Tämän tekniikan avulla laite voi LTE-verkossa pyytää nopeasti uudelleenlähetystä virheellisesti vastaanotetulle datapaketille lähettävältä laitteelta ja sen jälkeen käyttää virheenkorjaustyökaluja vastaanotetun datan korjaamisen epäsuoran tiedon avulla, joka kahden paketin eroavaisuuksista voidaan saada [2]. Uudelleenlähetystä voidaan pyytää jokaisen vastaanotetun datapaketin jälkeen ja tämän avulla saadaan minimoitua loppukäyttäjälle syntyvät tiedonsiirrosta johtuvat virheet.

Ajoitus on päätekijä LTE:n kehityksessä ja tehokkuudessa varsinkin paljon kuormitetuissa verkoissa. LTE:ssä sekä downlink (yhteys linkkitornista päätelaitteeseen) että uplink (yhteys päätelaitteesta linkkitorniin) yhteydet ovat tiukasti ajoitettuja toiminnallisuuden varmistamiseksi. Jos radiokanavien olosuhteet otetaan myös huomioon ajoitusta määritettäessä, saadaan aikaiseksi niin kutsuttu *channel-dependent scheduling,* eli kanavariippuvainen ajoitus, jossa lähetys on ohjattu käyttäjälle hetkellisesti kanavaa pitkin, jossa on suotuisimmat olosuhteet radiosignaalille. Koska downlink ja uplink lähetyksissä käytetään OFDM-tekniikkaa (Orthogonal Frequency-Division Multiplexing), on ajoituksen säätävällä piirillä mahdollista hallita sekä aika- että taajuusaluetta. Toisin sanoen ajoituspiiri voi jokaisella ajanhetkellä ja taajuusalueella valita käyttäjälle parhaat radiokanavan olosuhteet. [2-4]

LTE:n suunnittelussa taajuuksien uudelleenkäyttö oli yksi vaatimus. Se tarkoittaa, että samaa kantoaaltoa voidaan käyttää viereisissä soluissa. LTE suunniteltiin muutoinkin toimimaan hyvin suhteellisen matalalla signaali-häiriösuhteella (*signal-to-interference ratio*), joka on mahdollista taajuuksien uudelleenkäytön vuoksi. Mahdollisuus käyttää kaikkia taajuusresursseja vierekkäisissä linkkitorneissa on aina hyödyllistä, mutta järjestelmän tehokkuutta ja loppukäyttäjälle tulevan signaalin laatua voidaan parantaa entisestään, jos viereisten linkkitornien toimintaa voidaan koordinoida ja sillä tavalla välttää pahimmat häiriöt aiheuttavia tilanteita. Pahimmat häiriöt voivat muodostua esimerkiksi niin sanotuissa *heterogeneous network deployments*-tilanteissa, eli tilanteissa, joissa vierekkäisten tukiasemien peittoalueet menevät päällekkäin ja downlink lähetystehoissa on suuri ero. [2, 3]

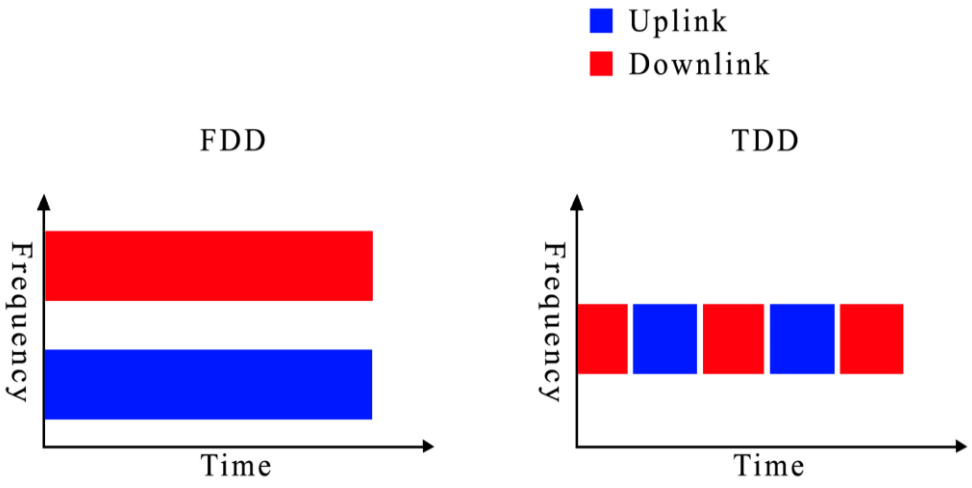
Jo alusta asti LTE on tukenut erilaisia moniantennitoteutuksia, joissa voidaan joko lähettää (SIMO, Single-Input-Multiple-Output) tai vastaanottaa (MISO, Multiple-In-

put-Single-Output) usealla antennilla tai sekä lähettää että vastaanottaa usealla antennilla yhtä aikaa, jolloin kyseessä on MIMO-systeemi (Multiple-Input-Multiple-Output). MISO-toteutuksella saadaan lisättyä vastaanoton monimuotoisuutta ja sillä pystytään vähentämään myös häiriöitä. SIMO-toteutuksella saadaan muun muassa tehtyä erilaisia keilakuvioita lähettävä järjestelmä. Keilakuvioilla voidaan vaikuttaa merkittävästi lähetettävän signaalin kantamaan ja kapasiteettiin. MIMO-toteutuksella saadaan useampi taajuuskaista yhtäaikaiseen käyttöön päätelaitteelle. MIMO-tekniikan avulla saadaan käyttöön suurempi tiedonsiirtonopeus ja kanavan olosuhteet voidaan määritellä, kun luodaan useampi rinnakkainen ”radiokanava”. [2, 4]

Radiokanavien joustavuus on yksi LTE:n radioliityntätekniikoiden pääominaisuuksista. Joustavuuden päätavoitteena on mahdollistaa pääsy LTE:n eri liityntätekniikoiden (RAT) avulla eri taajuusalueille, joilla on useita eri ominaisuuksia [2], mukaan lukien erilaiset kaksikerrosjärjestelmät ja erikokoisten avoimien taajuuskaistojen käyttö.

Yksi vaatimus kanavien joustavuuden suhteen LTE:tä suunniteltaessa oli mahdollisuus käyttää sekä parillisia että parittomia kaistoja [2]. LTE tukee taajuusjaettuja (FDD, Frequency-Division Duplexing) ja aikajaettuja kaksikerrosjärjestelmiä (TDD, Time-Division Duplexing). Kuvasta 1 nähdään hyvin ero FDD:n ja TDD:n toiminnan välillä.

FDD toimii siten, että downlinkille ja uplinkille on omat taajuuskaistansa ja välissä on niin kutsuttu *guard band,* eli suojakaista, joka estää downlinkin ja uplinkin lähetyksiä häiritsemästä toisiaan. FDD:n vahvuus on, että suunnittelu on helpompaa koska uplink ja downlink eivät ”kuule” toisiaan. FDD on hyvä valinta, kun tiedonsiirto on symmetristä, koska molemmille suunnille on tällöin oma kaistansa.



Kuva 1. Vasemmalla FDD, oikealla TDD

TDD:ssä käytössä on yksi kaista, jota katkotaan ajan suhteen downlinkille ja uplinkille. Käytännössä TDD:tä käytettäessä laite voi siis vain vastaanottaa tai lähettää kerrallaan. FDD:llä on suojakaista, samoin TDD vaatii suoja-ajan, *guard time*, eli lyhyen ajan, joka uplinkin ja downlinkin välillä on, jotta lähetykset eivät mene keskenään sekaisin. TDD sopii parhaiten epäsymmetrisiin tilanteisiin, koska se voi antaa dynaamisesti enemmän aikaa kaistalla uplink- tai downlink-yhteydelle, riippuen siitä kumpaa tarvitaan enemmän. Tiedonsiirron ollessa symmetristä TDD hukkaa kaistanleveyttä, joten se on yksi syy käyttää FDD:tä sellaisissa tilanteissa [2].

