VAKAUTTAVAN KAMERAJALUSTAN

MÄÄRITTELY JA SUUNNITTELU

Mikko Yli-Viikari

Opinnäytetyö
Marraskuu 2014
Sähkötekniikka
Älykkäät koneet

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Sähkötekniikan koulutusohjelma
Älykkäät koneet suuntautumisvaihtoehto

YLI-VIIKARI, MIKKO
Vakauttavan kamerajalustan määrittely ja suunnittelu

Opinnäytetyö 102 sivua, joista liitteitä 46 sivua
Marraskuu 2014

Tässä opinnäytetyössä tehtiin vaijeria pitkin kulkevalle kamerajalustalle tuotesuunnittelun alkuvaiheita.

Määrittelyssä selvitettiin suunniteltavan laitteen tärkeimmät ominaisuudet ja määriteltiin minkä ominaisuuden toteuttaminen oli kaikista haasteellisinta. Nämä ominaisuudet toimivat suunnittelua ohjaavina tavoitteina.

Suunnitteluosuus keskittyi laitteiston ja järjestelmän ohjauksen suunnitteluun. Kahdesta kamerajalustan osasta tehtiin 3D – tuloste ja arvioitiin niiden hyviä ja huonoja puolia. Tulostamisen jälkeen arvioitiin mikä olisi voitu tehdä paremmin tulosteen toteutuksessa. Opinnäytetyössä etsittiin karkea tapa, jolla voidaan arvioida tulostettavan kappaleen liikuttamiseen vaadittava vääntömomentti.

Laitteen ohjausta käsittelevässä osassa pohditiin sopivaa ratkaisua ohjauksen toteuttamiselle. Ohjaukselle tehtiin prototyyppikoodeja Arduinon mikro-ohjaimella. Ohjauksen suunnittelu keskittyi kameran vakauttamiseen, sillä se oli laitteen tärkein ominaisuus. Koodin eri osat käytiin läpi ja tutkittiin mitä ominaisuuksia niillä on.

Opinnäytetyö ei tuottanut valmista tuotetta, vaan jäi keskeneräiseksi ajan puutteen takia. Prototyyppikoodit onnistuttiin tekemään kamerajalustan perustoimintoja varten ja 3D-printtauksen toimivuutta prototyyppien testaamiseen.

Asiasanat: tuotesuunnittelu, kuvaus, Arduino piirilevy, servomoottorin ohjaus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Bachelor of Electrical Engineering
Option of Intelligent Machines

YLI-VIIKARI, MIKKO
Stabilizing camera stand's definition and design

Bachelor's thesis 102 pages, appendices 46 pages
November 2014

In this bachelor's thesis was made for making early steps of wire moving camera stand's product design.

In definition explained designed device's most important properties and which of these properties were most difficult to implement. These properties were objectives to guide designing.

Design part focused on hardware's and system's control designing. Two camera stands parts were made with 3D-printing and those parts advantages and disadvantages were evaluated. After printing it was evaluated what could have been done better during printing. In thesis was searched a rough way to evaluate how much torque was required to move printable object.

In devise's control section pondered fitting solution to accomplish it. Prototype codes were made with Arduino micro-controller. Designing of control was focused on cameras stability which was devise's most important feature. Different parts of code were reviewed and researched what kind of properties they have.

The bachelor's thesis did not produce finished product but it was incomplete because of lack of time. The prototype codes was created for basic functions for camera stand and test 3D-printers suitability for testing functionality of prototypes.

Key words: product design, photography, Arduino circuit board, servomotor control

SISÄLLYS

TIIVISTELMÄ 2

ABSTRACT 3

1. JOHDANTO 6

2. LÄHTÖKOHDAT 6

3 LAITTEEN MÄÄRITTELY 6

3.1 Laitteen toiminta ja tavoite 7

3.2 Markkinoilla olevat sovellukset 7

3.3 Ammattikuvaajan näkemys laitteen ominaisuuksista 8

3.3 Suunnittelun määrittelevät ominaisuudet 9

4 KAMERA 9

5 RUNGON 3D-SUUNNITTELU ja 3D-PRINTTAAMINEN 11

5.1 3D-mallinnusohjelma Autodesk Inventor Professional 2013 12

5.2 3D-suunnittelun aloittaminen 12

5.3 Ensimmäinen tulostaminen 13

5.4 Toinen tulostaminen 16

6 KUVAN VAKAUTTAMINEN 18

7 Sähkömoottorin KARKEA MITOITTAMINEN 20

7.1 Servojen vääntömomentti merkinnät 24

8 VAKAUTTAMISEN OHJAAMINEN 24

9 KOODAAMINEN ARDUINO MIKRO-OHJAIMELLA 26

9.1 Koodaamisen harjoittelu Arduinolla 27

9.2 Koodaamiseen liittyvää sanastoa 27

9.3 Arduinon siirtoyhteyden testaaminen 29

9.4 Anturidatan lukeminen 32

9.5 Datan uudelleen skaalaaminen 35

9.6 Muuttujan arvon suodattaminen funktiolla 37

9.7 Sähkömoottorin käyttö Arduinolla 41

9.7.1 Servon kytkeminen Arduinoon 42
9.7.2 Eri servotyypin käyttäytyminen Arduinossa 43
9.7.3 Servon ohjaus Arduinolla 43
10 Kauko-ohjaaminen 46
10.1 Langattomanohjauksen ongelmat 48
11 Aktiivisen vakauttamisen toteuttaminen Arduinolla 48
11.1 Gyron kytkeminen Arduinoon 49
12 Kamerajalustan suunnitelun lopputulos 53
LÄHTEET 54
LIITTEET 56
Liite 1. GY-521 mpu6050 datan lukeminen ja kolmen servon ohjaaminen 56

1. JOHDANTO

Tämän opinnäytetyön tarkoituksena on tehdä automaattisesti vakauttavan kamerajalustan tuotesuunnittelun alkuvaiheita ja tehdä määritelmä laitteen ominaisuuksista ja mahdollisista haasteista.

Työssä tutustutaan myös Arduino Unon käyttöön laitteen ohjaajana, tehdään koodit laitteen perustoimintoja varten ja selitetään miten koodit toimivat. Lisälaitteiden kytkemiseen Arduino Unoon käsitellään myös ja kerrotaan miten kytkeminen onnistuu.

Työssä kerrotaan myös prototyypin 3D-tulostamisesta ja sen eri vaiheista. Tulostamisyritysten onnistumiset ja epäonnistumiset käydään läpi.

2. LÄHTÖKOHDAT

Idea opinnäytetyöstä tuli TAMK Televisio ja Elokuva koulutusohjelman opiskelijalta Tatu Tuomiselta keväällä 2013. Tuominen kysyi, kuinka vaikeaa on toteuttaa vaijeria pitkin kulkeva ja kauko-ohjattava kamerajalusta. Keskustelun pohjalta lähdin tutustumaan Internetin avulla jo olemassa oleviin järjestelmiin. Toteuttaminen osoittautui haasteelliseksi, mutta juuri sopivaksi oman koulutussuuntautumisen opinnäytetyöksi.

Tein ensimmäisen suunnitelman opinnäytetyön sisällöstä ja listasin projektiin liittyviä haasteita, jonka jälkeen esittelin idean Sähkötekniikan koulutuspäällikölle Jarkko Lehtoselle. Hän hyväksyi aiheen ja tavoitteeksi asetettiin laitteen rakentaminen niin pitkälle kuin mahdollista.

3 LAITTEEN MÄÄRITTELY

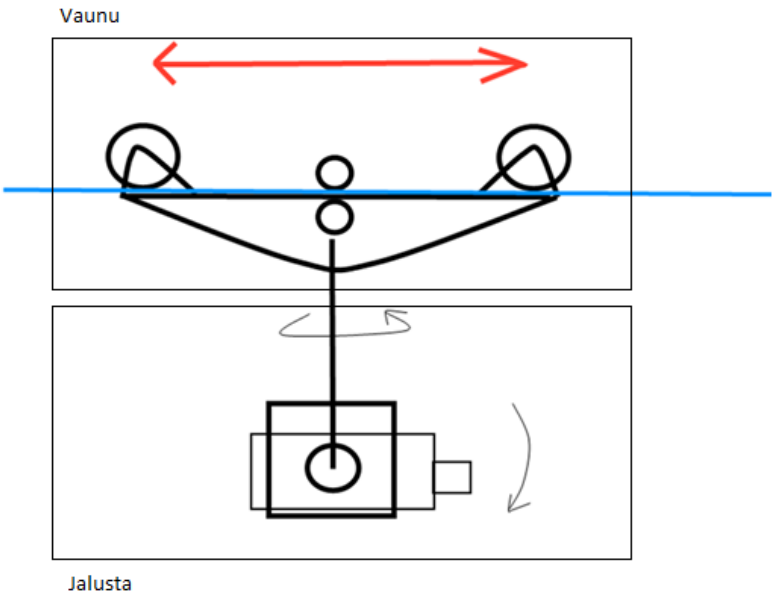
Ennen varsinaisen suunnittelun tehtiin laitteen määrittely. Määrittelyssä arvioitiin, mitkä ovat tuotekehityksen tavoitteet ja mitä vastaavanlaisia sovelluksia on jo olemassa sekä alettiin pohtiamaan tuotteen tärkeimpiä ominaisuuksia ja mahdollisia haasteita. Näin luotiin kokonaiskuva siitä, mitä kaikkea tuotteen kehittäminen pitää sisällään. Tarkoituksena ei ollut tässä vaiheessa mennä tarkkoihin yksityiskohtiin, jotta hahmottaminen olisi helppompaa.

Määrittelyyn voidaan palata myöhemmin tuotekehityksen aikana. Esimerkkinä tilanteessa, jossa tuotekehitysprojektia esitellään rahoituksen hankinnan yhteydessä, on selkeä määritelmä on hyvä työkalu.

3.1 Laitteen toiminta ja tavoite

Laitetta ohjataan kauko-ohjauksella ja laite kulkee vaijeria pitkin. Laitteistoon kiinnitetään kamera, jolla kuvataan haluttua kohdetta. Suunniteltu laitteisto pitää kameran vakaana ja mahdollistaa kameran liikkeen kuvattavan kohteen mukana.

Laite voidaan rakenteellisesti jakaa kahteen osaan eli vaunuun ja jalustaan. Vaunulla tarkoitetaan laitteiston osaa, joka ripustaa laitteiston vaijerille ja sisältää sähkömoottorit, jotka liikuttavat laitetta vaijerilla. Jalustalla tarkoitetaan laitteiston osaa, johon kamera kiinnitetään ja pyritään vakauttaa mahdollisimman hyvin. Jalustaan kiinnitetään myös vakauttamiseen tarvittavat komponentit. Kuvassa 1 on esitetty periaatekuva laitteistosta ja esitetään edellä kuvaillut osat.



KUVA 1. Periaatekuva laitteen osista

Kameran kiinnitystapa valitaan sellaiseksi, että mahdollisimman moni kameramalli saadaan kiinnitettyä laitteistoon. Suunniteltavan järjestelmä on erillään kamerasta. Kameralla on oma virtalähde. Järjestelmä ei säädä kameran polttoväliä ja kamera ei käynnisty tai sammu kauko-ohjauksella. Näin kameraan ei tarvitse tehdä muutoksia.

Kameran kohdistaminen tehdään myös langattomaksi, joten käyttäjä voi ohjauspisteeltään nähdä, mihin kamera on kohdistettu. Langattomassa kuvansiirrossa käytetään markkinoilta löytyviä valmiita ratkaisuja.

3.2 Markkinoilla olevat sovellukset

Opinnäytetyön aihe ei ole uniikki idea. Monet yritykset ovat jo tuoneet markkinoille omat laitteensa, joiden avulla voidaan kuvata lintuperspektiivistä. Tätä

kuvausmenetelmää käytetään erityisesti urheilutapahtumien ja konserttien kuvaamiseen. Määrittelyä tehdessä on hyvä kerätä inspiraatiota kirjallisuudesta ja nettisivuilta valmiista tuotteista. Tämä auttaa selvittämään haasteita ja ominaisuuksia, joita laitteessa tulisi olla.

Itävaltalaisen Jens C. Petersin on kehittänyt Spidercam® -järjestelmässä. Spidercamin® koostuu kamerajalustasta, johon on kiinnitetty 4 nostokaapelia. Kaapelit on kiinnitetty vinsseihin, joilla ohjataan kameraa kuvausalueen ilmatilassa. Tässä järjestelmässä tarvitaan kaksi käyttäjää. Toinen käyttäjä ohjaa jalustan liikettä ja kun taas toinen käyttäjä on vastuussa kuvauksesta.

Spidercam®:n antamien teknisten tietojen mukaan voidaan olettaa, ettei tämä kamerajalusta ratkaisu sovellu lyhyisiin kuvausprojekteihin. Koko järjestelmää varten täytyy Spidercam® antamien vaatimuksien mukaan varata paljon tilaa ja taata riittävä jännite. Lisäksi vinssejä vaativat massaksi kerrottiin Spidercam®:n nettisivuilla olevan n.350 kg. Näiden tietojen perusteella voidaan arvioida tämän kamerajalustaratkaisun olevan kallista.

3.3 Ammattikuvaajan näkemys laitteen ominaisuuksista

Määrittelyvaiheessa käytiin keskustelu ammattikuvaajan kanssa kuvauslaitteista. Jos suunnittelijalla ei ole omaa kuvaamiskokemusta, voi suunnitteluvaiheessa jäädä huomioimatta asioita, jotka ovat kuvaajasta tärkeitä.

Otin yhteyttä Tommi Moilaseen, joka toimi opinnäyteyötä tehtäessä nuorempana kuvauksen opettaja TAMK:n Televisio ja Elokuva koulutusohjelmassa. Moilanen kertoi, ettei Spidercam® kaltaiset laitteet ole yleisiä suomalaisessa elokuvatuotannossa ja hänellä ei ole omia kokemuksia laitteistosta. Moilanen kertoi kyseisen järjestelmän olevan käytössä lähinnä urheilutapahtumien kuvaamisessa.

Moilanen kertoi kuvauslaitteistoille asetettuja vaatimuksia, jotka pitää toteutua hyvän kuvaustuloksen saavuttamiseksi. ”On tärkeää, että kameralla voidaan kuvata liikkeen aikana, eikä liikkeelle lähtö ja pysähtyminen aiheuta tärähdystä kuvassa. Joskus luullaan, että leikkaaja ottaa kohtauksesta pois ne osat, joissa kamera lähtee liikkeelle tai pysähtyy. Joskus kuvaaja tai ohjaaja haluaa kuitenkin kameran pysähtyvän ja jatkavan liikettä ilman leikkausta.” kertoo Moilanen.

Haastattelun pohjalta voidaan päätellä, että kameran vakauttaminen ja tarkka ohjaaminen ovat kuvaajalle tärkeitä ominaisuuksia.

3.3 Suunnittelun määrittelevät ominaisuudet

Määrittelyn ja asiantuntijan kanssa käydyn keskustelun pohjalta tehdään tiivistelmä tärkeistä ominaisuuksista. Laitteisto:

- vakauttaa kameran kuvan automaattisesti ja laitteisto ei häiritse kuvausta
- kulkee vaijeria pitkin
- on kauko-ohjattu.

Nämä määritelmät toimivat suunnittelun tavoitteina ja pyritään toteuttamaan. Opinnäytetyö keskittyy kuitenkin vakauttamisen toteuttamiseen , koska tämä ominaisuus on kaikista haastavin ja tärkein ominaisuus.

4 KAMERA

Suunnittelun alussa valittiin sopiva kamera, jolle jalusta ja vaunu suunnitellaan. Kamerajalustaan pitäisi määritelmän mukaan pystyä asentamaan eri mallisi kameroita..

Sain ehdotuksen käyttää Sonyn HDW-790P kameraa, joka on yleisessä käytössä TAMK:n Televisio ja Elokuva koulutusohjelmassa. HDW-790P on esitetty kuvassa 2.

KUVA 2. Sonyn HDW -790P (Sony 2014)

Sonyn nettisivujen mukaan kamera painaa ilman lisävarusteita n 5,4 kg ja vaatii 12 VDC (*volts of direct current*) toimiakseen. Kameraan sopisi kiinnittimeksi Sonyn VCT-14 kolmijalka adapteri (Sony 2014). Tähän voisi kiinnittää myös muita kameroita, joissa on sama kiinnitystapa. Adapteri kiinnitetään yksinkertaisella ruuviliitoksella, joka olisi tehnyt adapterin kiinnityksen suunnittelun helpommaksi. Lisäksi TAMK omisti tuolloin kameran, jota olisi voitu lainata testaamista varten.

Ongelma Sonyn kameran käytössä on sen massa. Jos tätä kameraa olisi käytetty, täytyisi tukimateriaalit ja osia liikuttelevat moottorit mitoittaa vahvemmiksi. Tämä olisi nostanut prototyyppien hintaa.

Adapterin ja kameran korkea hinta olivat myös ongelma. Opinnäytetyötä tehdessä HDW-790P:n hinta oli arviolta n. 50 000 € (CPV, B&H, 2014). VCT-14 hinta oli n. 60 € (Ebay 2014).

Suuri ja kallis kamera olisi tehnyt testaamisesta hankalaa. Suunnittelun helpottamiseksi ja kulujen pienentämiseksi päädyttiin käyttämään Aiptek:in valmistamaa 720P HD-1 -kameraa. Kamera on esitetty kuvassa 3.

KUVA 3. Opinnäytetyössä käytettävä kamera (Solid Signal 2013)

Kameran ulkomitoiksi mitattiin 102 x 30 x 70 mm (korkeus x paksuus x leveys). Punnituksessa kameran massa on n.150g.. Koska kamera on kevyt, ei sitä liikuttelevien ja tasapainottavien moottoreiden tarvitse olla kovin voimakkaita. Kameraa tukevat rakenteet voidaan myös pitää kevyinä, joten servojen kokonaiskuorma olisi mahdollista pitää pienenä. Kameran hinta oli n. 20 € (Amazon 2014). Kamera ei sovellu ammattilaiskäyttöön huonon kuvan- ja äänenlaadun takia, mutta sitä käyttämällä vakauttamisen toteutuksien testaaminen on helpompaa.