LTE tukee myös puolikasta FDD-kaksikerrosjärjestelmää, *half-duplex FDD*, mikä tarkoittaa, että lähetys on jaettu sekä ajan että taajuuden suhteen. Sekä uplinkillä että downlinkillä on omat taajuuskaistansa, mutta sen lisäksi ne on myös jaettu ajan suhteen aivan kuten TDD:ssä. Tämän tekniikan etuna on huomattavasti yksinkertaisempi rakenne, koska kaksoissuodattimia (*duplex filter, mikä olisi hyvä suomennos?*) ei tarvita [2]. Tämä on hyödyllistä erityisesti silloin, kun laitetta voidaan käyttää useammalla kaistalla yhtä aikaa, jolloin laite tarvitsisi useita kaksoissuodattimia, jos käytettäisiin FDD:tä tai TDD:tä.

LTE:n laaja taajuuskaistojen tuki on mahdollista, koska LTE suunniteltiin käyttämään joustavaa lähetyskaistanleveyttä. Leveä lähetyskaista on välttämätön, että voitaisiin pitää tehokkaasti yllä suuria tiedonsiirtomääriä aina kun kaistaa on avoinna. Aina ei kuitenkaan ole mahdollista saada riittävän laajaa kaistaa käyttöön, näissä tapauksissa LTE kykenee toimimaan myös kapeammalla lähetyskaistalla, jolloin tiedonsiirtomääriä joudutaan luonnollisesti pienentämään. Taajuuskaistojen käytön joustavuutta on kehitetty ensimmäisistä LTE versioista, jotta päästäisiin eroon tiedonsiirron kapasiteetin pienenemisestä. [2]

Tällä hetkellä LTE radioliityntätekniikat tukevat paljon laajempia kaistanleveyksiä kuin RF-komponentit (RF, radio frequency), jotka on yleensä suunniteltu vain jollekin tietylle taajuusalueelle. Tulevaisuudessa uusien kaistojen käyttöönotto onnistuu siis vain päivittämällä RF-komponenttien vaatimukset sopivalla tavalla. [2]

2.3. Esineiden Internet

IoT-tekniikoiden läpimurto on tapahtumassa seuraavien vuosien aikana, kun lyhytkantamaiset radiotekniikat, kuten LTE-M (Long Term Evolution for Machines) ja NB-IoT (Narrowband Internet of Things), kehittyvät ja mahdollistavat suurten datamäärien siirtämisen useista tuhansista sensoreista yhteiseen pilvipalveluun. IoT:n haasteet tulevat todennäköisesti olemaan lähivuosina muita kuin puhtaasti teknisiä ongelmia, esi-

merkiksi kuinka valvotaan kuka saa liittyä verkkoon, paljonko tiedonsiirtokapasiteettia voidaan käyttää sekä mitä verkon käyttö maksaa suhteessa IoT-laitteen tai -sensorin hintaan.

Tällä hetkellä IoT:hen liittyvät sovellukset ovat niin uusia, ettei termistö ole vielä vakiintunut. Tämän takia IoT:stä puhuttaessa tulee varmistua, tarkoittaako puhuja ylipäänsä verkon yli toimivia laitteita, kuten 4K-valvontakameraa vai teollisia IoT -laitteita (*massive IoT*) eli sensoreita, joita on suuri lukumäärä ja jotka lähettävät dataa itsenäisesti. Yksittäisen 4K-kameran syöttämä datamäärä verkkoon voi olla suurempi kuin tuhansien yksittäisten IoT-sensorien syöttämä datamäärä. Se pitää huomioida myös verkon kuormituksessa. Tässä kandintyössä tarkoitetaan IoT-laitteilla nimenomaan teollisia IoT -laitteita. Teolliset IoT-laitteet ovat mielenkiintoinen tutkimuskohde juuri monimuotoisuutensa takia. Mahdollisuudet mitä mittausdataa voidaan tuottaa suurella määrällä IoT-sensoreita, on kiinnostava, koska se antaa mahdollisuuden luoda ympäröivästä maailmasta kuvan aivan uudella tavalla.

Tällä hetkellä voimakkaimmin viedään eteenpäin LPWAN-radiotekniikoita (Low-Power Wide-Area Network) [5]. Kilpailevia tekniikoita ovat muun muassa GSM (Global System for Mobile communications), LTE-M ja NB-IoT, kaikki tekniikat ovat 3GPP standardisoinnin saaneita. LTE-M ja NB-IoT ovat tekniikkoina kapeakaistaisia ja lyhytkantamaisia tekniikoita, jotka soveltuvat erityisesti IoT-maailman tarpeisiin. IoT-laitteet ovat yleisesti langattomassa verkossa kiinni lyhyellä kantamalla ja yksittäisen sensorin lähettämä datamäärä on pieni, jolloin ei tarvita laajakaistaisia verkkoja. Tyypillistä näille kaikille tekniikoille onkin suhteellisen pitkä kantama, pieni tiedonsiirtonopeus, noin 1 - 500 *kbps*, sekä matala tehonkulutus.

LTE-M tarjoaa parhaat turvallisuusominaisuudet verrattuna muihin verkkotekniikoihin, koska se käyttää LTE-tekniikan tavoin IPSec-protokollaa [6]. Muissa tekniikoissa tietoturva on toteutettu eri tavoilla, muun muassa käyttäjän luottamuksellisella tunnistuksella, kokonaisuuden tunnistuksella, mobiililaitteen tunnistuksella ja datan koskemattomuuden tunnistuksella [3, 6].

IoT-vallankumouksen mahdollistaa 5G-tekniikan kehittyminen. Pienemmät tiedonsiirtonopeudet riittävät IoT-laitteille, kun tietoverkkojen solukokoja saadaan pienennettyä, jolloin yhdessä tukiasemassa ei ole montaa sensoria kiinni ja kuormittamassa tukiaseman luomaa langatonta verkkoa.

Merkittävin muutos lienee solukoon pieneneminen, jonka 5G-tekniikka tuo tullessaan, samalla se mahdollistaa myös IoT-tekniikan kasvamisen, kun tiedonsiirtokyky kasvaa merkittävästi. Tällä hetkellä nopeimmissa LTE-verkoissa päästään 150 *Mbps* nopeuteen, 5G-tekniikoissa puhutaan gigabitti per sekunti luokan nopeuksista.

Pienet solut mahdollistavat suuremmat tiedonsiirtonopeudet myös tiheästi asutetuilla alueilla tai tapahtumissa, joissa on paljon ihmisiä paikalla samaan aikaan. Solukokojen pieneneminen on väistämätön seuraus käytettävien kaistojen taajuuksien kasvamisesta, koska korkeataajuinen signaali ei kanna pitkälle.

Friisin siirtoyhtälö (2) osoittaa tämän hyvin selvästi, koska taajuuden kasvaessa eli aallonpituuden lyhentyessä, tämä nähdään helposti aalto-opin perusyhtälöstä (1), vastaanotettu teho pienenee nopeasti.

$$c = f\lambda \qquad (1)$$

Yhtälössä c on valon nopeus tyhjiössä, f on sähkömagneettisen aallon taajuus ja λ on aallonpituus.

$$P_r(D) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 D^2 L}, \text{missä } G = \frac{4\pi A_e}{\lambda^2} \qquad (2)$$

P_t on lähetetyn signaalin teho, P_r on vastaanotetun signaalin teho, G_r ja G_t ovat antennivahvistuksia, D on etäisyys, L on häviökerroin, ja A_e on antennien efektiivinen pinta-ala.

Osa radiotaajuuksista on lisensioituja ja osa lisensioimattomia. Lisensioimattomilla radiotaajuuksilla kuka tahansa saa käyttää taajuutta. Esimerkiksi ISM-kaista (Industrial, Scientific and Medical) on tällainen kaista. Kaista on teollisuuden, tekniikan ja lääketieteellisten sovellutusten käytössä ja sen käyttämiseen ei tarvitse anoa lupia, Suomessa esimerkiksi luvista vastaa Ficora. IoT-sovellusten lisääntyessä herää kysymys mitä kaistoja voidaan käyttää. Käyttävätkö laitteet avoimia kaistoja, jolloin ne jossain kohtaa väistämättä ruuhkautuvat vai syntyykö uusi markkina-ala, joka jakaa maksullisia kaistoja IoT-laitteille?

Maksullisten palveluiden ongelmaksi tulee, miten hinnoittelu tapahtuu. Sensorien lukumäärän lähestyessä ääretöntä, lähestyy yksittäisen sensorin datan arvo nollaa, koska se voidaan saada myös välillisesti jostain toisesta sensorista. IoT-sensorin arvo yleisesti on muutamista euroista muutamiin kymmeniin euroihin, joten laitteen hinta on niin pieni, että helposti tiedonsiirrosta aiheutuvat kustannukset nousisivat laitteen hintaa suuremmiksi. Jos huomioidaan vielä, että tämä sama data voitaisiin saada myös epäsuorasti käyttämällä jo implementoitua sensoria, tulee selväksi, että palvelu ei olisi houkutteleva. [5]

Yksittäisen IoT-sensorin todentaminen verkossa lienee tärkein tekijä tässä tilanteessa. Jos yksittäinen sensori voidaan todentaa, sitä voidaan laskuttaa lähetetyn datan määrän mukaan. Tämän valvomiseen helpoin tapa voisi olla SW-SIM (Software-Subscriber Identity Module) eli ohjelmistopohjainen SIM-kortti, koska IoT-sensoreiden fyysinen koko ei ole kovin suuri. Edelleen ongelmaksi tulisi kuitenkin datansiirron hinta suhteessa saavutettuun hyötyyn.

Toinen vaihtoehto voisi olla elinikäinen käyttöoikeus, kun ostaa IoT-laitteen [5]. Tällöin laitteen hintaan olisi sisällytetty myös tiedonsiirron hinta. Tällainen tapa voisi olla kuluttajalle helpompi ja mahdollistaisi IoT-laitteiden määrän kasvun, kun jokaisen sensorin kohdalla ei tarvitsisi käydä pohdintaa sensorin datan tarpeellisuudesta. Tämän hyötynä olisi myös, ettei yksittäisistä laitteista tarvitsisi pitää kirjaa, koska kaikki verkkoon liitettävät laitteet olisivat ”maksaneet” jo verkon käytöstä. Tässä suurimmaksi ongelmaksi nousee, miten laitevalmistaja jakaa tuotot verkon ylläpitäjille.

Big data on uusi käsite, joka on syntynyt oikeastaan IoT-laitteiden yleistymisen myötä, sillä tarkoitetaan suurta määrää dataa, joka ei ole millään tavalla järjesteltyä ja se lisääntyy jatkuvasti ja sitä täytyisi pystyä käsittelemään analyyttisesti ja säilömään jonnekin. Big datan keskeisin ongelma on tietomäärän valtavuus ja ettei sitä pystytä käsittelemään mielekkäästi käyttäen perinteisiä työkaluja, koska laskenta-ajat kasvavat liian suuriksi. Big data liittyy keskeisesti IoT-laitteisiin, koska IoT-laitteiden määrän kasvaessa, niiden siirtämä datan määrä kasvaa ja täytyy miettiä, mihin se data saadaan säilöttyä myöhempää käyttöä varten. Tästä päästään seuraavaan ongelmaan eli kuinka paljon yksittäisen sensorin datalla on arvoa.

Datan arvo voidaan määritellä kahdella tavalla:

1. Monesta sensorista kerätty data
2. Yhdestä sensorista pitkältä aikaväliltä kerätty

Käsitellään ensin tapausta 1. Jos meillä on esimerkiksi moottori, jota mitataan 15 sensorilla, yksittäisen sensorin datalla ei välttämättä ole mitään arvoa, mutta kun kaikkien sensoreiden data yhdistetään, saadaan hyvä kuva moottorin kokonaistoiminnasta ja suorituskyvystä. Toisin sanoen yksittäisen sensorin datalla ei ole mitään arvoa, mutta datapisteiden yhteisvaikutus luo kokonaisdatalle arvon.

Tapauksessa 2 meillä voi olla esimerkiksi sääasema. Yksittäisellä datapisteellä ei ole mitään arvoa tässäkään tapauksessa, mutta kun saadaan esimerkiksi 5 vuoden ajalta kaikki yhden sensorin datapisteet käyttöön, voidaan muodostaa kuva sääoloista tietyssä paikassa, jolloin kokonaisdatalle tulee jälleen arvoa.

IoT-laitteiden kanssa tulee väistämättä mieleen ongelma, että mihin kaikki data säilötään. Jo tällä hetkellä maailma tuottaa uutta dataa ennennäkemättömällä vauhdilla, esimerkiksi Instagramiin ladataan keskiarvoisesti 52 miljoonaa kuvaa päivittäin [7]. Yhden Instagramiin ladattava kuvan koko on noin 4 *MB*, eli se tarkoittaisi, että päivittäin tarvitaan noin 208 *TB* uutta tilaa pelkästään Instagramiin julkaistaville kuville. Vuodessa se tarkoittaisi 75 920 *TB* uutta tilaa. Tähän kun lisätään kaikki muutkin sosiaalisen median palvelut ja niiden tarvitsema tila, ei ole epäilystäkään, etteikö tallennustila joskus lopu.

Yhden IoT-laitteen tuottama datapiste on kooltaan joitain kilotavuja. Kun laitteita on tulevaisuudessa miljardeja, syntyy joka hetki uutta dataa teratavujen vauhdilla, ja tämä kaikki data pitäisi säilöä johonkin. Kun päädytään tilanteeseen, että tallennuskapasiteettimme on loppu, täytyy miettiä mitä dataa aletaan poistaa, että kaikki uusi data saadaan mahdutettua ja onko kaikki uusi data niin arvokasta, että se kannattaa tallentaa.

Pelkkä datan säilöminen ei riitä, vaan sitä pitää pystyä analysoimaan. Analysointiin on useita eri menetelmiä, tärkeäksi IoT-laitteiden kanssa tulee kuitenkin analysoinnin nopeus, koska uutta dataa syntyy nopeasti. Analyysin nopeus auttaa sensorien tehokkaammassa toiminnassa, koska havainnot ovat reaaliaikaisia eivätkä tule useiden tuntien tai päivien viiveellä. Tässä työssä jyrsimeen kiinnitetystä tärinäsensorista tule-

vasta datasta täytyy saada eroteltua tärinän taajuuskomponentit ja analyysin avulla verrattua tärinää normaaliin, jotta voidaan tehdä johtopäätöksiä siitä, toimiiko laite oikein vai ei. Analyysin tekeminen ei kuulu tämän työn piiriin.

2.4. IoT sensorit

Tätä kandintyötä aloitettaessa sensoriksi oli useita vaihtoehtoja, muun muassa Libeilum, iProtox, Arduino ja Thingsee. Lisäksi Nokialta tulee IMPACT alusta, joka tukee IoT-kehitystä ja siirtotekniikoina LoRa ja NB-IoT. Kaikki edellä mainitut ovat kehitysalustoja, joissa on useita eri sensoreita tai mahdollisuus erilaisten sensorien liittämiseen. Kehitysalustojen tarkoituksena on osoittaa prototyypin toimivuus, ei luoda lopullista tuotetta. Kehitysalustojen ongelmana olisikin ollut todennäköisesti liian suuri koko lopulliseen toteutukseen.

Lopulta päädyin käyttämään iSensorin ADIS16227 sensoria, kun tutkimuskohde tarkentui nimenomaan CNC-jyrsimen tärinän mittaukseen. Sensorin valinnassa tärkeää oli sensorin toiminnallisuuden soveltuminen käsillä olevan ongelman ratkaisemiseksi, mutta myös sensorin koko.