Kameroiden kiinnittämisessä käytetään yleisesti kahta standardoitua tuuma ruuvikokoa. Ruuvit ovat halkaisijaltaan 1/4’’(≈ 0,635 cm) ja 3,8’’(≈ 0,953 cm). Aiptek 720P HD-1 kamera käyttää pienempää ”varttituuma” kokoista ruuvia kiinnityksiin. Tämä on merkittävä tieto sillä kameran kiinnitys voidaan suunnitella yksinkertaiseksi ja kiinnittää Aiptekin kameran kanssa samankokoisia kameroita.

On tärkeää, että käyttäjä näkee, mihin kameran linssi osoittaa käytön aikana. Opinnäytetyön laajuuden kannalta ei ole järkevää suunnitella alusta alkaen uutta tapaa kuvan seurannalle. On järkevämpää käyttää jo olemassa olevaa järjestelmää. Varsinainen kuvausmateriaali tallentuu kameran muistikortille, joten kuvan laadun seurantamonitorissa ei tarvitse olla todella terävä.

Kuvan lähettäminen voidaan tehdä joko langattomasti tai kiinteällä yhteydellä. Kiinteässä yhteydessä kuva siirretään kaapelia tai johtoa pitkin monitorille. Yhteys ei katkea helposti ja komponentit ovat yksinkertaiset. Aiptek 720P HD-1:ssä on HDMI (*High-Definition Multimedia Interface*) liitäntämahdollisuus. Kaapeliratkaisu ei kuitenkaan ole halpa ratkaisu. Mikäli kuvaa halutaan seurata HDMI-laadulla, täytyy käyttää sopivaa kaapelia. Opinnäytetyötä tehtäessä 25m HDMI –kaapelia maksoi 99 € (Kaapelikauppa, 2014). Mikäli kuvanlaatu muuttaa huonommaksi laaduksi, kuten VGA (*Video Graphics Array)* tai RAC (*Radio Corporation of America)*, täytyy hankkia muunnin tai adapteri, jotka puolestaan täytyy sijoittaa laitteeseen(Kaapelikauppa 2014).

Suurimmaksi ongelmaksi kaapelin käytössä tulisi sen hankaluus käytön aikana. Kaapeli tarvitsee oman järjestelmän, jotta kaapeli olisi jatkuvasti sopivan kireällä ja eikä sotkeutuisi käytön aikana. Kaapelia voidaan käyttää kuvan seurantaan testaamisen aikana.

Langattomassa kuvan siirtämisessä on etuna se, että kuvan seuranta ei rajoita laitteiston liikkumista. Jos halutaan siirtää kuvaa kaukaa ja hyvällä laadulla kuvansiirtojärjestelmän hinta nousee. Toinen toivottava ominaisuus lähettimelle olisi pieni koko ja pieni virrankulutus.

Kuvan seurannan ratkaiseminen ei kuitenkaan ratkaista tässä vaiheessa suunnittelua. Kun kameran vakauttaminen ja ohjaaminen on saatu toimimaan kuvan seurannan ratkaisuja ei kannata pohtia enempää, koska markkinoilta löytyvän ratkaisun käyttäminen on tavoite.

5 RUNGON 3D-SUUNNITTELU JA 3D-PRINTTAAMINEN

Opinnäytetyön yhteydessä tehtiin alustavia suunnitelmia jalustan rungosta. Tämän jälkeen kokeiltiin 3D-printtaamisen toimivuutta ja kannattavuutta prototyypin tuottamisessa.

Suunnitellessa runkoa pyrittiin saavuttaa mahdollisimman yksinkertainen ratkaisu, jotta osien hankkiminen ja mahdollinen valmistaminen olisi helppoa. 3D-suunnittelussa tutkittiin jo olemassa olevia ratkaisuja jalustan muodossa.

5.1 3D-mallinnusohjelma Autodesk Inventor Professional 2013

Autodesk Inventor Professional 2013 on mekaaniseen suunnitteluun valmistettu CAD-ohjelma (*Computer-Aided Design*), jolla on mahdollista tehdä 3D-malleja ja tuottaa näistä malleista erilaisia dokumentteja esimerkiksi kokoonpanokuvia, kappaleen räjäytyskuvia ja mittakuvia.

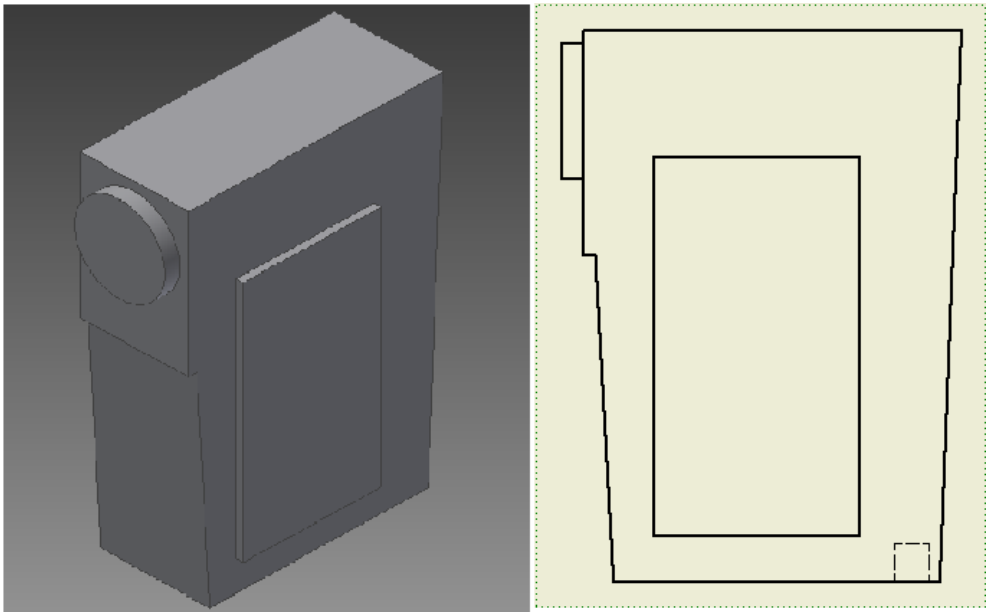
Lisäksi CAD-ohjelman avulla pystytään asettamaan kappaleille ominaisuuksia kuten valmistusmateriaali, jota voi käyttää apuna kappaleen massan arvioinnissa. Ohjelmalla voi myös kokeilla suunniteltavien osien liikeratoja. Ohjelman käyttö helpotti sähkömottoreiden mitoittamista, sillä ohjelmalla voitiin määrittää painopisteiden sijainti ja etäisyys pyörimisakselista.

Autodesk Inventor Professionalin avulla voi myös muuttaa mallinnetun kappaleen tiedostotyypin STL-muotoon (*Stereolithography*). Tämä tiedosto voidaan siirtää 3D-tulostusohjelmaan ja tuottaa tuloste.

Ohjelmisto oli omasta mielestäni helppo käyttää yksinkertaisten muotojen suunnitteluun. Törmätessäni ongelmiin suunnitellessani laitetta ja kysyessäni apua TAMK:n henkilökunnalta tai koulun ulkopuoliselta ammattilaiselta, kuvan esittäminen laitteesta auttoi ongelman selvityksessä paljon.

5.2 3D-suunnittelun aloittaminen

Ensimmäisenä mallinnettiin vakauttamisen kohde eli kuvassa 3 näkyvä Aiptek 720P HD-1. Kamerasta otettiin mahdollisimman tarkat mitat ja tehtiin kamerasta vastaava malli Inventorilla. Kuvassa 4 on 3D-malli Aiptekin kamerasta ja läpivalaisu, josta näkyy kiinnitysreiän sijainnin.



KUVA 4. 3D-malli Aiptek 720P kamerasta

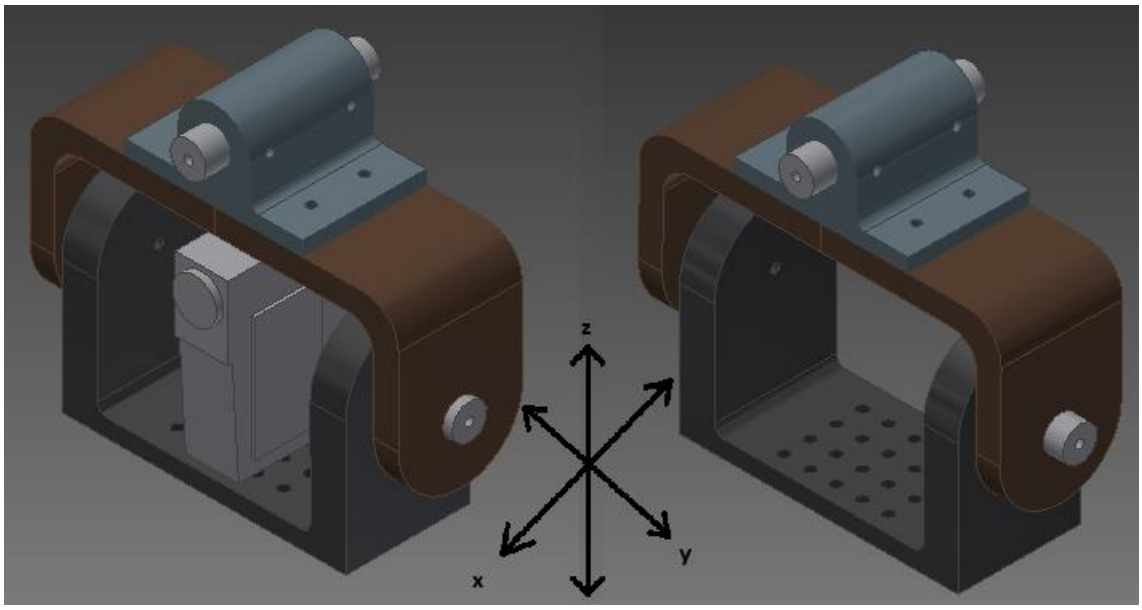
Tärkeimmät mitat mallissa oli kiinnitysreiän koko, kameran korkeus ja leveys. Nämä mitat olivat tärkeät, koska kameran muoto määriteltiin laskelmissa suorakulmaiseksi särmiöksi sähkömoottoreiden mitoituksen aikana. Mallista oli myös helppo tarkastaa kameran mitat tarvittaessa.

Jalustasta rungosta tuotettiin kaksi prototyyppiä kahdella eri 3D-tulostimella. Ensimmäinen prototyyppi valmistettiin TAMK:n tiloissa ja Stratasyksen Dimension Elite tulostimella. Toisen prototyypin osia valmistettiin Ultimaker ja RepRapPro Mendel mono 3D-tulostimilla Tampere Hacklab:in tiloissa.

5.3 Ensimmäinen tulostaminen

Sopivaa ratkaisua jalustan rungolle ei löytynyt heti vaan jouduttiin tuottamaan useita malleja. Usein lähestymistapa hylättiin liian monimutkaisen muodon takia. Kaikista mallinnuskokeiluissa löytyi kuitenkin hyviä ominaisuuksia, jotka otettiin käyttöön seuraavissa malleissa.

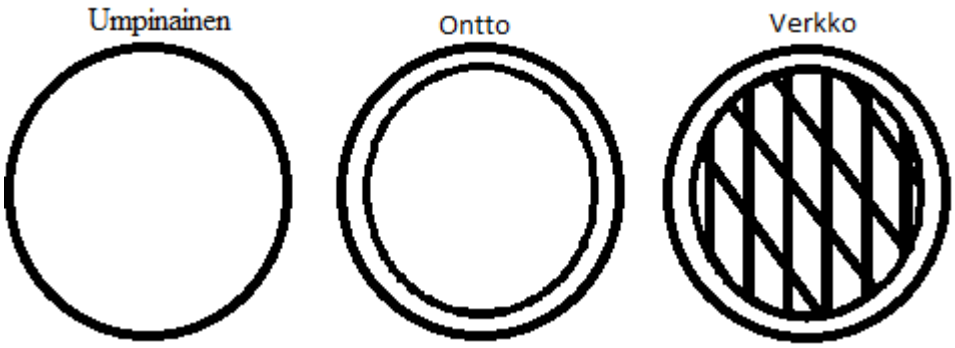
Ensimmäinen prototyypin valmistuksessa haluttiin testata 3D-tulostamisen toimivuutta ja tarkastella, kuinka tarkasti tuote tulostetaan. Samalla testattiin sen hetkisen kamerajalustamallin toimivuutta käytännössä. Kuvassa 5 on jalustan 3D-malli kameran kiinnitysjalustasta, josta tehtiin 3D-tuloste. Vasemmalla puolella kuvaa 5 jalustaan on kiinnitetty Aiptekin kamera ja oikealla puolella kamera on jätetty pois.



KUVA 5. Ensimmäisen tulostetun prototyypin 3D-malli

Jalusta on muodoltaan kehtomainen. Pohjalevyssä on reikiä, jolloin kameran voi kiinnittää pohjalevyyn useasta kohdasta. Tällä tavalla voidaan tasapainottaa jalustaa siirtelemällä kameran painopistettä. Kahden akselin avulla oli tarkoitus vakauttaa jalusta x ja y-akselin ympäri tapahtuva heiluntaliike. Tulostettavat osat oli tarkoitus kiinnittää toisiinsa M5 ruuveilla ja kamera jalustaan 1/4’’ruuvilla. Kuvan 5 edeltävä versiossa ei ollut pyöristyksiä. Nämä lisättiin malliin, jotta säästettäisiin tulostusaineen kulutuksessa.

Akseleita lukuun ottamatta kaikki osat piti valmistaa 3D-tulostimella. TAMK:n tulostin käytti tulostusaineena ABS-muovia. Alkuperäinen idea oli tehdä tulosteista umpinaisia sisältä, mutta tämä ei ollut tarpeen. Stratasyksen tulostimella on mahdollista tehdä malleista onttoja eri seinän paksuuksilla, verkkomaisia ja umpinaisia sisältä, samaan tapaan kuin kuvassa 6 on esitetty.



Kuva 6. 3D-tulosteen täyttömahdollisuuksia

Täyttötavaksi tulosteeseen valittiin verkkomainen täyttö. Umpinainen rakenne olisi käyttänyt huomattavasti enemmän tulostusmateriaalia kuin verkkomainen rakenne. Onton rakenteen pelättiin olevan liian hauras ja epävakaa. Tarkkuutta ei myöskään asetettu kaikista korkeimmalle tasolle, sillä se olisi lisännyt entisestään tulostusaikaa.

Akselit valmistettiin muovitangoista. Akseleiden muovityyppiä ei voitu määrittää tarkasti, koska niiden valmistamiseen käytettiin kierrätyspisteen muovia. Kaikkia osia ei saatu tulostettua kerralla, koska käytössä oli vain 20x20x20 cm alue.

Prototyypin valmistamiseen kului n. 32 h. Mallin pyöristysten ja verkkomaisen rakenteen ansiosta mallista tuli kevyempi kuin aluksi suunniteltiin.

Tulostusprosessin lopussa käytettyä lipeää pääsi tulosteen sisään pienistä rei'istä. Kesti noin viikon ennen kuin tuloste lakkasi tihkumasta lipeää sormille käsittelyn aikana. Kuvassa 7 on ensimmäinen prototyyppi kasattuna.

KUVA 7. Ensimmäinen 3D-tulosteprotyyppi (Kuva: Mikko Yli-Viikari 2014)

Tämän jälkeen alettiin arvioida suunnitellun tulosteen toimivuutta. Osat olivat tukevia ja kestivät hyvin pientä vääntämistä ja koottuna osat olivat tiukasti kiinni toisissaan. Kaikista rei'istä ei tullut täysin ympyrän muotoisia. Tämän takia tulostettujen osien kiinnittämiseen jouduttiin käyttämään M4 koon ruuveja ja ohentamaan sorvaamalla akselitappeja. Reiät oli kuitenkin oikeissa kohdissa ja kokoaminen onnistui suunnitellusti.

Kameran pystyttiin kiinnittämään suunnitelluilla 1/4''ruuvilla. Osa suorista pinnoista jäi myös kuperiksi. Kuperuus ei kuitenkaan aiheuttanut haittaa mallin toiminnan kannalta.

Osa, johon kamera kiinnitetään jalustassa, mahtui liikkumaan suunnitellusti mutta oli raskas kiertää. Suunnitteluvaiheessa oli unohdettu ottaa huomioon kitkan vaikutus. Testatessa mallia kokeiltiin voidella rasvalla akselitappeja, jotta kameran asentoa voitaisiin liikutella helpommin. Voitelun jälkeenkin osaa oli raskas kiertää sormivoimin. Akseleiden reiät eivät olleet täysin pyöreät. Jotta voitaisiin saavuttaa sulava liikkuvuus, täytyi seuraavissa jalustamalleissa ottaa huomioon laakerointi.

Suurin ongelma tulosteessa oli sen korkea hinta. Tulostamiselle tehtiin laskelma, jossa arvioitiin mm. työtunnit, jotka olisi kulunut TAMK:n henkilökunnalta osan työstämisessä ja materiaalikustannukset. Kaikkiaan tulostaminen olisi maksanut n. 530 €. Korkea hinta johtui ABS -muovin käytöstä, joka on kallis materiaali.

Kaikki osat olivat uniikkeja osia, joten rikkoutuneiden osien tilalle olisi tarvittu uusi tuloste. Uuden tulosteen tuottaminen ABS -muovista olisi ollut kallista ja vienyt aikaa. Seuraavissa malleissa pyritään käyttämään vähemmän 3D-tulostusta.

Tuloste tuotettiin liian aikaisin. Ei ollut vielä ratkaistu mitä sähkömoottoreita käytetään osien liikuttamiseen ja minkälainen voimansiirtoa käytetään. Sähkömoottoreille ja voimansiirrolle ei ollut tilaa tulosteessa.

Tulostaminen oli onnistunut hyvin, mutta kaikki ongelmat olivat heikkouksia suunnitelmassa. Päätettiin, ettei tätä prototyyppiä vietäisi eteenpäin. Testaus kuitenkin toi ilmi ongelmia suunnittelussa, jotka eivät näkyneet 3D-suunnitteluohjelmassa.