IoT-sensoreilla mitattavien suureiden määrä on lähes loputon, sensoreilla voidaan mitata helposti esimerkiksi lämpötilaa, kosteutta, värinää, sijaintia, nopeutta, kiihtyvyyttä ja painetta. Sensorien monimuotoisuuden ansiosta IoT-sensoreiden implementointi on mahdollista lähes mihin tahansa. Näiden lisäksi sensoreilla voidaan epäsuorasti mitata myös muita suureita, jotka saadaan määriteltyä mitattujen arvojen perusteella. Tällä hetkellä ainoastaan energiankulutus ja siten akun kesto tulevat vastaan sensorien mahdollisuuksia määriteltäessä.

IoT-laitteissa yksi tärkeimmistä ominaisuuksista on mahdollisimman pieni energiankulutus, ettei laitteiden paristoja tarvitse olla jatkuvasti vaihtamassa tai akkuja lataamassa. IoT-laitteiden paristojen kestoa voidaan parantaa, kun laite osaa mennä stand-by tilaan käyttämättömänä, ilman tätä ominaisuutta laite pystyy toimimaan vuodesta kymmeniin vuosiin ilman latausta muutaman ampeeritunnin virtalähteellä. Esimerkiksi Thingsee One kehitysalusta pystyy toimimaan jopa vuoden yhdellä lautauksella [8]. Merkittävin tekijä tässä on laitteen virrankulutus päällä ollessaan. Nykyiset langattomat tekniikat pyrkivät kohti pienempiä virrankulutuksia mahdollistaakseen IoT-laitteiden mahdollisimman pitkän käyttöiän. IoT-laitteille suurin energiankulutus syntyy, kun laite joutuu kuuntelemaan ympäristöään tai alkaa lähettää [9]. Tällöin tarvittava teho on noin 70 mW, jos laite toimii 3 voltin paristolla, se tarkoittaa Ohmin lain (3) mukaan, että tarvittava virta on noin 23 mA ja silloin parin milliampeeritunnin virtalähde toimisi noin 100 tuntia.

$$P = UI \tag{3}$$

$$P\ on\ teho, U\ jännite\ ja\ I\ virta$$

Nukkumistilassa tehonkulutus on vain muutaman milliwatin luokkaa, jolloin tarvittava virta putoaa noin milliampeeriin ja sensori jaksaisi toimia 2 ampeeritunnin virtalähteellä 2000 tuntia. Sensoreissa käytetään jännitteeltään suurempia akkuja, kuin edellä mainitsemani 3 volttia, minkä ansiosta akun kesto voidaan saada venytettyä vuosien mittaiseksi. Lisäksi sensoreissa voidaan käyttää erilaisia lataustekniikoita, esimerkiksi pietso-kiteellä voidaan liikkeestä saada aikaan sähköä, joka lataa akkua.

Sensoreiden tuottama datamäärä ei ole suuri ja sen vuoksi IoT-laitteita varten kehitetyt siirtotekniikat eivät tue kovin suuria siirtonopeuksia yksittäiselle laitteelle. 5G-tekniikan kehitys kuitenkin mahdollistaa IoT-laitteiden tehokkaan käytön, kun useista pienistä datavirroista syntyy suurempi kokonaisuus ja 5G-verkko pystyy käsittelemään sen edelleen riittävän nopeasti. IoT-laitteiden tiedonsiirtonopeudet ovat tyypillisesti 1-500 *kbps* luokkaa, mikä on selkeästi alhaisempi kuin 5G-verkon siirtokapasiteetti, joka voi olla jopa 10 *Gbps* downlink suuntaan [10]. Bluetooth on ainoa tiedonsiirtotapa, joka yltää 1 *Mbps* tiedonsiirtonopeuteen. Yksittäisen mittauspisteen synnyttämä datan määrä ei ole kovin suuri ja sen takia IoT-laitteille suunnitelluissa tiedonsiirtotekniikoissa on keskitytty ennemmin pieneen tehonkulutukseen kuin suureen tiedonsiirtokapasiteettiin.

IoT-sensorit tulevat olemaan osana joka päivästä elämäämme kaikkialla ympärillämme, ilman että edes huomaamme niitä. Sensorit tekevät elinympäristöstämme älykkään kertomalla, mitä elinympäristössämme tapahtuu. Yleisimmät esimerkit ovat älykodit ja älykaupungit.

Älykotien ideaa voidaan hyödyntää muun muassa vanhustenhoidossa. Muistisairaita ihmisiä ei tarvitsisi enää lukita asuntoihinsa vanhainkodeissa, vaan ovet voisivat olla normaalisti auki, mutta lukkiutuvat, kun muistisairas henkilö menee lähelle ovea. IoT-sensorit ja 5G-verkko mahdollistavat riittävän tarkan paikannuksen tällaisen systeemin toteuttamiseen.

Älykaupungeissa esimerkiksi jätehuolto voidaan toteuttaa älykkäästi, siten että roska-astiat kertovat jätehuollosta huolehtivalle taholle, milloin astia on valmis tyhjennettäväksi. Sensoroinnin avulla jätehuolto tulee ekologisemmaksi, kun turhilta jäteautojen kierroksilta vältytään ja toisaalta astiat eivät pullistele täysinä pitkiä aikoja ilman tyhjennystä. [11]

IoT-sensorit tulevat yleistymään urheilussa yhä enemmän. Tällä hetkellä esimerkiksi Polar ja Haltia kehittävät puettavaa teknologiaa, joka mittaa sykettä, kuljettua matkaa, unta ja muita ominaisuuksia käyttäjästään. Näiden sensorien data voidaan huippu-urheilussa yhdistää suoraan pilveen, josta valmentaja voi nähdä urheilijan suorituksen ja kehottaa harjoittelemaan kovempaa tai keventää harjoittelua. Erilaiset kannustusviestit ovat myös mahdollisia näiden uusien laitteiden kautta. Suurissa urheilutapahtumissa katsojat pystyisivät seuraamaan suosikkiurheilijaansa koko ajan, näkemään urheilijan sykkeen, lihasten lämmön tai hengitystiheyden.

2.5. Tietoturva

IoT-laitteiden kohdalla tietoturvan merkitys tulee kasvamaan, koska verkkoon liitettävien laitteiden määrä kasvaa valtavasti. Suojaamattomuudesta konkreettisena ongelmana on esimerkiksi palvelunestohyökkäysten mahdollisuus käyttäen sensoreita ja tietysti sensorien datan varastamisen tai muokkaamisen mahdollisuus. Palvelunestohyökkäyksestä hyvänä esimerkkinä on 26. lokakuuta 2016 tapahtunut hyökkäys Dyn:iä vastaan [12]. Hyökkäyksessä kaadettiin Dyn:in servereitä aiheuttaen useiden verkkosivujen kuten Twitter, Spotify ja Pinterest, kaatumisen. Hyökkäyksessä noin sata tuhatta IoT-laitetta hyökkäsi serverien kimppuun DDoS-hyökkäyksellä (Distributed Denial of Service). Tavalliset valvontakamerat, digitaaliset videotallentimet valvontakameroille ja reitittimet hyökkäsivät Mirai nimisen haittaohjelman saastuttamina palvelimen kimppuun. Niinkin pienellä määrällä kuin sata tuhatta laitetta pystyttiin aiheuttamaan jopa 1 *Tbps* datavirta servereille ja kaatamaan servereitä [12]. Hyökkäys oli isoin tähän saakka tapahtuneista hyökkäyksistä ja osoitti, kuinka vaarattoman tuntuisilla laitteilla saadaan aiheutettua paljon ongelmia, jos niitä ei suojata kunnolla. Huolestuttavaa on myös valvontakameroiden suojauksen puute, koska silloin kuka tahansa voi verkon välityksellä katsella valvontakameroiden kuvaa ja esimerkiksi suunnitella ryöstön perustuen ihmisten päivittäisiin rutiineihin.

Serverihallien sijoittaminen lähelle tukiasemaa parantaisi verkon nopeutta lyhentäen verkon latenssia, koska reunalaskentatekniikka (MEC, Mobile-Edge Computing) on silloin mahdollista. Ongelmatonta tämäkään ei kuitenkaan ole, koska jos serverihallit sijoitettaisiin lähelle mastoja, niiden vartiointi täytyisi toteuttaa hyvin. Jos serverihalliin pääsisi ulkopuolinen sisään, hän voisi päästä käsiksi servereihin ja sitä kautta niiden läpi kulkevaan dataan.

Taulukossa 1 on esitetty neljä eri tietoturvan riskityyppiä ja esimerkit tyypeistä. Tietoturva koostuu paljon muustakin kuin pelkästään palomuurin ylläpidosta. Jos servereille päästään käsiksi mitä tahansa reittiä, olipa se fyysisesti tai digitaalisesti, syntyy varmasti ongelmia.

Isoimmat IoT-laitteisiin liittyvät uhat ovat tietomurrot ja fyysisesti serverille pääseminen. Silloin laitteiden lähettämää tietoa voitaisiin muokata ja sitä kautta aiheuttaa ongelmia esimerkiksi yleisen turvallisuuden kannalta. Jos esimerkiksi tehdasoloissa tilaa valvovan sensorin dataa päästään muuttamaan, saattaa laite kertoa, että tilassa on liikaa jotain myrkyllistä kaasua, mutta asialle ei tehdä mitään, koska tietoa on muutettu matkalla ja valvova systeemi luulee kaiken olevan hyvin, koska sille tuleva data on kunnossa.

Datan muuntelun ongelmaan on onneksi kehitetty jo blockchain niminen tekniikka. Siinä ideana on, että kun lohko (block) on kerran määritelty, sitä ei voida enää muuttaa ja lohkossa on aikaleima kertomassa, milloin se on luotu. Lohko lisätään kronologisesti ketjuun (chain) ja sen paikkaa ei myöskään voida muuttaa enää paikalleen asettamisen jälkeen. Muokkaamisen tekee mahdottomaksi se, että lohkon tietoja muuttaakseen pitäisi muokata sekä lohkoa, että kaikkia lohkoa edeltäneitä lohkoja.

Blockchain on käyttäjälle turvallinen, koska kaikki käyttäjät voivat luottaa toisiinsa, vaikka eivät tuntisi toisiaan ja sen takia esimerkiksi BitCoin on yleistynyt paljon[13].

Taulukko 1. Taulukossa eri tietoturvariskien tyypit ja esimerkit siitä mitä ne tarkoittavat

Aiheuttaja / **Kohde**	**Fyysinen**	**Digitaalinen**
Fyysinen	**Ihmisen toimet** Tehdään fyysistä tuhoa fyysiselle rakenteelle.	**Fyysisesti serverille** Mennään serverihalliin ja sitä kautta päästään fyysisesti käsiksi serveriin.
Digitaalinen	**Hakkeroimalla uhkatila** Hakkeroidaan jokin kriittinen systeemi ja aiheutetaan sen avulla uhkatila.	**Tietomurrot** Tietomurrot verkon kautta hakkeroitumalla, perinteinen käsitys tietoturvaan liittyvistä uhista.

BitCoin perustuu blockchainin hyödyntämiseen, Bitcoinin luominen ei olisi ollut mahdollista ilman varmasti luotettavaa tekniikkaa, jolla voidaan olla varmoja, ettei samaa rahaa voi käyttää moneen kertaan, virtuaalinen rahan väärentäminen on siis estetty käyttämällä blockchainia, koska lohkoja ei pystytä muokkaamaan.

Blockchainin muita sovelluskohteita voisi olla digitaalinen äänestyssysteemi, koska silloin voitaisiin olla varmoja, että annettu ääni pysyy samana alusta loppuun. Blockchainin isoin hyöty ei ole kuitenkaan tekniikkaan liittyvä toteutus, vaan puhdas luottamus osapuolten välillä. Tätä voitaisiin hyödyntää myös IoT-laitteissa, kun voitaisiin olla varmoja, että saatu data on varmasti oikein. Tällä siis päästäisiin tietomurron tuomasta ongelmasta ja aiemman esimerkin kaltaisista vaaratilanteista.

3. TYÖN TOTEUTUS

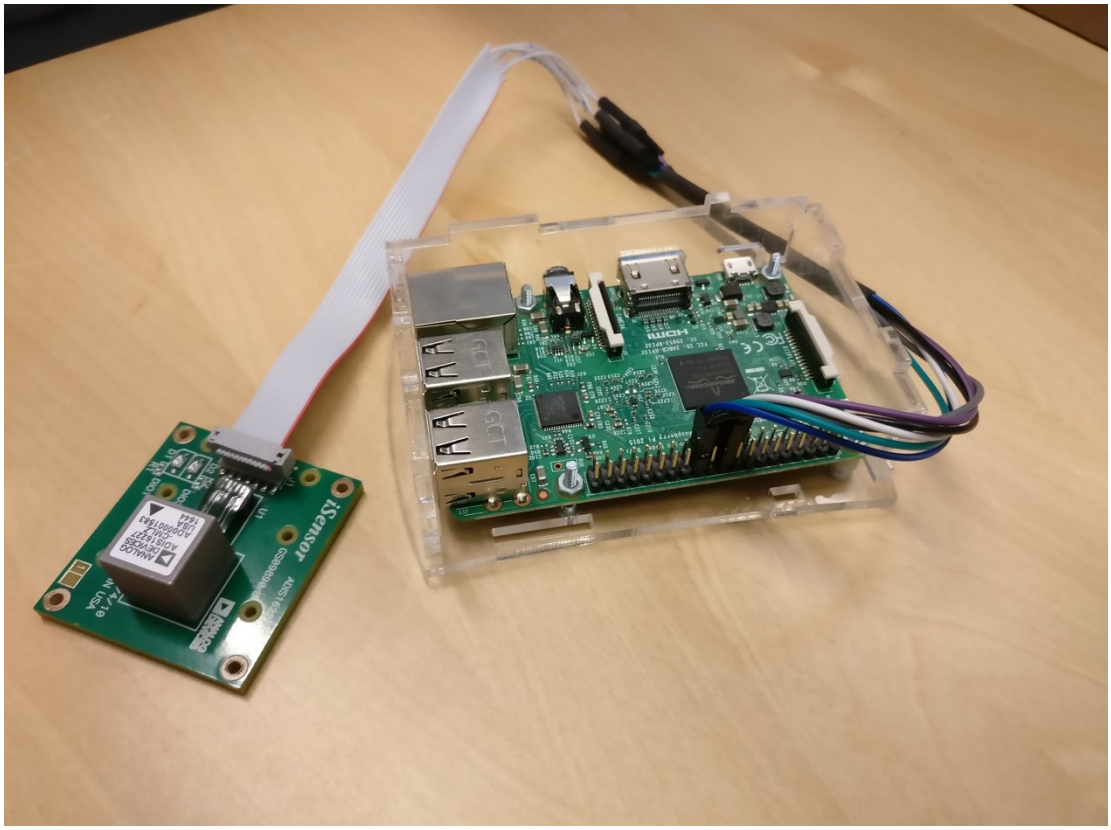
Tässä työssä käytettiin Raspberry Pi 3-tietokonetta ohjaamaan ADIS16227 sensoria, joka oli kiinnitetty valmistajan omaan ADIS1622x/PCBZ kehitysalustaan. Kehitysalustasta saatiin helposti tehtyä kytkennät Raspberry Pi-tietokoneeseen ja kaikki data saatiin kulkemaan SPI-väylän kautta.