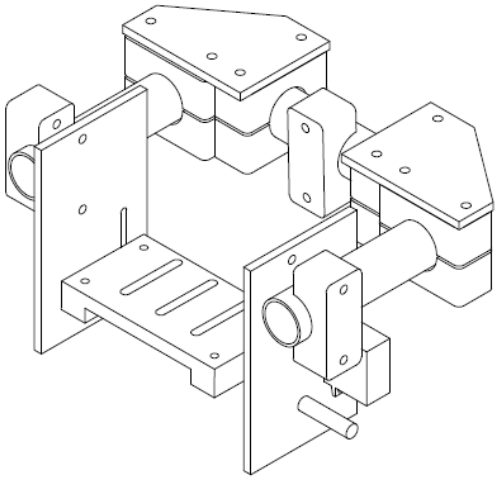
5.4 Toinen tulostaminen

Opinnäytetyön aikana tehtiin toinen kokeilu 3D-tulostimella. Tätä varten tehtiin uusi suunnitelma jalustan rakenteesta. Ainoastaan kameran kiinnitysjalusta tehdään 3D-tulostimella. Muissa osissa pyritään käyttää valmiita osia.

Akselin rakenne, jolla saadaan aikaiseksi pystysuunnassa tapahtuva liike kameralle, vaihdettiin toisenlaiseksi. Edellisessä suunnitelmassa akseli oli kaksi erillistä muovitappia. Uusimmassa suunnitelmassa on tarkoitus käyttää alumiinista yhtenevää akselia. Syy muutokselle on Tampereen Laakerikeskuksen esittämä kehityssuositus jalustan rakenteeseen. Laakerointi valinta ja suunnittelu jäävät opinnäytetyön ulkopuolelle, jotta opinnäytetyö ei laajenisi liikaa.

Tukirakenteet uusimmassa mallissa on tarkoitus tehdä 32 mm halkaisijalta olevasta PP -putkesta. Syy putken käyttöön on sen helppo saatavuus kaupoista ja kohtuullinen hinta.

Lisäksi putken kanssa voidaan käyttää kiinnityksiin valmiita putkiliittimiä. Esimerkiksi Dunlop Hiflexin 30-n532 kiinnittimen avulla voidaan 32 mm PP -putkea kiinnittää haluttuun kohteeseen. Kuvassa 8 on jalustan rakenteen uusin suunnitelma.



KUVA 8. Uusin jalusta suunnitelma

Kuvassa 8 näkyvät levyt täytyy valmistaa itse mutta ne on mahdollista tehdä sopivalla poraus- ja leikkaamistyökaluilla. Opinnäytetyössä muovilevynä testattiin peiliakryyliä (polykarbonaattia). Muovi oli tukevaa ja sitä oli helppo porata ja leikata. Levy on kuitenkin kallista hankkia.

Tulostaminen tehtiin eri tulostimella ja eri materiaalista. Tulostimena käytettiin Tampere Hacklab:in Ultimaker mallista tulostinta. Kappaleen täyttö tehtiin edelleen verkkomaiseksi.

Kappale, joka tulostettiin, oli huomattavasti pienempi kuin ensimmäisellä kerralla. Tulostamiseen kului n. 3 h. Ensimmäinen tuloste oli liian hauras, joten tulostus jouduttiin suorittamaan kahdesti. Syy haurauteen oli väärät asetukset tulostimessa. Tulostaminen maksoi n. 2 € per tuloste. Kulut tulivat pelkästään materiaaleista. Kuvassa 9 on toinen onnistunut 3D-tuloste mitoituksen kohteena olevan kameran kanssa.

KUVA 9. Toinen 3D-tuloste ja Aiptek 720P HD-1 (Kuva: Mikko Yli-Viikari)

Toinen tulostuskerta oli onnistuneempi. Kamera pysyi jalustassa hyvin pelkällä ruuviliitoksella ja tulostuskulut pysyivät alhaisempina. Kameran voi edelleen kiinnittää useammasta kohdata kiinnitysjalustaan painopisteen säätelemiseksi.

Toista prototyyppiä ei saatu valmiiksi kokonaan eikä sen toimivuutta testattu. Syy tähän oli kiireellisyys saada vakauttamisen ohjaus toimimaan. Lisäksi tässä prototyypissä oli tarkoitus käyttää laakerointia akseleilla. Lisäksi voimansiirto tapa sähkömoottoreille oli myös ratkaisematta. Tässä vaiheessa ei ollut myöskään selvää minkälainen anturi havaitsee heilunnan ja mikä olisi anturille paras sijainti jalustassa. Siksi päädyttiin pysäyttämään mekaaninen suunnittelu kunnes nämä asiat päätetyksi.

Mielestäni 3D-tulostamisen sopii hyvin mekaanisten ratkaisujen testaamiseen ennen lopullisten osien teettämistä, jos osat ovat sopivan kokoisia, tarvitaan pieniä määriä tulosteita, tulostaminen ei vie kauan aikaa ja ei ole liian kallista projektin varattuun budjettiin nähden.

6 KUVAN VAKAUTTAMINEN

Tämä on koko järjestelmän tärkein ominaisuus ja kaikista vaikein toteutettava. Tämän osa-alueen suunnittelulle varattiin paljon aikaa. Ongelmaa lähdettiin tutustumalla ihan fysiikan perusteisiin. Kirjallisuus, Internet ja televisio-ohjelmat antoivat paljon esimerkkejä, missä heiluntaa pyritään valokuvauksen lisäksi kompensoimaan.

Videoissa, joissa esitellään eri vakautusjärjestelmiä, näki paljon sovelluksia, joissa vakauttamista tarvitaan. Yritykset eivät kuitenkaan aina paljasta millä tavalla he ovat

vakautuksen toteuttaneet. Eri ratkaisuja tutkiessa jaoin vakauttamisen kahteen pääryhmään:

- aktiiviseen
- passiiviseen vakauttamiseen.

Passiivisella vakauttamisella tarkoitetaan esimerkiksi kamera jalustoja, jotka pelkästään painopistettä säätämällä vakauttavat kameran. Hyvä esimerkki tästä on Steadicam kamerajalustat. Kuvassa 10 on esimerkki Steadicam:in Merlin -mallinen kamerajalusta.

KUVA 10. Steadicam Merlin, passiivisesti vakauttava kamerajalusta (Tiffen. 2014)

Kuvan 9 kamera jalusta ei käytä lainkaan sähköisiä komponentteja tasapainottamisessa. Merlin hyödyntää pienikitkaista palloniveltä ja painopisteen säätöä vakauttamisen saavuttamiseksi. Jalustaan laitetaan lisäpainoja kameran koon ja vakauttamisen tarpeen mukaan. Passiivinen vakauttaminen on jatkuvasti käytössä ja voi vakauttaa vain tiettyyn rajaan asti. Hyvänä puolena tässä vakauttamisessa on se, että tätä vakauttamista voidaan käyttää kauan. Ainoa rajoite on käyttäjän jaksaminen kannatella laitetta. Muita tätä vakauttamistapoja ovat mm. iskunvaimentimet, kuten jouset, tärinää vaimentavat kumitassut ja erilaiset muotoiluratkaisut.

Aktiivisella vakauttamisella tarkoitetaan toteutuksia, joissa vakautusjärjestelmä menee päälle, kun sille on tarvetta. Vakausjärjestelmä havaitsee heilunnan anturin avulla ja aloittaa vakauttamisen moottoreiden avulla. Ympäristön muuttuvat olosuhteet saavat järjestelmän älykkäästi vastaamaan niihin.

Hyvä esimerkki on aktiivisesta vakauttamisesta voi nähdä National Geographic International:in dokumentissa nimeltään Richard Hammond's Engineering Connections HMS Illustrious, jossa esiteltiin lentotukialukseen liittyvää tekniikkaa. Dokumentissa selitettiin, miten lentotukialus pysyy vakaana kovassakin aallokossa, jotta lentokoneet voivat laskeutua tai lähteä alukselta turvallisesti. Lentotukialuksessa käytetään gyroskooppianturia, joka havaitsee kallistumisen ja ohjaa vakauttavia moottoreita kompensoimaan aallokon aiheuttaman heilunnan.

Koska pääsuuntautumiseni TAMK:issa oli ollut Älykkäät koneet ja sähkötekniikka, joten päädyin suosimaan aktiivista vakauttamista suunnittelussa. Pyrin kuitenkin ottamaan huomioon mahdollisuudet passiiviselle vakauttamiselle, kuten painopisteiden säätelylle. Valintaperuste on siis omat vahvuudet ja kokemukset edellisistä projekteista.

Eri vakautusvaihtoehtoja pohdittaessa heräsi kysymys: Kannattaako koko laitteisto vakauttaa vai kohdistetaanko se vain tiettyyn osaan laitetta? Jos laitteessa halutaan käyttää aktiivista vakauttamista, niin se tarkoittaa sähkömoottoreiden käyttämistä. Mitä suurempi kuorma on sitä tehokkaampi, suurempi ja kalliimpi moottori tarvitaan sen liikuttamiseen. Keskittämällä aktiivinen vakauttaminen tiettyyn osaan laitetta voidaan vakautusmoottorit ja muut komponentit jättää pienemmiksi.

Tämä pohdinnan perusteella päädyttiin siihen, että ainoastaan jalustassa on aktiivista vakauttamista.

7 SÄHKÖMOOTTORIN KARKEA MITOITTAMINEN

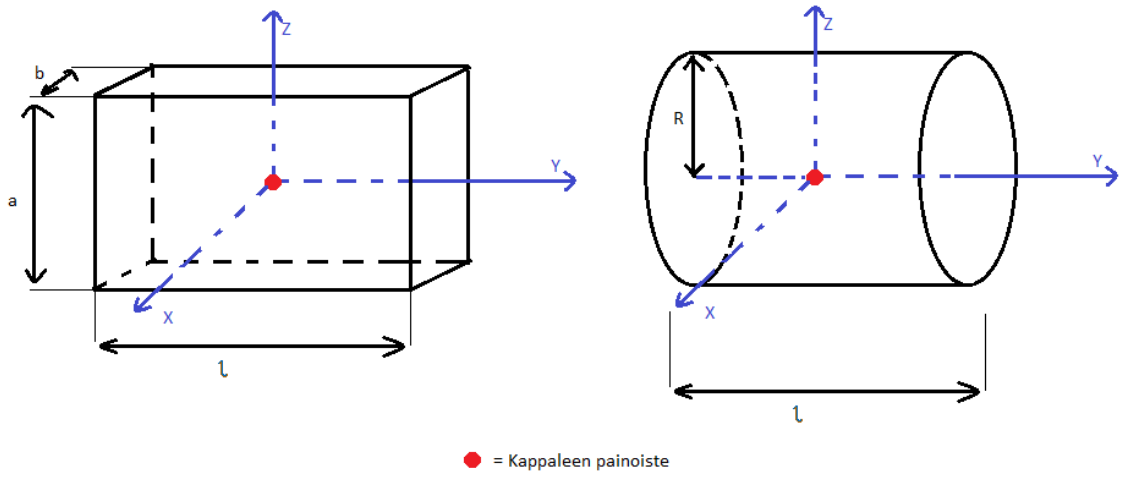
Koska vakauttamisessa tullaan käyttämään sähkömoottoreita, täytyy ne pystyä mitoittamaan. Sähkömoottoreita ei mitoiteta tässä vaiheessa tarkasti, sillä niillä testataan koodin toimivuutta. Moottoreita ei ole tarkoitus optimoida, vaan antaa arvio siitä, kuinka suuri moottorin kuorma on.

Ensimmäisenä määrittelin mikä on sähkömoottorin kuoman massa ja minkä muotoinen kuorma on. Näillä tiedoilla voidaan määrittää hitausmomentti *J*.

Jos kuorma on muodoiltaan monimutkainen, voidaan se jakaa pienempiin osiin ja laskea osille hitausmomentti erikseen. Nämä osat voidaan vuorostaan muokata laskuja varten yksinkertaisimpiin muotoihin kuten suorakulmaisiin särmiöihin ja umpinaisiin lieriöihin. Näille muodoille on laskukaavat valmiina. Palasille määritellään erikseen pyörimisakselin suhteen hitausmomentti. Lopuksi kappaleen osien hitausmomentit

summataan keskenään, jolloin saadaan kuorman pyörittämiseen tarvittava hitausmomentti.

Suorakulmaisella särmiön ja umpinaisen lieriön hitausmomentin laskukaava vaihtelee painopisteeseen sijoitetun pyörimisakseli suunnan mukaan. Kuvassa 11 suorakulmaisen särmiön ja umpinaisen lieriön painopisteeseen on laitettu xyz – koordinaatisto. Lisäksi kuvaan on merkitty mitat, joita tarvitaan hitausmomentin määrittämiseen.



KUVA 11. Särmiön hitausmomentin määrityksen apukuva

Kuvaa 11 voidaan käyttää apuna hahmottamaan tarvittavat kappaleen mitat, jotka tarvitsee tuntea ennen hitausmomentin määritystä. Kaavassa(1) ja (2) on umpinaisen lieriön hitausmomentin määrittämiseen tarvittavat kaavat (Mäkelä ym, 2008. 94).

$$J_{\text{lieriöX}} = J_{\text{lieriöZ}} = m\left(\frac{R^2}{4} + \frac{l^2}{12}\right) \quad (1)$$

$$J_{\text{lieriöY}} = \frac{1}{2}mR^2 \quad (2)$$

missä:

$J_{\text{lieriöX}}$ on lieriön hitausmomentti kun se pyörii x-akselin ympäri

$J_{\text{lieriöZ}}$ on lieriön hitausmomentti kun se pyörii z-akselin ympäri

$J_{\text{lieriöY}}$ on lieriön hitausmomentti kun se pyörii y-akselin ympäri

m on kappaleen massa

R on lieriön säde

l on lieriön pituus

Suorakulmaisensärmiön hitausmomentin voi määrittää kaavoilla (3), (4) ja (5) (Mäkelä ym, 2008. 94).

$$J_{\text{särmiöX}} = \frac{1}{12} m(a^2 + l^2) \quad (3)$$

$$J_{\text{särmiöY}} = \frac{1}{12} m(a^2 + b^2) \quad (4)$$

$$J_{\text{särmiöZ}} = \frac{1}{12} m(b^2 + l^2) \quad (5)$$

missä:

$J_{\text{särmiöX}}$ on suorakulmaisesärmiön hitausmomentti kun se pyörii x-akselin ympäri

$J_{\text{särmiöZ}}$ on suorakulmaisesärmiön hitausmomentti kun se pyörii z-akselin ympäri

$J_{\text{särmiöY}}$ on suorakulmaisesärmiön hitausmomentti kun se pyörii y-akselin ympäri

m on kappaleen massa

a on särmiön korkeus

b on särmiön leveys

l on särmiön pituus

Jakamalla kuorman pienempiin osiin saadaan tarkempi hitausmomentti. Yksinkertaisen muotoisista kappaleista on helpompi määrittää painopisteiden sijainti kappaleissa. Painopisteen avulla voidaan soveltaa Steinerin sääntöä, jolla voidaan lisätä hitausmomentin määrittämisen tarkkuutta.

Kaavat (1)-(5) olettivat, että kappaleen pyörimisakseli kulkee painopisteen kautta. Kappaleen painopiste ei aina sijaitse pyörimisakselilla vaan on siitä tietyn etäisyyden päässä siitä. Tällöin joudutaan soveltamaan Steinerin sääntöä, joka on esitetty kaavassa (6) (Mäkelä ym, 2008. 93).

$$J_A = J_P + md^2 \quad (6)$$

missä:

J_A on hitausmomentti mielivaltaisessa pisteessä A

J_P on hitausmomentti painopisteessä P

m on kappaleen massa

d pisteen A ja painopisteen P etäisyys toisistaan yhden suuntaisilla akseleilla

Kaavalla (6) voidaan lisätä tarkkuutta laskuihin tarvittaessa. Jos etäisyys *d* on kuitenkin pieni ja ei aiheuta suurta muutosta hitausmomentissa, jätettiin tämä tarkastelu pois.

Kuorman pyörittämiseen tarvittavan vääntömomentin *M* määrittämiseksi täytyy valita kuormalle sopiva pyörimisnopeus *n*. Jos ei ole tiedossa valmiiksi tarvittavaa nopeutta, voidaan tarkastelua varten nopeus ”ravistaa hihasta”. Valitsin tarkastelussa olevan servomoottorin pyörimisnopeuden kuorman pyörimisnopeudeksi. Tämän jälkeen määritellään kuormalle kulmanopeus ω ja kulmakiihtyvyys α.

Kulmanopeus voidaan laskea kierrosnopeuden avulla kaavalla (7) (Mäkelä ym, 2008. 92).

$$\omega = 2\pi n \tag{7}$$

missä:

ω on kuorman kulmanopeus

n on kuorman pyörimistaajuus eli pyörimisnopeus

Kulmakiihtyvyys saadaan kaavalla (8) (Mäkelä ym, 2008. 92). Kuorman kiihtymiseen kuluvaksi ajaksi voi valita servon kierrosajan eli ottamalla käänteisluku kierrosnopeudesta *n*.

$$\alpha = \frac{\omega}{t} = \frac{\omega}{\frac{1}{n}} = \omega n \tag{8}$$

missä:

α on kuorman kulmakiihtyvyys

ω on kuorman kulmanopeus

t on kiihtymiseen kulunut aika

n on kuorman pyörimisnopeus

Tämän jälkeen voidaan laskea kaavalla (9) kuinka suuren vääntömomentin kuorma pyörittämiseen tarvitaan (Mäkelä ym, 2008. 93).

$$M = J\alpha \quad (9)$$

missä:

M on kuorman pyörittämiseen tarvittava vääntömomentti

J on kuorman hitausmomentti

α on kuorman kulmakiihtyvyys

Tämä laskutoimitus ei ota huomioon kitkan ja voimansiirron vaikutusta. Koska valmistaja ilmoittaa servon vääntökyvyn useammalla jännitetasolla, valitsin moottorin alhaisemman jännitetason vääntömomentin perusteella. Näin varmistetaan, että moottori jaksaa pyörittää kuormaa.