3.1. Kytkennän kokoonpano

Sensorin koodaukseen käytin C-kielellä. Aluksi yritin käyttää wiringPi-kirjastoa SPI-väylän ohjaamiseen, mutta koska kirjasto ei toiminut haluamallani tavalla ilman kirjaston muokkaamista, siirryin käyttämään spidev.h header-tiedostoa, jonka avulla sain SPI-väylän toimimaan. SPI-väylän saattaminen toimivaksi vaati melko paljon töitä ja useiden asetusten säätämistä. Lopulta sain SPI-väylän ja sensorin kommunikoimaan keskenään seuraavilla asetuksilla:

1. CE0 pinni Raspberry Pi:stä
2. SPI_mode_3
3. Sanapituus 8 bittiä
4. Väylän nopeus 1 *MHz*
5. Viive bitin jälkeen 0

Sensorin ja Raspberry Pi-tietokoneen yhteyden testaamisessa hyvä viesti oli PROD_ID, joka palauttaa tuotteen tunnuksen 0x3F63, tätä oli helppo käyttää asetusten toimivuuden tarkastamisessa. Kuvassa 2 näkyy käytetty kokonaisuus. Sensorista tulee oma lyhyt liitin kehitysalustan GPIO (General Purpose I/O) pinnikampaan. Kampa on yhdistetty Raspberry Pi:n GPIO pinneihin 17-24, joilla ohjataan SPI-väylää.



Kuva 2. Vasemmalla ADIS16227 sensori (pieni kuutio) kehitysalustallaan ja oikealla Raspberry Pi tietokone.

3.2. Datan kerääminen

Sensorille kirjoitetaan yksi käsky aina kahdella 16 bitin yhdistelmällä, joiden jälkeen odotetaan vähintään 20 mikrosekuntia ennen seuraavien käskyjen kirjoittamista, odotusaika voi olla pidempikin, datalehdestä löytyvät käskyjen vaatimat ajat [14]. Ensimmäiset 16 bittiä ovat nollia, ja jälkimmäiseen 16 bittiin kirjoitetaan haluttu rekisteri ja käsky. Rekisterien osoitteet löytyvät sensorin datalehdestä [14]. Rekisterin bittejä voidaan kirjoittaa vain 8 ylintä tai alinta yhdellä kerralla. Ensin kirjoitetaan ylemmät 8 bittiä eli bitit 15-8. Bitti 15 täytyy olla 1 kun halutaan kirjoittaa rekisteriin ja 0 kun halutaan lukea rekisteristä. Tämän jälkeen Kirjoitetaan rekisterin alemmat 8 bittiä eli bitit 7-0. Tämän jälkeen pidetään 20 mikrosekunnin tauko ennen seuraavan viestin kirjoittamista.

Ensimmäiseksi sensorille lähetetään asetukset, joilla dataa halutaan tallentaa. Nämä tiedot tallennetaan sensorin REC_CTRL rekisteriin. Sen jälkeen odotetaan, että kaikki tiedot on tallennettu rekisteriin. Sen jälkeen GLOB_CMD rekisterin bitti 7 asetetaan

ykköseksi, jotta tallennus alkaisi REC_CTRL rekisteriin tallennettujen asetusten mukaan. Käyttämilläni asetuksilla tallennus kestää 42.15 ms.

Kun tallennus on valmis, sensorille lähetetään kaksi kertaa lukukäsky. Ensimmäinen käsky valmistelee rekisterin ja toinen aloittaa varsinaisten arvojen lukemisen rekisteristä. Jokainen mittaus tuottaa 256 näytettä jokaiselle akselille. Jokainen mittaus vaatii siis yhteensä 768 iteraatiota, että kaikkien kolmen akselin data saadaan luettua rekisteristä. Iteraatioiden määrä aiheuttaa merkittävää viivettä datan lukemisessa ja siksi sensori ei yksinään sovi reaaliaikaiseen mittaukseen.

3.3. Pohdinta

Tähän työhön valittu ADIS16227 värinäsensori osoittautui työn aikana huonosti soveltuvaksi haluttuun tehtävään. Työn toteutuksen aikana kävi ilmi, ettei sensorilla pystytä suorittamaan reaaliaikaista mittausta ja sen takia tässä työssä käytettyä laitekonfiguraatiota voidaan käyttää enemmän musta laatikko –tyyppisenä ratkaisuna, kun tarkastellaan jälkeenpäin mitä on tapahtunut. Sensorin rekisteristä lukeminen on liian hidas toimenpide, jotta voitaisiin saada edes lähelle reaaliaikainen mittaus. Yhden rekisterin sisällön lukeminen kestää useita kymmeniä sekunteja ja sen takia viive on liian suuri, että sensoria voitaisiin käyttää reaaliaikaiseen valvomiseen, ettei CNC-jyrsintä käytettä esimerkiksi väärillä asetuksilla.

Reaaliaikaisuuden vaatimus voitaisiin täydentää kahdella sensorilla, joilla olisi yhteiset väylät muutoin paitsi CE (Chip Enable). Silloin ensimmäisen sensorin DIO1 (Digital Input/Output 0) pinni voitaisiin yhdistää toisen sensorin DIO1 pinniin. Ensimmäisellä sensorilla DIO1 pinnin arvo merkitsisi vastakkaista kuin toisessa sensorissa eli jos ensimmäisen sensorin DIO1 pinnin arvo 0 merkitsisi kiireistä, tarkoittaisi se toisessa sensorissa vapaata.

DIO1 pinniin voitaisiin sitten ohjata signaali, joka kertoo, onko sensorilla mittaus käynnissä. Kun ensimmäinen sensori menisi tilaan, ettei se voi tallentaa, asettuisi toisen sensorin DIO1 pinni tilaan, joka käynnistäisi tallennuksen. Tällöin datavirta värinästä olisi jatkuvaa. Tällaisen mittausjärjestelyn ongelmaksi muodostuu kuitenkin tarkkuus. Koska molemmat sensorit eivät voi olla täsmälleen samassa paikassa, on niiden havainnoima tärinäkuvio myös hieman erilainen. Erilaiset tärinäkuviot aiheuttavat hankaluuksia analyysiin, koska virhearvojen marginaalia joudutaan silloin kasvattamaan, että virheelliset ilmoitukset oikeasta datasta saadaan karsittua minimiin.

Tässä työssä sensorin ja Raspberry Pi:n toiminta yhdessä oli sujuvaa ja mielestäni tällainen yhdistelmä olisi toimiva, jos Raspberry Pi –tietokone saisi virtansa helposti laitteelta, jota mitataan. Muutoin virtajohdon sijoittelu vaikeuttaa yhdistelmän käyttöä ja todellisuudessa ei voida puhua oikeasta IoT-laitteesta, koska se ei toimi patteri- tai

akkukäyttöisesti. Kehitysalustana Raspberry Pi on erittäin hyvä, mutta lopullisen sovelluksen valmistaminen Raspberry Pi- tietokoneen avulla ei olisi järkevää.

Tätä kokoonpanoa voitaisiin kehittää reaaliaikaiseksi järjestelmäksi useammalla sensorilla, jotka mittaisivat vuorollaan laitteen tärinää. Tästä työstä saatua dataa täytyisi käsitellä vielä, siten että voitaisiin luoda automaatiojärjestelmä, joka tunnistaisi milloin laitetta käytetään väärillä asetuksilla ja estäisi silloin käytön automaattisesti. Lisäksi usean sensorin käyttöön liittyvä ongelma tärinäkuvioiden erilaisuudesta täytyisi ratkaista käyttämällä sopivaa algoritmia, joka laskee useista pisteistä jonkinlaisen referenssituloksen.

4. YHTEENVETO

Tässä työssä on käyty läpi teoriaa, jota tarvitaan LTE-tekniikkaa hyödyntävissä sovelluksissa. Teorian lisäksi tässä työssä on esitelty esineiden Internetin toimintaa ja sen tuomia mahdollisuuksia sekä ongelmia, joita IoT-laitteiden käytössä tulisi huomioida. Näistä tärkeimpinä on IoT-laitteiden tietoturva ja mitä ongelmia huonosti suojatut IoT-laitteet voivat aiheuttaa.