7.1 Servojen vääntömomentti merkinnät

Servojen datalehdet eivät aina käytä SI-yksiköitä ilmoittaessaan servon vääntömomenttia. Tämä aiheutti hieman hämmennystä etsiessäni sopivaa servoa. Esimerkiksi etsiessäni sopivaa servoa valmistaja ilmoitti vääntömomentin yksikkönä kg*cm. Joskus servoja myyvät nettisivut ilmoittavat vääntömomentin kg/cm, kg-cm tai kg.cm. Momenttiyksikön kg*m voi karkeasti muuttaa Nm:eiksi seuraavasti (Conversion of Measurement Units. 2014):

1 kg*m = 9.81 Nm

1 kg*cm = 10.20 Nm

8 VAKAUTTAMISEN OHJAAMINEN

Käydään läpi, mitä vaatimuksia ohjaukselle on asetettu. Jalustan täytyy pystyä liikuttamaan kameran kuvaa samanaikaisesti horisontaalisti ja vertikaalisesti. Kuvauksessa horisontaalista liikkeestä käytetään nimitystä panorointi ja vertikaalista liikkeestä tilttaus (pystypanorointi) (Videon peruskurssi v2 kuvaustekniikka. 2002). Tilttaus ja panoroinnin täytyy tapahtua sulavasti (Moilanen. 2013). Lisäksi vaunun pitää

liikkua pehmeästi, ettei se häiritse kuvausta. Kun ohjausta ei ole, pyrkii jalusta vakauttamaan itsensä automaattisesti.

Ohjaukselta vaaditaan monen asian hoitamista kerralla ja valitsemaan, onko automatiikka vai käyttäjä ohjaamassa moottoreita. Vaunun moottorit, jotka liikuttavat moottoria vaijerilla, tarvitsevat vain käyttäjän ohjauksen.

Vakausmoottorin ohjaus voidaan jakaa siis 4 pääalueeseen:

- Manuaalinen ohjaus
- Automaattinen ohjaus
- Ohjauksen valinta
- Ohjauksen siirto toimilaitteelle

Jakamalla ohjauksen suunnittelun pääalueisiin on helpompi hahmottaa ohjauksen suunnittelun kokonaisuutena. Ratkaisemalla jalustan ohjaustavan saadaan myös vaunun moottoreiden ohjaamiseen tarvittavat ratkaisut.

Ongelmaksi kamerajalustan suunnittelun etenemiselle oli älykkyyden tuominen laitteella. Ennen opinnäytetyön aloittamista minulla oli vähän kokemusta automaatiosuunnittelusta, ohjelmoitavien logiikoiden tai mikro-ohjainten käytöstä.

Ystävän suosituksesta tutustuin Arduinon valmistamiin mikro-ohjaimiin ja käyttäjien tekemiin projekteihin. Mikro-ohjaimella tarkoitetaan pientä tietokonetta, jolla on prosessori ja muistia. Ohjelmoidulla mikro-ohjaimella voidaan vilkuttaa LED-lamppuja tai ohjata 3D–tulostimen moottoreita (Karvinen, T. Karvinen, K. 2010). Kuvassa 11 on erilaisia Arduinon valmistamia mikro-ohjaimia.

KUVA 12. Erilaisia Arduinon mikro-ohjaimia (Arduino Robotics Projects. 2012)

Arduinon mikropiirejä käytetään paljon harrastekäytössä, koska niillä on halpa hinta suhteessa tehokkaaseen prosessoriin ja pieneen kokoon. Nämä mikro-ohjaimet käyttävät C++:asta kehitettyä ohjelmointikieltä ja ohjelma, jolla koodi tehdään, toimii Windowsin, Mac OS X:n ja Linuxin käyttöjärjestelmissä (Karvinen, T. Karvinen, K. 2010).

Koska TAMK:lla ei ollut opinnäytetyön teon aikana kursseja Arduinosta, opiskelin itsenäisesti sen käyttöä. Pääsääntöisesti tein harjoituksia Arduinon kotisivuilta ja tutustuin muihin mikro-ohjainta projekteihin netissä.

Sain myös paljon neuvoa tamperelaiselta Hackerspace 5w yhteisöltä, kun kohtasin ongelmia harjoitusten aikana. Yhteisön omissa tiloissa oli myös 3D-tulostin, jota käytin toisen prototyyppiosan valmistamiseen.

9 KOODAAMINEN ARDUINO MIKRO-OHJAIMELLA

Aloittaessa koodaamisen harjoittelun ensimmäisenä on hankittava tarkoitukseen sopiva Arduinon piirilevy, USB A-B – kaapeli ja Arduinon kehitysympäristö tietokoneelle. Kehitysympäristöllä tehdään koodit, jotka syötetään mikro-ohjaimeen. Ohjelman voi ladata ilmaiseksi Arduinon kotisivuilta.

Opinnäytetyössä käytettiin Arduino Unoa. Tämä mikro-ohjain tyyppi valittiin, koska monet kokeneemmat käyttäjät suosittelivat sitä. Jos ohjaukseen pystytään käyttämään

pienempää mikro-ohjainta, kuten Arduino Nano, voidaan mikro-ohjain vaihtaa optimoituun ratkaisuun myöhemmissä prototyypeissä.

Suositeltavia hankittavia tarvikkeita mikro-ohjaimen käyttöä varten ovat koekytkentälevy, hyppyjohdot, 12 V ulkoinen jännitelähde ja potentiometri. Koekytkentälevylle on helppo tehdä ja purkaa harjoituskytkentöjä testatessa koodin toimivuutta. Eriväriset hyppyjohdot helpottavat kytkennän tarkastamista ennen käyttöä ja pitävät kytkennän selkeänä. Ulkoinen jännitelähde tarvitaan, jos kytkentä ei saa tarpeeksi virtaa USB-kaapelin kautta. Arduino Uno:on käy keskipositiivinen 12 V AC/DC adapteri. Potentiometriä käytetään paljon Arduinon esimerkeissä.

9.1 Koodaamisen harjoittelu Arduinolla

Kehitysympäristöstä löytyy paljon valmiita esimerkkejä eri tarkoituksiin tehdyistä koodeista. Nämä koodit ovat kategorioitu ja ovat vapaasti käytettävissä kaikissa projekteissa.

Asetin muutaman tavoitteen aloittaessani koodauksen harjoittelun:

1 Tutustu kehitysympäristöön ja perusteisiin

2 Saa mikro-ohjain lukemaan anturidataa

3 Saa mikro-ohjain ohjaamaan sähkömoottoria

4 Saa mikro-ohjain suorittamaan useaa tehtävää samanaikaisesti

5 Toteuta kauko-ohjaus mikro-ohjaimella

Opinnäytetyössä keskitytään jalustan ohjauksen suunnitteluun sen monimutkaisuuden takia.

9.2 Koodaamiseen liittyvää sanastoa

Arduino-ohjelmissa koodit koostuvat muuttujista (*variables*), vakioista (*constants)* ja funktioista (*functions*). Muuttujiin ja vakioihin tallennetaan tietoa. Ero muuttujan ja vakion välillä on se, ettei vakion arvoa voi muuttaa määrittämisen jälkeen. Muuttujan arvo voi muuttua ohjelman aikana. Ennen kuin muuttujaa tai vakiota voidaan käyttää koodissa, täytyy sille tehdä määrittely, jossa määritellään muuttujan tyyppi, nimi ja arvo. Alla on esimerkki kahden eri muuttujan ja yhden vakion määrittämisestä:

```
int example = 13;
```

```
long example2 =14;

const example3 = 15;
```

Muuttujan tyyppi määräytyy siihen tallennettavasta datasta ja sen tarvitsemasta tallennustilasta. Esimerkiksi int -tyyppinen muuttuja (*integer*) voi tallettaa 16 bittiä (2 tavua) dataa. Jos muuttujan tyyppi on long, se käyttää 32 bittiä (4 tavua) muistia (Arduino int. 2014; Arduino long. 2014).

Muuttujat voidaan jakaa myös globaaleihin (*global*) ja paikallisiin (*local*) muuttujiin. Globaalit muuttujat on määritelty kaikissa osissa koodia, joten niitä voi käyttää kaikkialla koodissa. Paikalliset muuttujat vaikuttavat vain tietyissä osissa koodia kuten yksittäisissä funktioissa, joka on rajattu aaltosulkeilla ({…}). Ero näiden muuttujien välillä on missä osassa koodia ne määritellään. Globaalit muuttujat eivät ole minkään aaltosulkeen sisällä, jolloin ne ovat ”ulkona” kaikista funktioista(Arduino Variables. 2014; Arduino Const. 2014).

Kun muuttuja on määritelty, voidaan sitä käyttää koodissa. On mahdollista käyttää muuttujan nimeä koodissa numero arvon sijaan. Alla on esimerkki, jossa määritellään muuttujalle example uusi arvo:

```
example = example +1;
```

Sama määrittely numeroarvoilla:

```
example = 13+1;
```

Funktioiden on tarkoitus tehdä koodaajaan määräämiä tehtäviä koodissa. Esimerkiksi koodissa usein toistuvalle laskutoimitukselle kannattaa tehdä funktio. Suoritettuaan toimintansa funktio palauttaa arvon siihen kohtaan koodia, jossa sitä oli kutsuttu (Arduino Functions. 2014). Arduinon kotisivuilla on kattava ohje funktioiden määrittämisestä:

http://arduino.cc/en/Reference/FunctionDeclaration

Arduinon koodi koostuu aina kahdesta pääfunktiosta:

```
void setup(){…}

void loop (){…}
```

Ohjelman alussa suoritettava setup()-funktion tehdään vain kerran. Tässä osassa koodia voidaan tehdä alustukset koodille. Esimerkiksi määritellään mitkä mikro-ohjaimen pinneistä toimii sisääntuloina tai ulostuloina.Kun setup()-funktio on suoritettu, siirtyy koodi automaattisesti loop()-funktioon. Kun funktio viimeinen koodirivi on käyty läpi, siirtyy ohjelma automaattisesti loop()-funktion alkuun ja aloittaa koodirivien lukemisen ensimmäiseltä riviltä. Setup()- ja void()-funktiot määritellään void-tyyppisiksi funktioiksi, sillä ne eivät palauta mitään arvoja ja eivät vastaanota parametreja (Karvinen, T. Karvinen, K. 2010).

Lisäksi koodeissa voi nähdä kommentteja (*comments*), joita on koodin tekijä lisännyt. Kommenteilla voi selittää koodin toimintaa lukijalle. Tämä helpottaa työskentelyä koodin parissa pitkän tauon jälkeen. Kun kirjoitettu koodi siirretään mikro-ohjaimeen, kommentit eivät siirry mukana. Kommenttien tekeminen ei siis vie tilaa mikro-ohjaimen muistista. Kommentit näkyvät kehitysympäristössä harmahtavana tekstinä. Kommentteja voi kirjoittaa koodiin esim.:

```
// Loppu rivi tästä eteenpäin on kommenttia

/* Kaikki merkit ja rivit näiden merkkien välissä ovat kommenttia*/
```

Joskus Arduinon koodissa käytetään kirjastoja (*Libraries*), joilla voidaan laajentaa koodin käyttömahdollisuuksia. Kirjastoja käytetään yleensä kun mikro-ohjaimeen kytketään toimilaite tai halutaan käsitellä dataa(Arduino Libraries. 2014). Esimerkiksi servo-kirjasto mahdollistaa RC-servojen ohjaamisen Arduinolla (Arduino Servo Library. 2014).

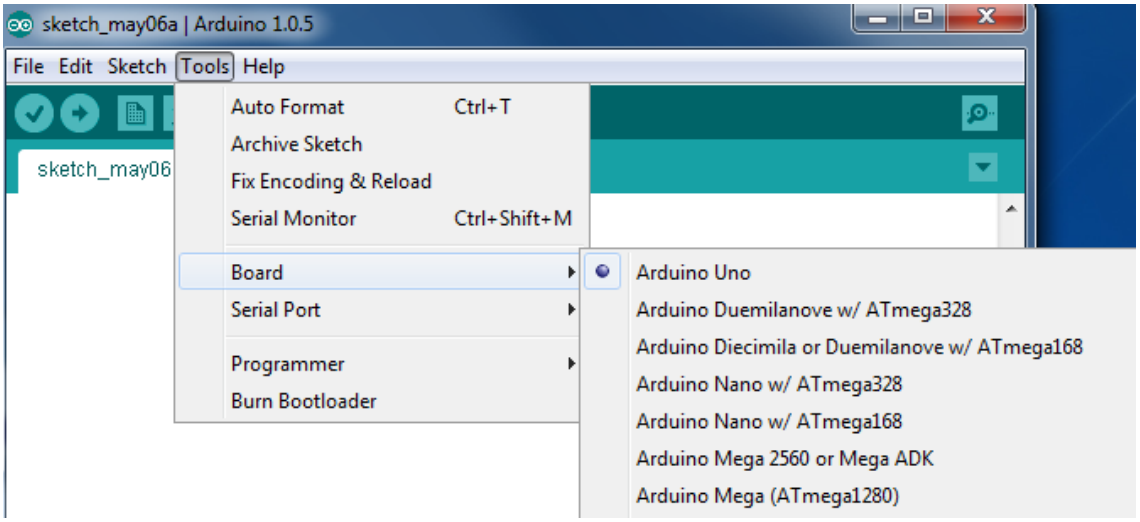
9.3 Arduinon siirtoyhteyden testaaminen

Kun alkaa työskennellä Arduino mikro-ohjaimella kannattaa testata, toimiiko koodin lataaminen ja poistaa vanhat koodit mikro-ohjaimesta. Kun ohjelman lataa mikro-ohjaimeen, ei sitä pysty enää helposti tarkastelemaan samassa muodossa kuin kehitysympäristössä (Arduino forums CC. 2014). Jos on epävarma, mikä ohjelma mikro-ohjaimessa on viime käytön jälkeen, kannattaa poistaa vanhat koodit ennen työskentelyn aloittamista.

Joskus ohjelman latauksessa tapahtuu ongelmia, mutta niihin ei aina ole syynä rikkoutunut piirilevy. Yleisiä ongelmia koodin siirrossa syntyi kun:

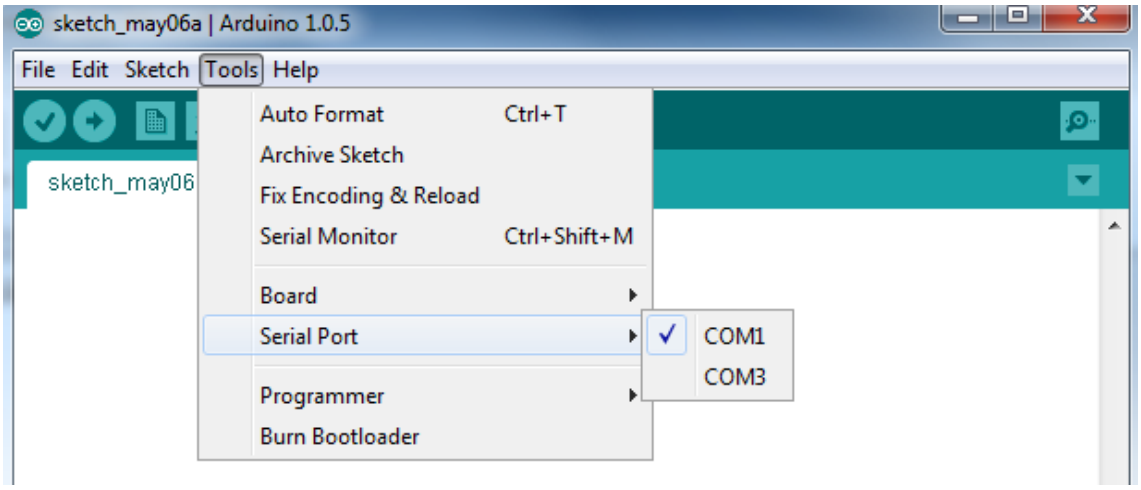
- kehitysympäristössä oli valittuna väärä mikro-ohjain tyyppi
- koodissa on kirjoitusvirhe
- mikro-ohjain on kiinni väärässä USB-portissa
- tietokone ei tunnistanut Arduinon mikro-ohjainta
- mikro-ohjain ei saanut tarpeeksi virtaa

Ennen ohjelmien lataamista voi tarkistaa kehitysympäristön työkalut-valikosta (*tools)*, että valittuna on oikea mikro-ohjain tyyppi (kuva 13).



KUVA 13. Arduino kehitysympäristössä mikro-ohjaintyypin valinta

Samasta työkalut-valikosta voi tarkistaa myös, mistä USB -porttista ohjelma etsii mikro-ohjainta (kuva 14).



KUVA 14. Arduino kehitysympäristössä USB-portin valinta

Windows käyttöjärjestelmissä voi tarkistaa tunnistaako tietokone mikro-ohjainta lainakaan laitehallinnassa (*Device Manager*) kohdassa portit (*ports)*. Mikro-ohjainten ajurien asentaminen auttaa tähän ongelmaan. Vastaava tilanne tapahtui myös kun mikro-ohjain ei saanut tarpeeksi virtaa koodin lataamisen yhteydessä. Kun mikro-ohjaimeen lataa uuden ohjelman, kannattaa kytkeä virta pois kaikista laitteista, jotka on kytketty mikro-ohjaimeen.