Lopuksi tässä työssä on esitelty Raspberry Pi –tietokoneen ja ADIS16227 värinäsensorin avulla toteutettu IoT-sovellus CNC-jyrsimen tärinän mittaamiseen. Esittelyssä on kerrottu käytetyt asetukset, sekä mitä tulisi huomioida, jos tätä sovellusta yrittää itse toteuttaa.

Tämän työn pohjalta voitaisiin kehittää kattavampaa mittausjärjestelmää käyttäen useampaa sensoria ja saadun datan käsittelyä. Tulevaisuudessa automatisoitu käytön seuranta voisi olla mahdollista, jos mittausjärjestelmä saadaan toimimaan reaaliaikaisesti.

5. LÄHTEET

[1] Hekkala A. (luettu 4.3.2017), 5GTN Overview URL: http://5gtn.fi/overview/.

[2] Dahlman E., Parkvall S. & Skold J. (2016) 4G, LTE Evolution and the Road to 5G, Third Edition. Place of publication not identified: Elsevier Science and Technology Books, Inc.

[3] Cox C. (2012) An Introduction to LTE: LTE, LTE-Advanced, SAE, and 4G Mobile Communications. Chichester, West Sussex, U.K.: John Wiley & Sons.

[4] Korowajczuk L. (2011) LTE, WIMAX, and WLAN Network Design, Optimization and Performance Analysis. Chichester, West Sussex, U.K.: John Wiley & Sons.

[5] Hänninen T. (2017), Haastattelu.

[6] Donegan P. (luettu 24.3.2017), IPSec Deployment Strategies for Securing LTE Networks URL: http://go.radisys.com/rs/radisys/images/paper-seg-ipsec-deployment.pdf.

[7] Statistic Brain Research Institute. (luettu 11.4.2017), Instagram Company Statistics URL: http://www.statisticbrain.com/instagram-company-statistics/.

[8] Haltian. (luettu 12.4.2017), Thingsee One URL: https://thingsee.com/thingsee-one.

[9] Kyung C. (2015) Smart Sensors for Health and Environment Monitoring. In: Anonymous KAIST Research Series. Dordrecht: Springer: 281-306.

[10] RS Components. (luettu 12.4.2017), 11 Internet of Things (IoT) Protocols You Need To Know About URL: https://www.rs-online.com/designspark/eleven-internet-of-things-iot-protocols-you-need-to-know-about.

[11] Medvedev A., Fedchenkov P., Zaslavsky A., Anagnostopoulos T. & Khoruzhnikov S. (2015) Waste Management as an IoT-Enabled Service in Smart Cities. In: Balandin S, Andreev S & Koucheryavy Y (eds) Internet of Things, Smart Spaces, and Next Generation Networks and Systems. : 104-115.

[12] Khandelwal S. (luettu 26.6.2017), Friday's Massive DDoS Attack Came from Just 100,000 Hacked IoT Devices URL: http://thehackernews.com/2016/10/ddos-attack-mirai-iot.html.

[13] LLC I. (luettu 11.4.2017), Blockchain URL: http://www.investopedia.com/terms/b/blockchain.asp;.

[14] Analog Devices. (luettu 13.4.2017), URL: http://www.analog.com/media/en/technical-documentation/data-sheets/ADIS16227.pdf.

6. LIITTEET

Liite 1 spi.h header-tiedosto

Liite 2 reg_def.h header-tiedosto

Liite 3 cntrl_src.c lähdekoodi

Liite 4 main.c pääohjelma

Liite 1. spi.h header-tiedosto

```
#ifndef INC_SPI_H_
#define INC_SPI_H_

#ifdef __cplusplus
extern "C" {
#endif

#include <time.h>
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <getopt.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>
#include "reg_def.h"

#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))

struct axes {
    int16_t x;
    int16_t y;
    int16_t z;
};

void initSPI(int *fd, const char *device, uint8_t mode, uint8_t bits, uint32_t speed);
void writeSPI(int fd, uint8_t *msg);
uint16_t readSPI(int fd, uint8_t *msg);
void sendToAzure(struct axes *data_ptr);
int16_t acceleration(uint16_t value, int range);
void recordingSettings(int fd);
void startRecording(int fd);
void readBuffers(int fd, struct axes *data_ptr);
void delay (unsigned int howLong);

#ifdef __cplusplus
}
#endif

#endif /* INC_SPI_H_ */
```

Liite 2. reg_def.h header-tiedosto

```
// -------------- ADIS16227 Buffers --------------------------------//

// Define Three Data Buffers
short int X_DATA[512];              // 16 bit
short int Y_DATA[512];              // 16 bit
short int Z_DATA[512];              // 16 bit

// Define Three FFT Average Buffers (accumulate FFTs)
unsigned short X_FFT[256];        // 16 bit
unsigned short Y_FFT[256];        // 16 bit
unsigned short Z_FFT[256];        // 16 bit

// -------------- ADIS16227 User Registers -------------------------//
enum user_regs
{
   ENDURANCE          = 0x00,
   X_NULL             = 0x02,
   Y_NULL             = 0x04,
   Z_NULL             = 0x06,

   REC_FLSH_CNT       = 0x08,
   SUPPLY_OUT         = 0x0A,
   TEMP_OUT           = 0x0C,
   FFT_AVG            = 0x0E,
   BUF_PNTR           = 0x10,
   REC_PNTR           = 0x12,
   X_BUF              = 0x14,
   Y_BUF              = 0x16,
   Z_BUF              = 0x18,
   REC_CNTR           = 0x1A,
   REC_CTRL           = 0x1C,
   REC_PRD            = 0x1E,

   ALM_F_LOW          = 0x20,
   ALM_F_HIGH         = 0x22,
   ALM_X_MAG1         = 0x24,
   ALM_Y_MAG1         = 0x26,
   ALM_Z_MAG1         = 0x28,
   ALM_X_MAG2         = 0x24,
   ALM_Y_MAG2         = 0x26,
   ALM_Z_MAG2         = 0x28,
   ALM_PNTR           = 0x30,
   ALM_S_MAG          = 0x32,
   ALM_CTRL           = 0x34,
   DIO_CTRL           = 0x36,
   GPIO_CTRL          = 0x38,
   dummy0             = 0x3A, // 2 bytes
```

```
    DIAG_STAT           = 0x3C,
    GLOB_CMD            = 0x3E,

    ALM_X_STAT          = 0x40,
    ALM_Y_STAT          = 0x42,
    ALM_Z_STAT          = 0x44,
    ALM_X_PEAK          = 0x46,
    ALM_Y_PEAK          = 0x48,
    ALM_Z_PEAK          = 0x4A,
    TIME_STAMP          = 0xFFFF,   // 32 bit time stamp, in seconds
    dummy1              = 0xFF,     // reserved, 2 bytes
    REV                 = 0x52,
    DATE                = 0x53,
    MONTH               = 0x54,
    YEAR                = 0x55,
    PROD_ID             = 0x56,
    SCRATCH_A           = 0x58,
    SCRATCH_B           = 0x5A,
    dummy2              = 0xFF,     // reserved, 20 bytes
    ALM_X_FREQ          = 0x70,
    ALM_Y_FREQ          = 0x72,
    ALM_Z_FREQ          = 0x74,
    REC_INFO            = 0x76
};
```