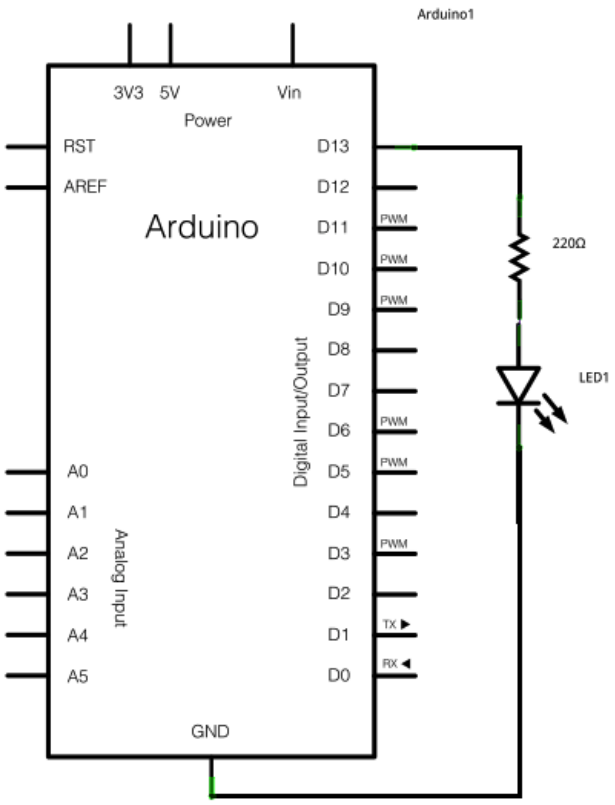
Hyvä tapa poistaa vanha koodi mikro-ohjaimesta on ladata siihen Arduinon tekemän koodiesimerkin nimeltä ”Blink”. Koodin on yksinkertainen ja sen siirtäminen ei kestä kauan. Kuvassa 15 on kyseinen koodi.

Blink

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

KUVA 15. Arduinon Blink esimerkin koodi

Tällä koodilla voi varmistaa, että siirtoyhteys tietokoneen ja mikro-ohjaimen välillä on kunnossa. Samalla pyyhitään mikro-ohjaimessa ennestään olevat ohjelmat. Blink saa Arduino Unossa pinni 13 alla olevan pienen LED:in vilkkumaan sekunnin välein. Kaikissa Arduinon mikro-ohjaimissa ei ole LED:iä. Kytkemällä pinniin 13 LED ja 220 Ω vastuksen sarjaan kuvan 16 mukaisesti, voidaan tehdä sama tarkastuskytkentä (Arduino Blink. 2014).



KUVA 16. Arduinon Blink esimerkin kytkentä (Arduino Blink. 2014)

(Lähdeviite Arduino Blink. 2014. Www -sivu. Luettu 6.5.2014. http://arduino.cc/en/tutorial/blink)

9.4 Anturidatan lukeminen

Arduinon Uno:ssa on 16 digitaalista pinniä, jotka voi toimia sekä sisään että ulostuloina ja 6 analogista sisääntulo pinniä. Molemmille pinnityypeille on mahdollista määrittää joko sisääntulo tai ulostulotila käyttämällä komentoa pinMode(). Digitaaliset pinnit Arduinon mikro-ohjaimissa ovat oletusarvoisesti sisääntuloina, jolloin niiden tilaa ei tarvitse aina määrittää koodissa (Arduino Digital Pins. 2014). Koodin lukemisen helpottamiseksi merkitsin aina tarvittaessa pinnin tilan koodiin. Digitaalista dataa luetaan komennolla pulseIn(). Komennon avulla piiriohjain lukee digitaalisen pulssin keston mikrosekunteina. Tämä komento mahdollistaa siis pulssileveysmoduloidun (PWM, *Pulse-Width Modulation*) signaalin lukemisen Arduinon mikro-ohjaimilla (Arduino pulseIn(). 2014).

Arduinon kotisivut eivät kuitenkaan suosittele muuttamaan analogista pinniä ulostuloksi. Komento analogRead(), jolla luetaan analogista signaalia, ei toimi oikein vaan antaa virheellistä dataa. Tilanmäärittäminen analogiselle pinnille ei ole siis tarpeen. Lukema, jonka analogRead() antaa on jännitearvo välillä 0-5 V, joka on

skaalattu välille 0-1023 (Arduino Analog Input Pins. 2014; Arduino analogRead(). 2014).

Arduino Unon saa lukemaan analogista ja digitaalista sisääntuloa kuvassa 17 esitetyllä koodilla. Koodin pohjana toimi Arduinon tekemä esimerkki ”ReadAnalogVoltage”.

Peruspohja_ja_pinnin_luku §

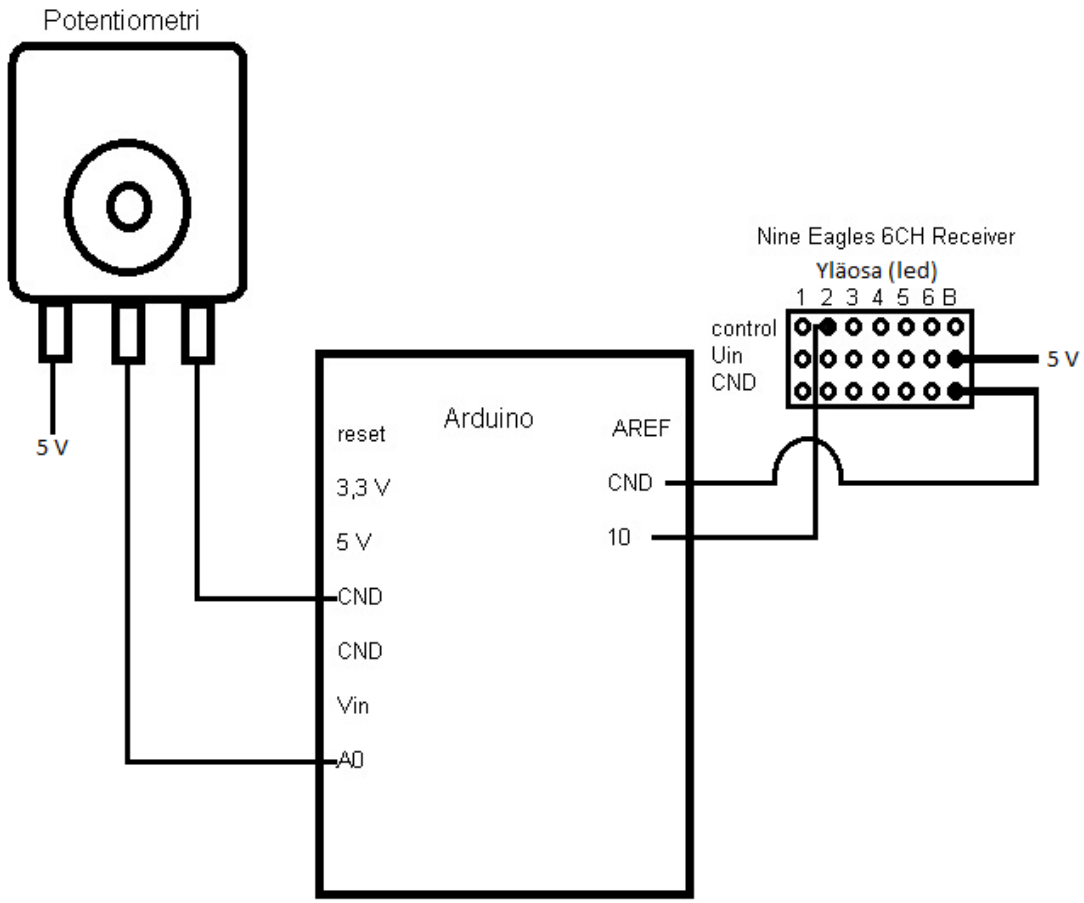
```
/*Koodissa käytettävät kirjastot*/
//Ei käytössä

/*----------------------------------------*/
/*Koodissa käytettävät globaalit muuttujat*/

int digi = 10; //Muuttujan digi luetaan pinnistä 10
int analog; //Muuttujaan analog talletetaan analoginen arvo
/*----------------------------------------*/
/*Koodin alustaminen*/
void setup()
{
  Serial.begin(9600); //Määritellään millä nopeudella dataa siirretään tietokoneelle
  pinMode (10, INPUT); //Määritellään, että pinni 10 on sisääntulo

}

/*Koodin varsinaiset toiminnot*/
void loop()
{
  /*Pinnien arvojen lukeminen*/

  digi = pulseIn(10,HIGH,25000); //Pinnin 10 arvo luetaan ja tallennetaan muuttujaan
  analog = analogRead(A0); //Pinni A0 arvo luetaan ja tallennetaan muuttujaan

  /*Pinnien lukemien tulostaminen Serial monitoriin*/
  Serial.print("Pinnin 10 arvo ");
  Serial.print("\t"); // Lisää tabulaattorin serial printtiin
  Serial.print(digi);
  Serial.print("\t"); // Lisää tabulaattorin serial printtaukseen

  Serial.print("Pinnin A0 arvo ");
  Serial.print("\t"); // Lisää tabulaattorin serial printtiin
  Serial.print(analog);

  Serial.println(""); // Aloittaa seuraavan loobin arvojen kirjoittamisen uudelle riville
  delay(10); // Tauko millisekunteina kunnes alkaa uusi looppi
}

/*Koodissa käytettävät funktiot*/
//Ei käytössä
```

KUVA 17. Digitaalisen ja analogisen sisääntulon lukemisen mahdollistava koodi

Opinnäytetyötä tehdessä opin, että mikro-ohjaimen vastaanottamat signaalit kannattaa aina tarkistaa Serial Monitorilla ennen niiden käyttämistä muissa osissa koodia. Tarkastamiseen on voi käyttää kuvassa 17 näkyvää tulostuskomentoja.

Asettamalla tabulaattorit tulosteeseen saadaan tulosteesta kehitysympäristön Serial Monitor -toiminnossa, mutta myös tekee helpommaksi vastaanotettujen signaaliarvojen siirtämisen Windowsin Excel-taulukko-ohjemaan, jossa signaalista voidaan tehdä kuvaajia. Kuvaajista on helpompi hahmottaa signaalin muutos kuin nopeasti vaihtuvista numeroarvoista Serial Monitorissa.

Kuvan 17 koodia voi kokeilla kuvassa 18 olevalla kytkennällä.



KUVA 18. Sisääntulojen lukemisen testaamiskytkentä

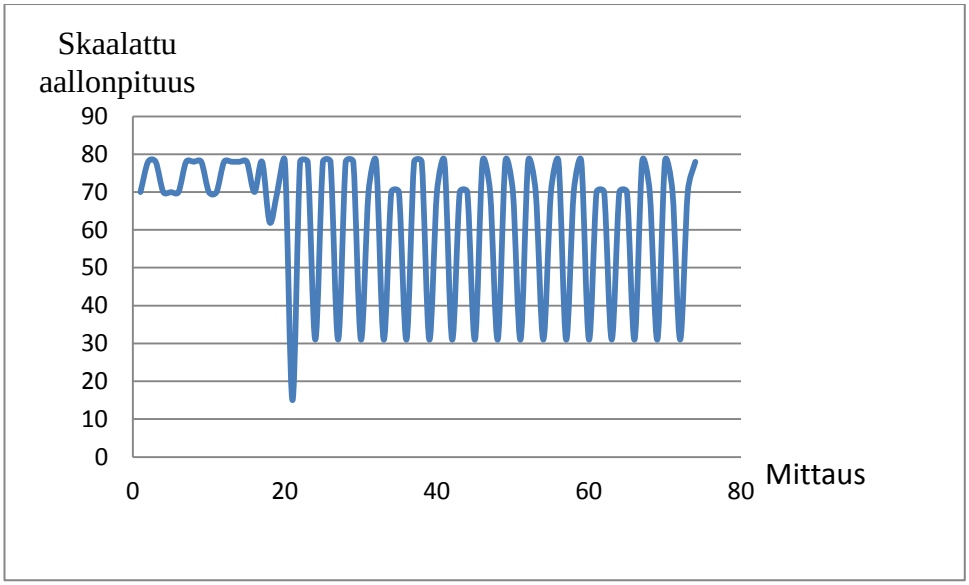
Opinnäytetyössä Arduino Unon pinniin 10 kytkettiin Nine Eagles:in valmistava kuusi kanavainen radiovastaanotin NE-RxX628-2,4GHz. Analogisekseen sisääntuloon voi kytkeä 2 kΩ suuruisen potentiometrin, jolloin mitataan sen yli olevaan jännitettä. Kuvan

17 koodilla voi testata uuden anturin lähettämää signaalin lukua ja tarkastella raa'an datan laatua.

9.5 Datan uudelleen skaalaaminen

Arduinon kehitysympäristössä on mahdollista käyttää valmista map()-funktiota vaihtamaan muuttujan lukuarvon vaihtelun. Esimerkiksi, jos vastaanotetun signaalin pulssipituus vaihtelee välillä 1209–1843µs, voidaan se muuttaa 0-180µs, -180-0µs tai -180–180µs (Arduino map(value, fromLow, fromHigh, toLow, toHigh). 2014).

Tutustuessa map()-function käyttöön huomattiin siinä olevan heikkous. Jos uusi skaalattu numeroväli on huomattavasti suurempi tai pienempi kuin alkuperäinen, syntyy numeroarvoon virhettä. Muuttujan skaalattu arvo ei välttämättä saavuta uutta minimi- tai maksimiarvoa ja arvon muuttaminen ei tapahdu tasaisesti. Kuviossa 1 on Nine Eagles vastaanottimen signaali kun se on skaalattu 1209–1843 µs:sta 0-5000µs:iin ja asetettu minimitasolle.



KUVIO 1. Voimakkaan skaalaamisen vaikutus digitaaliseen signaaliin

Kuviosta nähdään, ettei muuttujan arvo saavuta asetettua minimiarvoa. Tämän lisäksi pulssinpituudessa tapahtuu vaihtelua, vaikkei ohjauksessa ei tapahdu muutosta.

Map()-funktiota varten voi tehdä oman skaalaamisfunktion loop():in ulkopuolelle ja kutsua sitä tarvittaessa. Tämä tapa oli hyödyllinen kun monta muuttujaa täytyi skaalata samalla tavalla ja helpottaa koodin lukemista. Kuvassa 19 on esimerkki skaalaamisfunktion kutsumisesta loop():n ulkopuolelta.

```
digi = pulseIn(10,HIGH,25000); //Pinnin 10 arvo luetaan ja tallennetaan muuttujaan
digi = scaleDigital(digi); //Kutsutaan funktio, syötetään muuttuja ja talletaan
                           //skaalattu arvo muuttujaan

analog = analogRead(A0); //Pinni A0 arvo luetaan ja tallennetaan muuttujaan
analog = scaleAnalog(analog);  //Kutsutaan funktio, syötetään muuttuja ja talletaan
                               //skaalattu arvo muuttujaan
```

KUVA 19. Funktion kutsumisesimerkki

Kuvassa 19 käytetään samoja muuttujia kuin kuvassa 17. Tämän lisäksi täytyy tehdä loop():in aaltosulkeiden ulkopuolelle kaksi funktiota, joissa skaalaaminen toteutetaan analogisen ja digitaalisen. Kuvassa 20 on esimerkki näistä skaalaamisfunktioista.

```
/*Digitaalisen sisääntulon skaalaamisfunktio*/
int scaleDigital (int digi)
{
  digi=map(digi,1199,1834,0,180);
  return digi;

}

/*Analogisen sisääntulon skaalaamisfunktio*/
int scaleAnalog (int analog)
{
  analog=map(analog,0,1032,0,180);
  return analog;
}
```

KUVA 20. Skaalaamisfunktio esimerkki

Tämä oli tapa, jolla harjoittelin funktioiden tekemistä loop():in ulkopuolelle, joka on hyödyllinen taito tehdessä monimutkaisempia koodeja. Funktio ”return” lopettaa funktion toiminnan ja palauttaa sen jälkeen kirjoitetun muuttujan arvon siihen kohtaan koodia, jossa funktiota kutsuttiin (Arduino return. 2014).

Kun harjaantuu lukemaan koodia, voidaan skaalaaminen tehdä myös kuvan 21 esittämällä tavalla. Funktiota ei tehdä loop():in ulkopuolelle, vaan samalla koodirivillä suoritetaan sisääntulon lukeminen, tiedon tallentaminen muuttujaan ja skaalaaminen uudelleen.

```
digi=map(pulseIn(10,HIGH,25000),1199,1834,0,180);
//Pinnin 10 arvo luetaan ja tallennetaan muuttujaan,
//jonka jälkeen se skaalataan uudelleen
```

KUVA 21. Datanlukeminen, tallentaminen muuttujaan ja uudelleen skaalaaminen yhdellä koodirivillä

Kuvan 21 tavalla koodin kirjoittaminen nopeutuu ja koodi pysyy siistinä, jos monet muuttujat vaativat erilaisen skaalauksen. Kuvan 19 ja 20 tavassa tehdä skaalaaminen oli aluksi selkeämpää kuin monen funktion kirjoittaminen sisäkkäin.

9.6 Muuttujan arvon suodattaminen funktiolla

Testattaessa datanlukua huomattiin muuttujan arvon vaihtelevan joskus turhan nopeasti herkän sisääntulon ja nopean ohjelmakierron seurauksena. Digitaalisissa sisääntuloissa tämä oli ongelmana erityisesti herkkien anturien dataa käytettäessä.

Sisääntulon vaihtelua voi suodattaa tekemällä muutoksia koodiin. Yksi tapa on hidastaa koodin kierrosnopeutta delay() -funktiolla. Tämä funktio lisää tauon millisekunteina siihen kohtaan koodia, johon se kirjoitetaan. Nostamalla tauon liian pitkäksi kadotetaan sulava muutos anturi datassa. Lisäksi delay() vaikuttaa kaikkiin koodin funktioihin. Kun delay():n asettama tauko on päällä, kaikki mikro-ohjaimen toiminnot pysähtyvät! Toimilaitteet eivät liiku ja sisääntulojen lukeminen loppuu.

Arduinon tekemässä Smoothing -esimerkissä on tapa, jossa analogisen sisääntulon muuttujan tasoittaminen tehdään laskemalla keskiarvo muuttujan edellisten arvojen avulla. Esimerkki löytyy myös Arduinon kotisivuilta:

http://arduino.cc/en/Tutorial/Smoothing

Tarkoituksena on tehdä funktio, joka tasoittaa digitaalisen sisääntulon muuttujaa. Suodatusfunktiota varten täytyy määrittää 3 globaalia muuttujaa ja yksi vakio jokaista tasoitettavaa muuttujaa kohden. Kuvassa 22 on esimerkki kahden muuttujan suodatusta varten tehdyistä globaaleista muuttujista ja vakio. Suodattamisen kohteena on kuvan 17 muuttujat int digi ja int analog.

```
/*---------------------------------------*/
/*Koodissa käytettävät globaalit muuttujat*/

int digi = 10; //Muuttujan digi luetaan pinnistä 10
int analog; //Muuttujaan analog talletetaan analoginen arvo

#define samples 7

long total =0;
int reading[samples];
int index=0;

long total2 = 0;
int reading2[samples];
int index2=0;
/*---------------------------------------*/
/*Koodin alustaminen*/
void setup()
```

KUVA 22. Suodatusfunktion globaalit muuttujat ja vakio

Vakion ”samples” avulla määritellään kuinka monen ohjelmakierron päästä keskiarvo lasketaan. Koodi käyttää samaa vakiota kaikissa suodatusfunktiossa. Merkintä #define samples 7 on hyödyllinen työkalu, jos on tarpeellista käyttää vakio arvoja. Tämä komento muuttaa koodiin kirjoitetun samples -sanan numeroksi 7 kun koodi siirretään mikro-ohjaimeen kehitysympäristöstä (Arduino Define. 2014).

Huono puoli tässä vakion määritystavassa on se, ettei sanaa ”samples” vapaudu käyttöön edes osana muuttujan nimeä. Esimerkiksi, jos muuttuja määritellään int newsamples, muuttaisi ohjelma muuttujan määritelmän int new7, kun se siirrettäisiin mikro-ohjaimeen. Lisäksi kaikki kehitysympäristön funktiot eivät pysty käyttämään #define tavalla määriteltyjä vakioita. Tässä funktiossa tämä määrittely kuitenkin toimii (Arduino Define. 2014).

Malli suodatusfunktiosta kahdelle muuttujalle on esitetty kuvassa 23.

```
//Muuttujan arvojen tasoittamiseen käytettävät funktiot
  int digiSmoothing ( int digi)
 {
   total = total - reading[index]; //Vähennetään viime kierroksen arvo muuttujasta
   reading[index] = digi;          // Muuttujaan talletetaan funktion syötetty arvo
   total = total + reading[index]; // Summataan uusi arvo edellisten kierrosten arvoihin

   index++;                        //Lasketaan kuinka monta kierrosta on tehty

     if (index >= samples){        //Asetetaan ehto keskiarvon laskun uudelleen aloittamiselle
       index = 0;                  //Jos ehto ei täyty ohjelma ohittaa tämän kohdan
     }

    return total / samples;       //Palautetaan arvo takaisin missä funktiota kutsuttiin.

 }



  int analogSmoothing ( int analog)
 {
   total2 = total2 - reading2[index];
   reading2[index] = analog;

   total2 = total2 + reading2[index];

   index2++;

     if (index2 >= samples){
       index2 = 0;
     }

    return total2 / samples;
 }
```

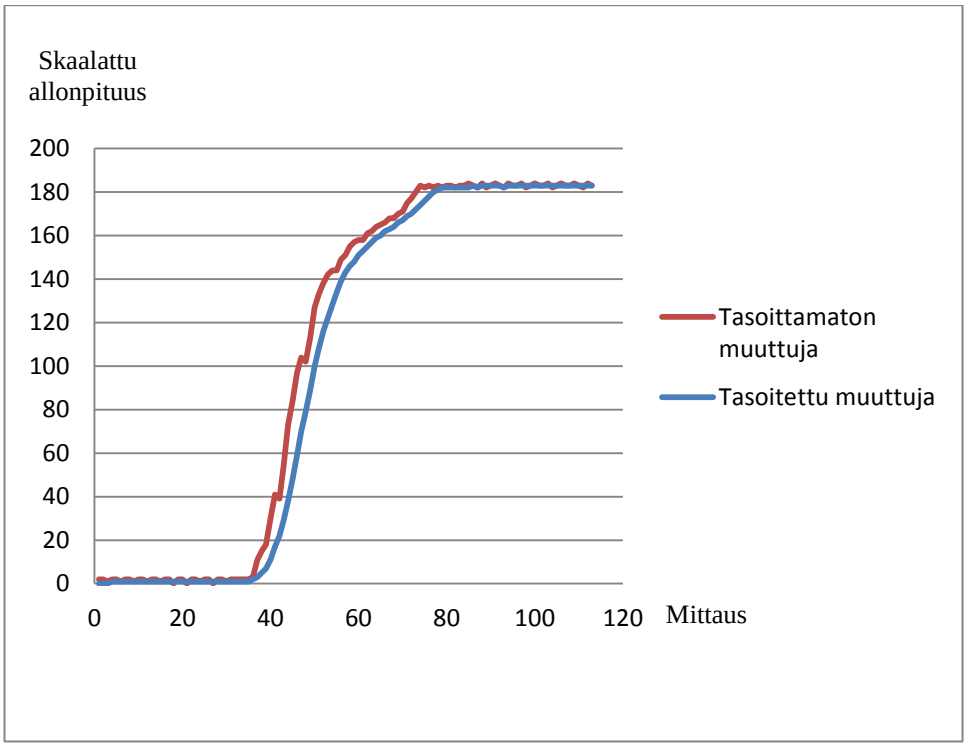
KUVA 23. Esimerkki kahdelle muuttujan suodattamisfunktiosta

Tämän jälkeen voidaan kutsua suodattamisfunktiota koodissa kun sille on tarvetta. Kuvassa 24 on esimerkki digiSmoothing -funktion kutsusta muuttujan määrittämisen jälkeen.

```
digi = pulseIn(10,HIGH,25000); //Pinnin 10 arvo luetaan ja tallennetaan muuttujaan
digi = digiSmoothing(digi);//Kutsutaan funktio, syötetään muuttuja ja talletaan
                           //skaalattu arvo muuttujaan
```

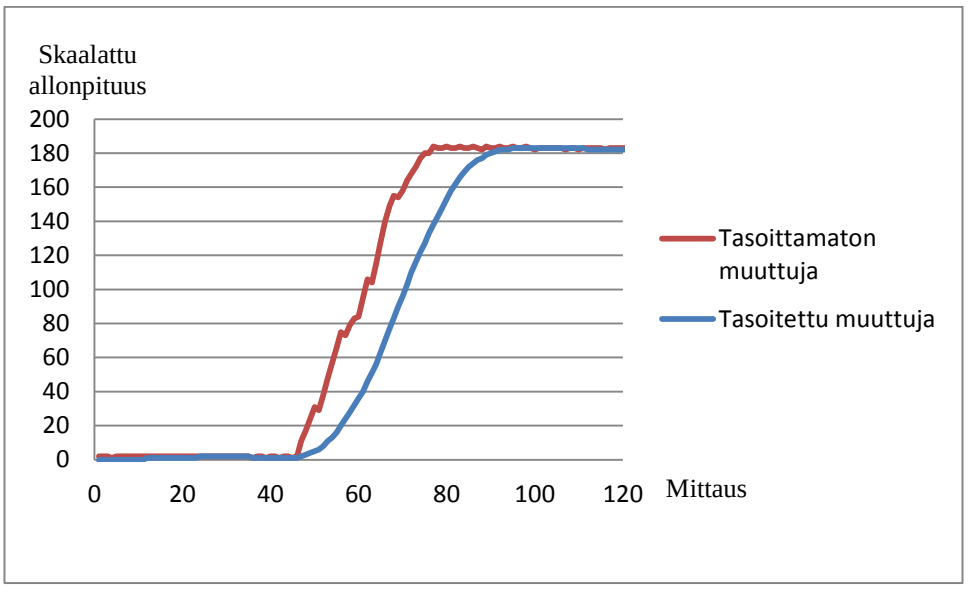
KUVA 24. Suodatusfunktion kutsuminen

Tehtiin testi, kuinka suodatusfunktio vaikuttaa muuttujan arvoihin ja verrattiin niitä suodattomattomiin arvoihin. Kuviossa 2 on yhden muuttujan arvo eri mittaushetkinä. Digitaalisen sisääntulon arvoa nostetaan tasaisesti ja tarkkaillaan, miten suodatusfunktio vaikuttaa arvon muuttumiseen. Ylempi käyrä on muuttujan arvo ennen suodatusfunktion käyttöä ja alempi funktion käytön jälkeen. Keskiarvo otettiin 7 edellisen ohjelmakierron arvoista.



KUVIO 2. Suodatusfunktion vaikutus muuttujan arvon vaihteluun

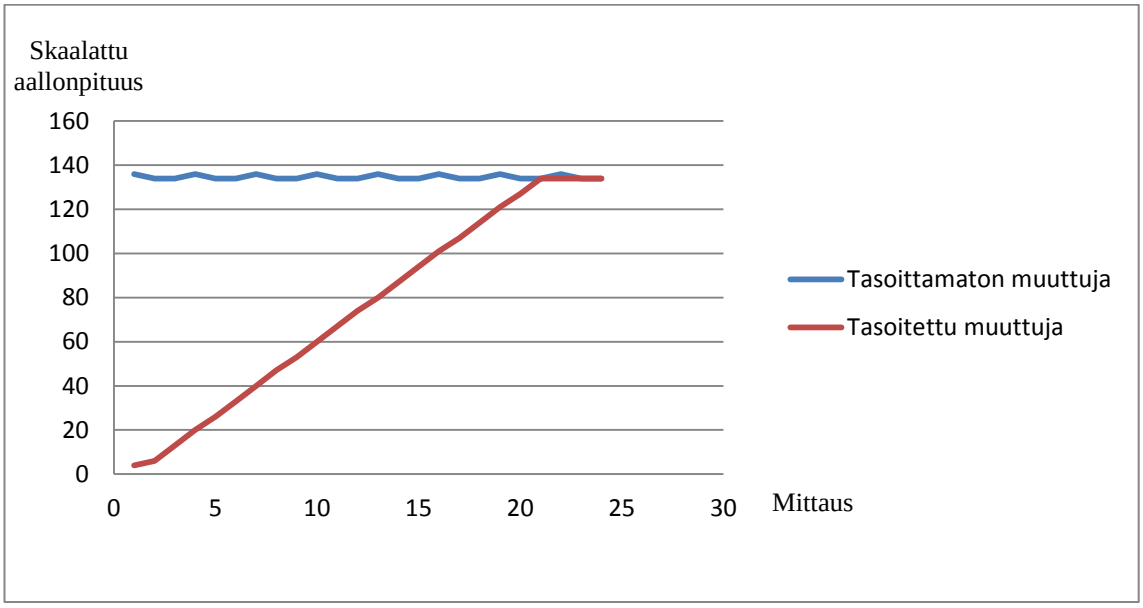
Suodatusfunktio vaikuttaa muuttujan arvoon selkeästi. Molempien käyrien muoto on samanlainen, mutta suodatettu muuttuja muuttuu tasaisemmin kuin suodattamaton. Muuttujan arvoa säädettiin käsisäädöllä, joten sisääntulon arvoa ei muuttunut täysin tasaisesti. Kuviossa 3 on sama tilanne kuin kuviossa 2, mutta keskiarvo otetaan 20:tä ohjelmakierrokselta.



KUVIO 3. Suodatusfunktion vaikutus muuttujan arvoon suurella näytteenotto määrällä

Kuviosta 3 nähdään, että muuttujan suodatettu arvo muuttuu entistä tasaisemmin kuin kuviossa 2. Tosin suodatettu signaalin tulee huomattavasti suodattamattoman signaalin jäljessä. Jos tämä muuttuja ohjaisi vakauttamisesta vastuussa olevaa moottoria, voi nopeat heilahdukset jäädä poistamatta.

Suodattamisfunktio tuo myös muita ongelmia. Muutaman ensimmäisen ohjelmakierroksen aikanatasoitettu muuttujan arvo on virheellinen. Esimerkiksi, jos keskiarvo lasketaan 20 edellisen ohjelmakierroksen ajalta, täytyy ohjelman käydä 20 ohjelmakierrosta, jotta funktio alkaa toimia halutusti. Kuviossa 4 on tasoittamaton ja suodatetun muuttujan arvo n. 25 ensimmäisen ohjelmakierroksen ajan. Nouseva käyrä on suodatettu muuttujan arvo ja toinen on suodattamattoman muuttujan arvo.



KUVIO 4. Suodatusfunktion hidas alku

Testatessa servojen liikuttamista, suodatusfunktion hidas alku aiheutti servojen nopean heilahtamisen koodin alussa. Kuviossa 4 esitetty ongelma voidaan korjata asettamalla setup() -funktioon koodin, jossa tehdään suodatettavan sisääntulolle muutama testikierros ennen loop()-funktioon siirtymistä. Testi kierroksia tulee tehdä vähintään yhtä monta kuin suodatusfunktion arvo on.

Funktio tuo siis hyviä ja huonoja ominaisuuksia. Tätä funktiota kannattaa käyttää jos sen hyvät ominaisuudet päihittävät ongelmat.

9.7 Sähkömoottorin käyttö Arduinolla

Koska päätettiin Arduinon ohjaavan kamerajalustaa, alettiin seuraavaksi tutkia mahdollisuuksia mitä sähkömoottoreita voidaan käyttää. Arduinon mikro-ohjaimella

pystyy ohjaamaan erilaisia DC-moottoreita (*Direct Current motors*). Perehtyessä Arduinoa käyttävien harrastajien projekteihin kolme sähkömoottori typpiä olivat yleisessä käytössä:

- tavallinen tasasähkömoottori
- askelmoottori
- servomoottori

Ensinnäkin pitää tietää, mihin osaan laitetta moottori kytketään. Halutaanko moottorin liikuttavan laitetta vaijerilla vai vakauttavan kameraa? Koska opinnäyteyössä suunnittelu keskittyy jalustan suunnitteluun, ei oteta kantaa vaunun moottoreiden valintaan. Niiden mitoittaminen ja sopivan moottorityypin valitseminen siirretään myöhempiin suunnittelu vaiheisiin.

Jalustan moottorit ovat vastuussa kameran vakauttamisesta ja suuntaamisesta kohteeseen. Niiltä vaaditaan siis tarkkuutta ja vakaata liikettä. Lisäksi alhaisen budjetin takia niillä ei saa olla liian paljon hintaa. Vaatimukset ovat samat kuin mikro-ohjainta valittaessa. Uusi vaikuttava tekijä on moottorityypin kytkemisen ja käytön helppous Arduinolla.

Tällä perusteella päädyttiin käyttämään vakauttamiseen RC-servomoottoreita (*Radio Control*). Servomoottorit olivat helposti saatavilla. Hyvä ominaisuus RC-servoissa on se, että ne voi kytkeä suoraan Arduinon pinneihin. Tavallinen tasasähkömoottori ja askelmoottori tarvitsevat mm. erillisen ohjainpiirin, transistorin ja diodin, jotta ne voi kytkeä mikro-ohjaimeen turvallisesti ja niiden ohjaaminen olisi mahdollista.

9.7.1 Servon kytkeminen Arduinoon

Jotta Arduinolla voi ohjata servoa, täytyy kehitysympäristöön ladata Arduinon kotisivuilta Servo-kirjaston (*Servo Library*). Kun kirjasto on asennettu, voidaan käyttää funktioita, jotka on suunniteltu servoja varten. RC-servoihin on kytkettynä erilaisia liittimiä, joissa on kolme eriväristä johtoa. Tärkeintä on tunnistaa, millä johdolla tuodaan ohjaussignaali servolle. Tämä johto kytketään Arduinon digitaaliseen pinniin. Kaksi muuta johtoa on jännitteen tuomista varten. Jännitteen servoa varten voi ottaa mikro-ohjaimesta, tai ulkoisesta DC-jännitelähteestä. Tärkeää on kuitenkin se, että mikro-ohjaimen ja servon maadoitettu on samalla tasolla. Servon ohjaaminen ei muuten toimi.

Johtojen värikoodi ei ole sama kaikilla servojen valmistajilla (Aripekan RC sivut. 2014). Esimerkiksi markkinoilta löytyy johtoja joiden väritys on:

- Keltainen, punainen ja musta
- Valkoinen, punainen ja musta
- Oranssi, punainen ja musta

9.7.2 Eri servotyypin käyttäytyminen Arduinossa

Arduinon Servo-kirjasto toimii eri tavalla erilaisilla servoilla. RC-servot voidaan jakaa kahteen tyyppiin niiden pyörimistavan mukaan:

- Jatkuvasti pyöriviin servoihin
- Tietyn kulman pyöriviin servoihin

Kun käytetään Arduinon Servo-kirjaston funktioita ”kulmaservoihin”, voidaan muuttujilla asettaa servo haluttuun asentoon vakio nopeudella. Jatkuvasti pyörivät servoihin kirjasto vaikuttaa eri tavalla. Nyt funktioita voidaan käyttää servon pyörimisnopeuden ja pyörimissuuntaa säätelyyn, mutta tarkkaa asentotietoa ei voi syöttää servolle.

Tasapainottamisen kannalta on tärkeää, että voidaan asettaa servon kuorma tarkasti tiettyyn kulmaan ja pitämään asemansa. Kulmaservoa käytetään siis jalustan vakauttamisessa.

9.7.3 Servon ohjaus Arduinolla

Pohjana servon ohjaamiseen käytettiin Arduinon tekemää Knob-esimerkkiä, joka löytyy kehitysympäristön esimerkkilistasta. Ensimmäisenä pitää ottaa käyttöön servojen kirjasto (Arduino Servo library. 2014). Kirjasto otetaan käyttöön kirjoittamalla koodin alkuun ennen setup()-funktiota:

```
#include <Servo.h>
```

Tämän jälkeen määritellään käytettäville servoille muuttuja. Annetaan esimerkiksi servolle nimeksi servo1:

```
Servo servo1;
```

Setup() – funktiossa pitää määrittää mihin pinniin servo on kytketty. Servo voi olla esimerkiksi kytkettynä pinniin 8:

```
servo1.attach(8);
```

Tämän jälkeen on mahdollista käyttää, joko analogista tai digitaalista sisääntuloa ohjaamaan servon asentoa. Käytetään esimerkkinä muuttujaa digi edellisistä esimerkeistä.

```
servo1.write(digi);
```

Jos käytössä on tiettyyn kulmaan asettuva servo, muuttujan digi arvo kertoo mihin kulmaan servo asettuu. Kulma voi olla väliltä 0-180°. Jatkuvasti pyörivän servon tapauksessa 0 on maksiminopeus toiseen pyörimissuuntaan ja 180 vastakkaiseen suuntaan. Kun muuttujan arvo on 90, pysyy servo paikoillaan (Arduino write(). 2014).