Liite 3. cntrl_src.c lähdekoodi

```
#include "spi.h"

static void pabort(const char *s) {
   perror(s);
   abort();
}

void initSPI(int *fd, const char *device, uint8_t mode, uint8_t bits, uint32_t speed) {
   // Initialize SPI, first open SPI and then set up mode, bits per word and max speed
   int ret;
   uint8_t tx[4] = {0};

   *fd = open(device, O_RDWR);
   if (*fd < 0)
      pabort("Can't open device");

   // SPI mode
   ret = ioctl(*fd, SPI_IOC_WR_MODE, &mode);
   if (ret == -1)
      pabort("Can't set SPI mode");

   ret = ioctl(*fd, SPI_IOC_RD_MODE, &mode);
   if (ret == -1)
      pabort("Can't get SPI mode");

   // Bits per word
   ret = ioctl(*fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
   if (ret == -1)
      pabort("Can't set bits per word");

   ret = ioctl(*fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
   if (ret == -1)
      pabort("Can't get bits per word");

   // Max speed Hz
   ret = ioctl(*fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
   if (ret == -1)
      pabort("Can't set max speed Hz");

   ret = ioctl(*fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
   if (ret == -1)
      pabort("Can't get max speed Hz");

   // Restore factory register settings and clear the capture buffers
   tx[2] = GLOB_CMD | 0x81; tx[3] = 0x00;
   writeSPI(*fd, tx);
   delay(0.020);
```

```
    tx[2] = GLOB_CMD | 0x80; tx[3] = 0x08;
    writeSPI(*fd, tx);
    delay(81);
    // Software reset
    tx[2] = GLOB_CMD | 0x81; tx[3] = 0x00;
    writeSPI(*fd, tx);
    delay(0.020);
    tx[2] = GLOB_CMD | 0x80; tx[3] = 0x40;
    writeSPI(*fd, tx);
    delay(55);
    // Offset correction
    tx[2] = GLOB_CMD | 0x81; tx[3] = 0x00;
    writeSPI(*fd, tx);
    delay(0.020);
    tx[2] = GLOB_CMD | 0x80; tx[3] = 0x01;
    writeSPI(*fd, tx);
    delay(681);
}

void writeSPI(int fd, uint8_t *tx) {
/*
 *
 *  A single register read requires two 16-bit SPI cycles. The first sequence sets
 *    R/W = 0 and communicates the target address (Bits 15 to 8). Bits 7 to 0 are
 *    don't care bits for a read DIN sequence. Second 16-bit SPI cycle sends the
 *    actual message to register.
 *
 */
    int ret;
    static uint8_t bits = 8;
    static uint32_t speed = 1000000;
    static uint16_t delay_u;

    uint8_t rx[ARRAY_SIZE(tx)] = {0, };

    struct spi_ioc_transfer tr = {
        .tx_buf = (unsigned long)tx,
        .rx_buf = (unsigned long)rx,
        .len = ARRAY_SIZE(tx),
        .delay_usecs = delay_u,
        .speed_hz = speed,
        .bits_per_word = bits,
    };

    ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
    if(ret < 1)
        pabort("Cant send SPI message");
}
```

```
uint16_t readSPI(int fd, uint8_t *tx) {
   // Similar to writeSPI but this function returns the value that sensor returns
   uint16_t value;
   int ret;
   static uint8_t bits = 8;
   static uint32_t speed = 1000000;
   static uint16_t delay_u;

   uint8_t rx[ARRAY_SIZE(tx)] = {0, };

   struct spi_ioc_transfer tr = {
      .tx_buf = (unsigned long)tx,
      .rx_buf = (unsigned long)rx,
      .len = ARRAY_SIZE(tx),
      .delay_usecs = delay_u,
      .speed_hz = speed,
      .bits_per_word = bits,
   };

   ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
   value = (rx[0] << 8) | rx[1];

   if(ret==-1)
      pabort("Can't read SPI");
   return value;
}

void sendToAzure(struct axes *data_ptr) {
   // Send data from data_ptr to Azure, now this only prints data to Terminal
   printf("x axis: %d mg\ny axis: %d mg\nz axis: %d mg\n\n", data_ptr->x, data_ptr-
>y, data_ptr->z);
}

int16_t acceleration(uint16_t value, int range) {
   /* These calculations are for 0g to 70g range, change the coefficient n depending
on what range you are using.
    *
    * Range:
    * 1: 0g to 1g
    * 2: 0g to 5g
    * 3: 0g to 20g
    * 4: 0g to 70g
    */

   int16_t data, n;
   if(range == 1) {
      n = 0.0305;
   }
   else if(range == 2) {
```

```
    n = 0.1526;
}
else if(range == 3) {
    n = 0.6104;
}
else if(range == 4) {
    n = 2.3842;
}
else {
    n = 0;
}

switch(range) {
    case 1: {
        if(value < 0x8013) {
            data = value * n;
        }
        else if(value > 0x8012 && value <= 0xFFFF) {
            data = -(0xFFFF - value + 1) * n;
        }
        else {
            data = 0;
        }
        break;
    }
    case 2: {
        if(value < 0x8000) {
            data = value * n;
        }
        else if(value > 0x7FFF) {
            data = -(0xFFFF - value + 1) * n;
        }
        else {
            data = 0;
        }
        break;
    }
    case 3: {
        if(value < 0x8000) {
            data = value * n;
        }
        else if(value > 0x7FFF) {
            data = -(0xFFFF - value + 1) * n;
        }
        else {
            data = 0;
        }
        break;
    }
```

```
        case 4: {
            if(value < 0x72B0) {
                data = value * n;
            }
            else if(value > 0x8D4F && value <= 0xFFFF) {
                data = -(0xFFFF - value + 1) * n;
            }
            else {
                data = 0;
            }
            break;
        }
        default: {
            pabort("Incorrect range");
        }
    }
    return data;
}

void recordingSettings(int fd) {
    // Set according table 10 in ADIS16227 datasheet, these control settings for re-
cording data
    uint8_t tx[4] = {0};
    tx[2] = REC_CTRL | 0x81; tx[3] = 0x02;
    writeSPI(fd, tx);
    delay(0.020);
    tx[2] = REC_CTRL | 0x80; tx[3] = 0x32;
    writeSPI(fd, tx);
    delay(0.020);
}

void startRecording(int fd) {
    // This command starts recording using settings written to REC_CTRL register in
recordingSettings() function
    uint8_t tx[4] = {0};
    tx[2] = GLOB_CMD | 0x80; tx[3] = 0x00;
    writeSPI(fd, tx);
    delay(0.020);
    tx[2] = GLOB_CMD | 0x81; tx[3] = 0x08;
    writeSPI(fd, tx);
    delay(42.15);
}

void readBuffers(int fd, struct axes *data_ptr) {
    // Read data for x-, y- and z-axis, every measurement has 256 samples for every
axis.
    uint8_t tx[4] = {0};
    uint8_t value;
    tx[2] = X_BUF; tx[3] = 0x00;
```

```
    writeSPI(fd, tx);
    delay(0.020);
    value= readSPI(fd, tx);
    data_ptr->x = acceleration(value, 4);
    delay(0.020);

    tx[2] = Y_BUF;
    writeSPI(fd, tx);
    delay(0.020);
    value= readSPI(fd, tx);
    data_ptr->y = acceleration(value, 4);
    delay(0.020);


    tx[2] = Z_BUF;
    writeSPI(fd, tx);
    delay(0.020);
    value= readSPI(fd, tx);
    data_ptr->z = acceleration(value, 4);
    delay(0.020);
}



void delay (unsigned int howLong) {
    // Delay in milliseconds
  struct timespec sleeper, dummy ;

  sleeper.tv_sec  = (time_t)(howLong / 1000) ;
  sleeper.tv_nsec = (long)(howLong % 1000) * 1000000 ;

  nanosleep (&sleeper, &dummy) ;
}
```

Liite 4. main.c pääohjelma

```
#include "spi.h"

int main() {
   int i, fd;
   struct axes *data_ptr, *tmp;
   static const char *device = "/dev/spidev0.0";
   static uint8_t mode = SPI_MODE_3;
   static uint8_t bits = 8;
   static uint32_t speed = 1000000;        // 1 MHz

   i = 0;

   tmp = calloc(1, sizeof(struct axes));
   if(tmp != NULL) {
      data_ptr = tmp;
      initSPI(&fd, device, mode, bits, speed);
   }

   while(tmp != NULL) {
      recordingSettings(fd);
      startRecording(fd);
      readBuffers(fd, data_ptr);
      sendToAzure(data_ptr);
      i++;
      memset(data_ptr, 0, sizeof(struct axes));
   }

   if(tmp != NULL) {
      free(data_ptr);
   }
}
```