Kun koodissa yhdistetään edellä opitut muuttujat, voidaan tehdä kuvissa 25–27 esitetty koodi. Koodin avulla voidaan ohjata yhtä servoa digitaalisella signaalilla. Koodissa on edelleen mukana analogisen sisääntulon lukeminen, jonka voi vaihtaa ohjaavaksi signaaliksi. Sisääntulojen skaalaaminen ja suodattaminen on myös mukana. Tätä koodia voi käyttää pohjana usean servon ohjaavaan koodiin. On kuitenkin muistettava, jos haluaa käyttää suodatusta usean servon ohjauksessa, täytyy jokaiselle servolle tehdä oma suodatusfunktio ja omat muuttujat.

Suodatusfunktion takia servo tekee nopean nykäyksen, kun mikro-ohjain pistetään päälle tai painetaan reset -nappia mikro-ohjaimessa. Tätä heilahdusta voidaan vaimentaa lisäämällä setup()-funktioon koodi, joka lukee servoa ohjaavan digitaalisen sisääntulon arvon muutaman kerran ennen loop() -funktioon siirtymistä. Tämä on mahdollista while–loopilla, jossa määritellään ehto kuinka kauan ohjelma pysyy kyseisessä loopissa. Kun asetettu ehto muuttuu epätodeksi, ohjelma siirtyy pois while-loopista (Arduino whileloops. 2014).

Kuvassa 25 on esimerkki tarvittavista globaaleista muuttujista ja kirjastoista kun halutaan ohjata yhtä servoa suodatetulla signaalilla. Uutena muuttujana on int start. Tämän muuttujan arvoa käytetään while -loopin kierrosten laskentaan.

```
/*Koodissa käytettävät kirjastot*/
#include<Servo.h> //Otetaan käyttöön Servo-kirjasto

/*---------------------------------------*/
/*Koodissa käytettävät globaalit muuttujat*/

int digi = 10; //Muuttujan digi luetaan pinnistä 10
int analog; //Muuttujaan analog talletetaan analoginen arvo

Servo servo1; //Tehdään ohjattavalle servolle muuttuja

#define samples 7

int start = 0; // Tällä muututjalla lasketetaan
               // while -loopin kierroksia
long total =0;
int reading[samples];
int index=0;

long total2 = 0;
int reading2[samples];
int index2=0;
/*---------------------------------------*/
```

KUVA 25. Servon ohjaamiseen suodatetulla signaalilla tarvittavat globaalit muuttujat ja kirjastot

Setup()-funktio servon ohjauksessa on esitetty kuvassa 26. Muuttujien skaalaaminen on tehty tässä koodissa kuvan 21 esittämällä tavalla.

```
/*Koodin alustaminen*/
void setup()
{
  Serial.begin(9600); //Määritellään millä nopeudella dataa siirretään tietokoneelle
  pinMode (10, INPUT); //Määritellään, että pinni 10 on sisääntulo

  servo1.attach(8); //Määritellään servon ohjauspinnin sijainti

  while(start <= samples) //Määritellään ehto kuinka monta loop-kiertoa
  {                       // tässä osassa koodia ollaan
    start++;      //Muuttujan arvo nousee yhdellä joka loop-kierrolla
    digi = map(pulseIn(10,HIGH,25000),1199,1834,180,0);
    digi = digiSmoothing(digi);
    delay(10);
  }
}
```

KUVA 26. Esimerkki setup()-funktiosta servoa ohjatessa

Tällä ehdolla kuvan 26 while-loopissa olla yhtä monta kierrosta kuin suodatusfunktio ottaa näytteitä.

Kuvassa 25 on loop()-funktion sisältö, kun halutaan ohjata servoa suodatetuilla ja skaalatulla sisääntuloilla. Loop()-funktion ulkopuolelle jää vielä suodattamisfunktiot. Ne ovat edelleen samanlaiset kuin kuvassa 23.

```
void loop()
{
  /*Pinnien arvojen lukeminen ja uudelleen skaalaaminen*/

  digi = map(pulseIn(10,HIGH,25000),1199,1834,180,0); //Pinnin 10 arvo luetaan,
                                                      //tallennetaan muuttujaan ja skalaataan
                                                      // ilman erillistä functiota
  analog = map(analogRead(A0),0,1032,0,180); //Pinni A0 arvo luetaan
                                             //ja tallennetaan muuttujaan ja skaalataan
                                            // ilman erillistä functiota

  /*Muuttujien arvojen suodattaminen suodatuksen jälkeen*/

  digi = digiSmoothing(digi);
  analog = analogSmoothing(analog);

  servo1.write(digi); // Siirretään muuttujan arvo servolle
//servo1.write(analog); //Haluttaessa servoa voisi ohjata analogisella signaalilla
  Serial.println("");
  delay(10);
}
```

KUVA 27. Loop()-funktio servon ohjauksessa suodatetulla signaalilla

Jos servo liikesuuntaa haluaa muuttaa toisenlaiseksi, esimerkiksi tilanteessa jossa servon siipi liikkuu ylöspäin kun ohjainta työntää alaspäin, voi signaalin skaalaamisen muuttaa seuraavan laiseksi:

digi = map(pulseIn(10,HIGH,2500),1199,1834,0,180);

10 KAUKO-OHJAAMINEN

Kauko-ohjaaminen on olennainen osa laitteen toimintaa. Tässä vaiheessa suunnittelua päätettiin käyttää vain yhtä radio-ohjainta. Valmiin laitteen ohjaaminen toteutettaisiin kahdella ohjaimella, jolloin yksi käyttäjä on vastuussa kameran suuntaamisesta ja toinen jalustan liikkeestä vaijerilla. Syy tähän on se, että yksittäisen käyttäjän ei tarvitse keskittyä liian moneen asiaan kerralla.

Ensimmäisenä on päätettävä, montako kanavaa radio-ohjaimessa täytyy olla. Arvioidessa pohdittiin myös tulevia tarpeita ohjaukselle. Jalustassa ohjattavat toiminnot ovat kameran panorointi ja tilttaus. Molemmille liikkeille täytyy varata oma kanava. Vaijeri liikettä varten pitää vara yksi kanava.

Kauko-ohjain täytyy pystyä kytkemään Arduinon mikro-ohjaimeen. Monet Arduinon käyttäjät valitsevat projekteissaan kauko-ohjaamiseen tavallisia RC-ohjaimia. Arduinon

lisälaitteiden avulla olisi mahdollista rakentaa oma kauko-ohjainjärjestelmä. Opinnäytetyötä varten käytössä oli kaksi Arduino Uno mikro-ohjainta, joista toisesta olisi voinut valmistaa radiolähetin lisäkomponenttien avulla. Nämä komponentit olivat edullisia. Tämä vaihtoehto olisi ollut halvempi toteuttaa kuin RC-vastaanottimen ja lähettimen käyttö.

Rakennetun kauko-ohjauksen suurin ongelma olisi langattoman yhteyden muodostaminen kahden mikro-ohjaimen välille. Tätä varten olisi pitänyt tehdä oma koodi langattoman tiedonsiirrolle muun ohjaamisen lisäksi, kun taas RC-lähettimessä ja -vastaanottimessa tämä langatonyhteys olisi jo valmiina.

RC-Vastaanotin kytketään mikro-ohjaimen digitaaliseen sisääntuloon ja luetaan sen vastaanottamia signaaleja. Näiden signaalien lukemiseen oli jo koodit valmiina. Tämän takia päädyttiin käyttämään RC- ohjainta ja vastaanotinta.

Opinnäytetyössä käytettiin Nine Eagles NE-T008 lähetintä ja sen mukana tullutta NE-RxX628-2,4GHz vastaanotinta. Lähetin ja vastaanotin ovat kuvassa 28.

KUVA 28. Opinnäytetyössä käytetyt radiovastaanotin ja –lähetin (Kuva: Yli-Viikari 2014)

Käyttämällä kuvissa 25–27 esitettyä koodia ja kuvan 28 RC-tarvikkeita, saatiin servo liikkumaan lähettimen ohjainsauvalla. Langaton ohjaus saatiin toimimaan arviota

helpommin. Koodi testattiin enimmäkseen yhdellä servolla ja sillä tehtiin yksinkertaisia liiketestejä.

Radiovastaanottimen asetuksia muuttamalla on mahdollista tehdä lisäsäätöä ohjaukseen. Lähettimen asetukset laitettiin tehdasasetuksille ja vastaanottimen takana oleva vipu asetettiin MODE 1 asentoon eikä näitä asetuksia muutettu sen jälkeen. MODE 1 asento estää oikeaa ohjainsauvaa palautumasta keskiasentoon kun siitä irrottaa. Vasen ohjainsauva toimii päinvastoin. Tässä suunnitteluvaiheessa testattiin kuinkin langattoman yhteyden muodostamista mikro-ohjaimen ja radiolähettimen välille.

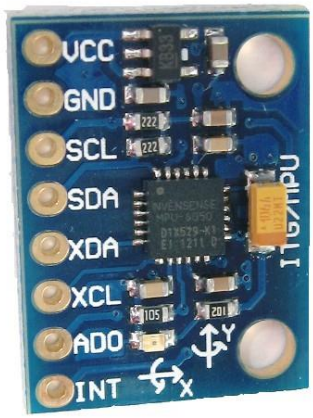
Koska jo käytössä oleva digitaalisen sisääntulon lukemiseen tehty koodipohja (kuva 17) toimi langattoman ohjauksen toteuttamisessa, alettiin seuraavaksi suunnitella koodia automaattista vakauttamista varten.

10.1 Langattomanohjauksen ongelmat

Testien aikana ilmeni outo ominaisuus Arduinon mikro-ohjaimen ja radiovastaanottimen välillä. Joskus uuden koodin lataamisen yhteydessä servoa ohjaava radiokanava vaihtui toiseen ohjaussauvan kanavaan. Esimerkiksi ennen koodin lataamista oikea ohjaussauva oli vastuussa vakauttavan servon ohjaamisesta. Koodin uudelleen lataamisen jälkeen vasen sauva ohjaa vakauttavaa servoa. Vastaanotin otettiin pois päältä uuden koodin lataamisen ajaksi. Ongelmalle ei löytynyt ratkaisua opinnäytetyön aikana.

11 AKTIIVISEN VAKAUTTAMISEN TOTEUTTAMINEN ARDUINOLLA

Aktiivista vakauttamista varten piti löytää anturi, jolla Arduinon mikro-ohjain havaitsee heilunnan. Anturiksi valittiin 3 –akselinen kiihtyvyysgyroskooppianturi. Käytetyn anturin tyyppi oli GY-521 mpu6050. Anturin hinta oli n. 5 €. Kuvassa 28 on esitetty gyroskooppikiihtyvyysanturi GY-521 mpu6050.



KUVA 28. Gyroskooppianturi GY-521 mpu6050 (MPU-6050 Accelerometer + Gyro. 2014)

Halvemmalla anturilla on hyvä testata koodin toimivuutta ja tarkastellaan, onko anturin tarpeeksi tarkka jatkokäyttöä varten. Kiihtyvyysgyroskooppi anturilla on mahdollista mitata x-, y- ja z-akselilla kiihtyvyyden tai kulman suuruuden. Tästä eteenpäin tähän anturiin viitataan nimellä ”gyro”.

11.1 Gyron kytkeminen Arduinoon

Gyron kytkeminen oli monimutkaisempaa kuin aluksi oletettiin. Aluksi yritettiin saada dataa gyrosta samaan tapaan kuin RC-vastaanottimesta, mutta tämä ei onnistunut. Gyro ja Arduinon välinen tiedonsiirto toimii I^2C -väylällä ja vaatii Wire nimisen kirjaston funktioita, jotta anturin dataa voitaisiin lukea. Gyron kytkemiseen on ohjeet Arduinon kotisivuilla:

http://playground.arduino.cc/Main/I2CBi-directionalLevelShifter

Gyron kytkennän testaamiseen käytettiin i2c_scanner nimistä koodia, jonka voi kopioida Arduinon kotisivuilta. Koodilla testataan löytääkö Arduino I^2C -väylää käyttävää laitetta. Koodi ilmoittaa serial monitorissa, jos laite löytyy. Alla on nettiosoite, josta voi kopioida koodin:

http://playground.arduino.cc/Main/I2cScanner

Gyron datan lukemiseen täytyi siis löytää uusi koodi. Gyrosta haluttiin tieto missä asennossa anturi on ja käyttää tätä tietoa servon ohjaamiseen. Yleisesti aloittelijoita neuvottiin tutustumaan Arduinon kotisivuilla löytyvään MPU-6050 ohjeisiin (MPU-6050 Accelerometer + Gyro. 2014)

Arduinon MPU-560 ohjeilla onnistuttiin lukemaan gyron raakaa dataa. Raakaa dataa ei voitu kuitenkaan käyttää servon ohjaamiseen. Gyron antamat arvot vaihtelivat liian herkästi, vaikka gyro lepäsi vakaalla alustalla. Vaihtelua ei saatu pienenettyä aiemmin käytetyillä suodatusfunktiollakaan. Näytteiden ottoa olisi pitänyt nostaa todella korkeaksi, joka vuorostaan olisi hidastanut anturia liikaa. Gyroa varten täytyi tehdä oma suodatusohjelma, jotta gyron data olisi käyttökelvollista.

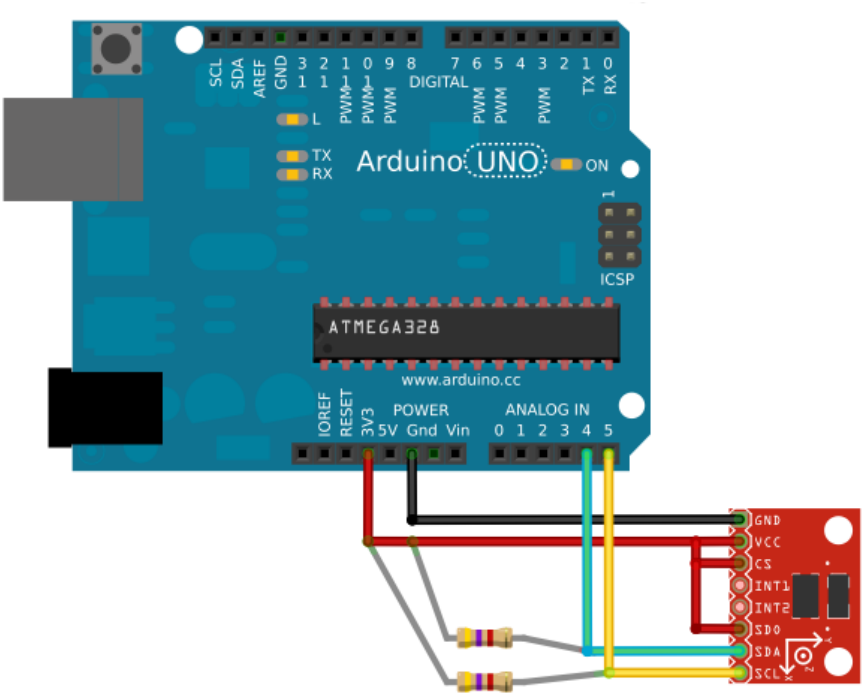
Alettiin etsiä muiden Arduinon käytäjien tekemiä koodeja, jotka olisivat vapaasti käytettävissä ja mahdollistaisi gyron datan lukemisen. Geek Mom Projects -kotisivuilta oli Arduino projekteja ja ohjeita MPU-560 käyttöön ja suodatin gyron datan lukemiseen. Suodatin toimii yhdistämällä gyron sisältämän gyroskoopin ja kiihtyvyysanturin dataa määrittämään anturin asennon. Alla on linkki kyseiseen nettisivuun:

http://www.geekmomprojects.com/gyroscopes-and-accelerometers-on-a-chip/

Koodi, jota käytettiin opinnäytetyössä pohjana gyron datan lukemiseen, on nimeltään ”gy_521_send_serial_filtered_data”. Koodin avulla saatiin gyrosta tarkkoja arvoja ja ne olivat vakaita. Tästä eteenpäin tähän koodiin viitataan ”gyrofiltterikoodina”.

Gyrofiltterikoodi on todella pitkä ja sisältää lähes 1000 koodiriviä, joten sen esittäminen kirjallisessa muodossa on hankalaa. Koodissa on kuitenkin enimmäkseen vakioiden määrittämistä. Haasteena oli lisätä tähän asti suunnitellut osat mukaan koodiin ja varmistaa, että ohjelma toimii kunnolla. Tärkeintä on tunnistaa koodista osat, joita muutamalla saadaan koodi toimimaan halutulla tavalla.

Alkuperäisessä muodossaan koodi tulostaa serial monitoriin gyron mittaaman x, y ja z-tason raa’an kallistuskulman ja kiihtyvyyden ja jokaisen edellä mainitun akselin suodatetun kallistuskulman. X ja y -akselin suodatettu kallistuskulma on gyron todellinen kallistumiskulma. Z-akselin kulma on kiertokulma mikro-ohjaimen käynnistyshetkestä lähtien. Gyrofiltterikoodin täytyy antaa käydä muutaman sekunnin ajan ennen kuin se antaa oikean lukeman. Kuvassa 29 on kytkentäkuva I^2C-väylää käyttävän laitteen kytkemiselle Arduino Unoon. Tätä kytkentää käytettiin gyrofiltterikoodin testaamisessa.



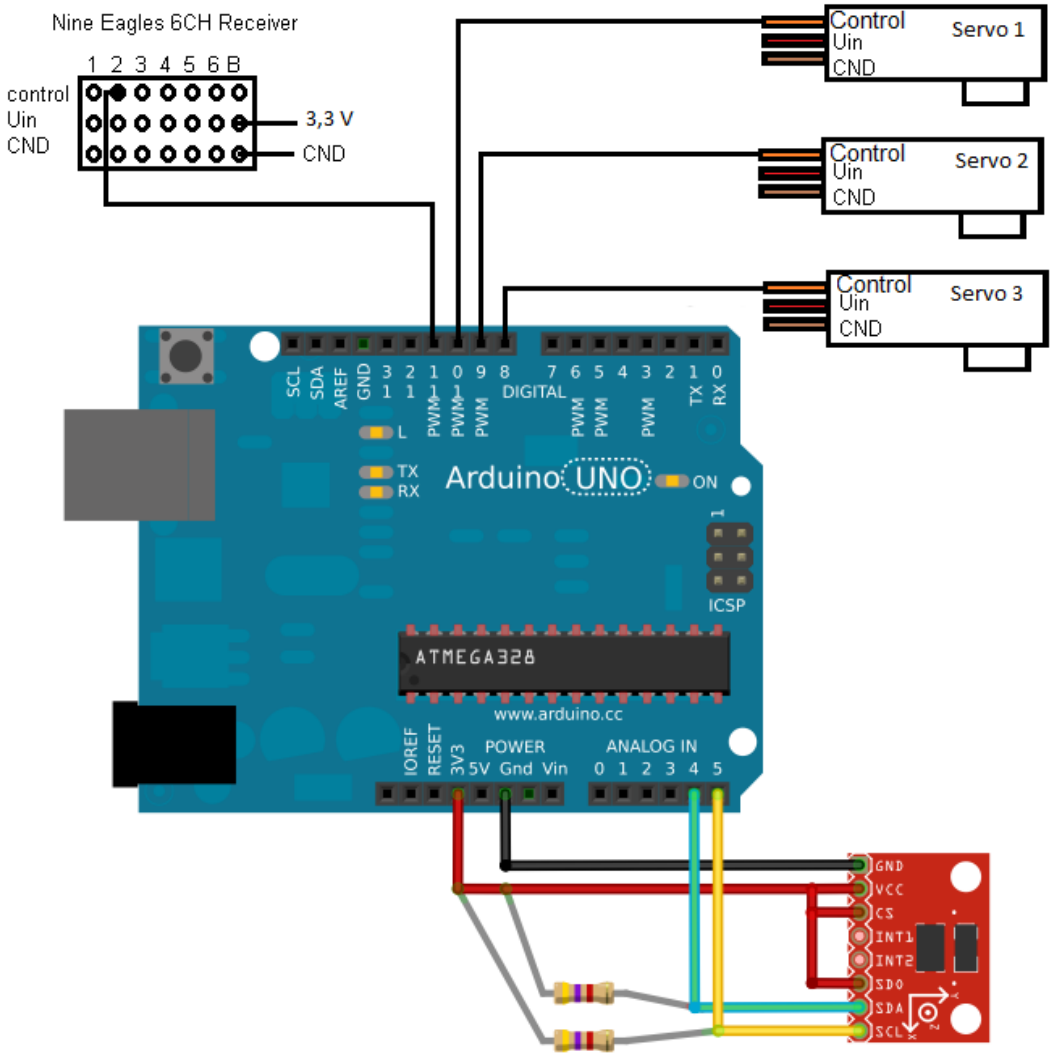
KUVA 29. I^{2}C-väylää käyttävän laitteen kytkeminen Arduino (I^{2}C bi-directional level shifter. 2014)

Gyrossa ei ole kuvan 29 CS ja SDO pinniä, mutta tämä ei haitannut. Kuvan 29 vastuksina käytettiin 4,7 kΩ vastuksia. Gyro toimi kaikissa testikoodeissa hyvin tällä kytkennällä.

Geek Mom Projects:in tekemään alkuperäiseen koodiin tehtiin muokkauksia, jotta se saataisiin toimimaan opinnäytetyöhön sopivemmaksi. Tässä muodossa servot seuraavat gyron asentoa. Tässä muodossa muokattu gyrofiltterikoodi mahdollistaa Arduino Unoon radiovastaanottimen, kolme servon ja gyron kytkemisen. Koodi on esitetty liitteessä 1. Koodiin tehtyjä muokkauksia on liitteen 1 riveillä:

- 815-825, jossa määrittellään servoille ja digitaaliselle sisääntulolle muuttuja
- 876-881, jossa määrittellään mihin pinniin servot ja digitaalinen sisääntulo on kiinni ja tarvittaessa pinnien tila
- 887-889, jossa luetaan ja skaalataan uudelleen digitaalinen sisääntulo
- 906-919, 926-943, 947-951 ja 994-1010 karsittiin serial monorista pois ylimäärinen tieto muuttamalla koodirivit kommenteiksi
- 1018-1027, jossa lisättiin serial monitoriin digitaalisen sisääntulon tuloste ja servojen ohjaus funktiot

Liitteen 1 koodilla voi tarkastella, miten kukin sisääntulo käyttäytyy kun gyroa liikuttaa. Koodissa on myös tehty muuttujat kolmen servon ohjaamiselle ja digitaalisen sisääntulon lukemiselle. Servot liikkuvat gyron kallistumisen mukana eivätkä vielä tee vakauttavia liikkeitä. Servo 1 liikkuu x-akselin kallistuskulman mukaan, servo 2 y-akselin ja servo 3 z-akselin mukaan. Digitaaliseen sisääntuloon on mahdollista kytkeä RC- vastaanotin ja sisääntulossa on skaalaaminen mukana. Serial monitor tulostaa, missä kulmassa gyro on kullakin akselilla on ja mikä arvo digitaalinen sisään tulo antaa. Kuvassa 30 on kytkentäkuva liitettä 1 varten.



KUVA 30. Gyron, kolmen servon ja RC-vastaanottimen kytkentäkuva (I^2C bi-directional level shifter. 2014. muokattu)

Kuvan 30 kytkennässä on tärkeää, että kaikki käytetyt komponentit ovat maadoitettu samalle tasolle, jotta kytkentä toimii oikein. Servoille ja vastaanottimelle käytettiin ulkoista jännitelähdettä sillä mikro-ohjaimen pinneistä otettu virta ei riittänyt.

12 KAMERAJALUSTAN SUUNNITELUN LOPPUTULOS

Liitteen 1 koodi jäi keskeneräiseksi, sillä ohjattavat servot eivät vielä toteuta vakauttamistoimintaa. Koodi toimii kuitenkin hyvänä pohjana jatkokehitykselle ja sen avulla pääsee tutustumaan gyron toimintaan. Opinnäytetyössä esitetyillä koodeilla on mahdollista toteuttaa koodaamisen harjoittelulle asetetut tavoitteet.

Opinnäytetyössä löydettiin ohjauksen perustoiminnoille ratkaisut ja saatiin näkemystä miten Arduino soveltuu kamerajalusten ohjaamiseen. Arduino soveltuu hyvin prototyyppi ohjauksen suunnitteluun, sillä se on monipuolinen ja helposti saatavilla. Arduinon ohjelmointiin oli myös suhteellisen helppoa oppia vaikkei ohjelmointi kokemusta olisikaan paljon. Omaan projektiin sopivaa tietämystä voi olla kuitenkin vaikea löytää ja tämä tekee Arduinon kanssa työskentelystä aluksi vaikeaa. Sen jälkeen, kun tietolähteet ja sopivat esimerkit löytyivät, helpottui työskentely Arduinolla huomattavasti. Arduinon käyttäjien yhteisön jakamat ohjeet ja projektit olivat myös avuksi.

Opinnätetyön koodi ei täysin tuottanut haluttua tulosta, mutta se toimii hyvänä pohjana kamerajalustan ohjaamisen toteuttamiselle. Servojen sopivuutta projektiin ja gyron ohjauskeen käytettävää koodia täytyy vielä testata enemmän.

Opinnäytetyön suunnittelu ei edennyt alussa määriteltyyn tavoitteeseen, jossa testattava olisi asetettuja toimintoja toteuttava prototyyppi. Yhdelle henkilölle suunnittelun loppuun vieminen olisi vaatinut enemmän tietämystä mekaanisesta suunnitelusta ja tuotteen valmistamisprosessista. Opinnäytyö ratkaisi vain pienen osan ongelmista, joihin kamerajalustan suunnittelussa törmää.

Olen kuitenkin itse tyytyväinen saavutukseen sillä sain paljon uutta osaamista ja pääsin pohtimaan itse kiinnostavia ongelmia ja sain uusia taitoja. Samalla sain näkemystä, mitkä ovat omat vahvuudet ja heikkoudet tuotesuunnilussa. Opinnäytetyön tekeminen tästä aiheesta oli palkitsevaa.

LÄHTEET

Amazon. 2014. Broadcast & Pro A/V. Www -sivu. Luettu 13.3.2014.
http://www.amazon.com/gp/offer-listing/B001GXR03A/?tag=dealscheckers-20

Arduino Analog Input Pins. 2014. Www -sivu. Luettu 12.5.2014.
http://arduino.cc/en/Tutorial/AnalogInputPins

Arduino analogRead(). 2014. Www -sivu. Luettu 26.5.2014.
http://arduino.cc/en/Reference/analogRead

Arduino Blink. 2014. Www -sivu. Luettu 6.5.2014. http://arduino.cc/en/tutorial/blink

Arduino Const. 2014. Www -sivu. Luettu 5.5.2014.
http://arduino.cc/en/Reference/Const

Arduino Define. 2014. Www -sivu. Luettu 3.6.2014.
http://arduino.cc/en/Reference/Define

Arduino Digital Pins. 2014. Www -sivu. Luettu 12.5.2014.
http://arduino.cc/en/Tutorial/DigitalPins

Arduino Functions. 2014. Www -sivu. Luettu 5.5.2014.
http://arduino.cc/en/Reference/FunctionDeclaration

Arduino int. 2014. Www -sivu. Luettu 5.5.2014. http://arduino.cc/en/Reference/int

Arduino Libraries. 2014. Www -sivu. Luettu 5.5.2014.
http://arduino.cc/en/Reference/Libraries

Arduino long. 2014. Www -sivu. Luettu 5.5.2014. http://arduino.cc/en/Reference/Long

Arduino map(value, fromLow, fromHigh, toLow, toHigh). 2014. Www -sivu. Luettu 27.5.2014. http://arduino.cc/en/reference/map

Arduino pulseIn(). 2014. Www -sivu. Luettu 12.5.2014.
http://arduino.cc/en/Reference/pulseIn

Arduino return. 2014. Www -sivu. Luettu 3.6.2014.
http://arduino.cc/en/Reference/Return

Arduino Robotics Projects. 2012. Www -sivu. Luettu 23.4.2014.
http://www.robotshop.com/blog/en/arduino-robotics-projects-3666

Arduino Servo Library. 2014. Www -sivu. Luettu 5.5.2014.
http://arduino.cc/en/Reference/Servo

Arduino Variables. 2014. Www -sivu. Luettu 5.5.2014.
http://arduino.cc/en/Tutorial/Variables

Arduino whileloops. 2014. Www -sivu. Luettu 18.8.2014.
http://arduino.cc/en/Reference/While

Arduino write(). 2014. Www -sivu. Luettu 12.8.2014.
http://arduino.cc/en/Reference/ServoWrite

Aripekan RC sivut. 2014. Www -sivu. Luettu 9.6.2014.
http://personal.inet.fi/koti/liljeroos/servo/servo.html

B&H.2014. Camrecorders. Www-sivu. Luettu13.3.2014.
http://www.bhphotovideo.com/c/product/619431-
REG/Sony_HDW790_HDW_790_HDCAM_High_Definition.html

Conversion of Measurement Units. 2014. Convert kilogram centimeter to newton-meter. Www -sivu. Luettu 11.6.2014.
http://www.convertunits.com/from/kilogram+centimeter/to/newton-meter

CPV. 2014. Camrecorders. Www-sivu. Luettu 13.3.2014.
http://cvp.com/index.php?t=product/sony_hdw-790p

Ebay. 2014. Cameras & Photo. Www -sivu. Luettu 13.3.2014

Forum Arduino CC. 2014. Www -sivu. Luettu 5.5.2014.
http://forum.arduino.cc/index.php/topic,48250.0.html

HMS Illustrious-Aircraft Carrier _ Richards Hammond's EngineeringConnections. 2013. Dokumenttitelevisosarja . Katsottu 9.2013.
http://www.youtube.com/watch?v=GzKzqyQaOyow

I2C bi-directional level shifter. 2014. Www -sivu. Luettu 8.11.2014.
http://playground.arduino.cc/Main/I2CBi-directionalLevelShifter

Kaapelikauppa. 2014. Tuoteluettelo. Www -sivu. Luettu 13.3.2014.
http://www.kaapelikauppa.fi/catalog/hdmi-musta

Karvinen, T. Karvinen, K. 2010. Sulautetut. Opi rankentamaan robotteja ja muita sulautettuja järjestelmiä. 2. painos. Porvoo: WS Bookwell Oy

Moilanen, T. kuvauksen opettaja. 2013. Haastattelu. 25.9.2013. Haastattelija Mikko Yli-Viikari

MPU-6050 Accelerometer + Gyro. 2014. Www -sivu. Luettu 8.11.2014.
http://playground.arduino.cc/Main/MPU-6050

Mäkelä, M. Soininen, L. Tuomola, S. Öistämö, J. 2008. Tekniikan kaavasto. Matematiikan, fysiikan, kemian ja lujuusopin peruskaavoja sekä SI-järjestelmä.6. painos. Tampere: Tammertekniikka

Solid Signal. 2013. Broadcast & Pro A/V. Www -sivu. Luettu 22.10.2013.
http://www.solidsignal.com/pview.asp?p=hd1&d=aiptek-hd1-720p-high-definition-camcorder-with-buil-in-1gb-storage-%28hd-1%29

Sony. 2014. Broadcast & Pro A/V. Www -sivu. Luettu 13.3.2014.
http://www.sony.co.uk/pro/hub/home

Spidercam. 2013. Products. Www-sivu. Luettu 21.10.2013. http://www.spidercam.org

Spidercam. 2013. Technical. Www-sivu. Luettu 22.10.2013. http://www.spidercam.org

Tiffen. 2014. Press Release Merlin. Www -sivu. Luettu 7.4.2014.
http://www.tiffen.com/Press_Release_Merlin.htm

Videon peruskurssi v2 kuvaustekniikka. 2002. Www -sivu. Luettu 22.4.2014.
http://koti.mbnet.fi/pranta/vidper2.htm

LIITTEET

Liite 1. GY-521 mpu6050 datan lukeminen ja kolmen servon ohjaaminen

```
1  // MPU-6050 Accelerometer + Gyro
2  // ----------------------------
3  //
4  // By arduino.cc user "Krodal".
5  // June 2012
6  // Open Source / Public Domain
7  //
8  // Using Arduino 1.0.1
9  // It will not work with an older version,
10 // since Wire.endTransmission() uses a parameter
11 // to hold or release the I2C bus.
12 //
13 // Documentation:
14 // - The InvenSense documents:
15 //   - "MPU-6000 and MPU-6050 Product Specification",
16 //     PS-MPU-6000A.pdf
17 //   - "MPU-6000 and MPU-6050 Register Map and Descriptions",
18 //     RM-MPU-6000A.pdf or RS-MPU-6000A.pdf
19 //   - "MPU-6000/MPU-6050 9-Axis Evaluation Board User Guide"
20 //     AN-MPU-6000EVB.pdf
21 //
22 // The accuracy is 16-bits.
```

```
23  //
24  // Temperature sensor from -40 to +85 degrees Celsius
25  //   340 per degrees, -512 at 35 degrees.
26  //
27  // At power-up, all registers are zero, except these two:
28  //      Register 0x6B (PWR_MGMT_2) = 0x40  (I read zero).
29  //      Register 0x75 (WHO_AM_I)   = 0x68.
30  //
31
32  #include <Wire.h>
33
34
35  // The name of the sensor is "MPU-6050".
36  // For program code, I omit the '-',
37  // therefor I use the name "MPU6050....".
38
39
40  // Register names according to the datasheet.
41  // According to the InvenSense document
42  // "MPU-6000 and MPU-6050 Register Map
43  // and Descriptions Revision 3.2", there are no registers
44  // at 0x02 ... 0x18, but according other information
45  // the registers in that unknown area are for gain
46  // and offsets.
```

```
47  //
48  #define MPU6050_AUX_VDDIO          0x01   // R/W
49  #define MPU6050_SMPLRT_DIV         0x19   // R/W
50  #define MPU6050_CONFIG             0x1A   // R/W
51  #define MPU6050_GYRO_CONFIG        0x1B   // R/W
52  #define MPU6050_ACCEL_CONFIG       0x1C   // R/W
53  #define MPU6050_FF_THR             0x1D   // R/W
54  #define MPU6050_FF_DUR             0x1E   // R/W
55  #define MPU6050_MOT_THR            0x1F   // R/W
56  #define MPU6050_MOT_DUR            0x20   // R/W
57  #define MPU6050_ZRMOT_THR          0x21   // R/W
58  #define MPU6050_ZRMOT_DUR          0x22   // R/W
59  #define MPU6050_FIFO_EN            0x23   // R/W
60  #define MPU6050_I2C_MST_CTRL       0x24   // R/W
61  #define MPU6050_I2C_SLV0_ADDR      0x25   // R/W
62  #define MPU6050_I2C_SLV0_REG       0x26   // R/W
63  #define MPU6050_I2C_SLV0_CTRL      0x27   // R/W
64  #define MPU6050_I2C_SLV1_ADDR      0x28   // R/W
65  #define MPU6050_I2C_SLV1_REG       0x29   // R/W
66  #define MPU6050_I2C_SLV1_CTRL      0x2A   // R/W
67  #define MPU6050_I2C_SLV2_ADDR      0x2B   // R/W
68  #define MPU6050_I2C_SLV2_REG       0x2C   // R/W
69  #define MPU6050_I2C_SLV2_CTRL      0x2D   // R/W
70  #define MPU6050_I2C_SLV3_ADDR      0x2E   // R/W
```

```
71  #define MPU6050_I2C_SLV3_REG      0x2F   // R/W
72  #define MPU6050_I2C_SLV3_CTRL     0x30   // R/W
73  #define MPU6050_I2C_SLV4_ADDR     0x31   // R/W
74  #define MPU6050_I2C_SLV4_REG      0x32   // R/W
75  #define MPU6050_I2C_SLV4_DO       0x33   // R/W
76  #define MPU6050_I2C_SLV4_CTRL     0x34   // R/W
77  #define MPU6050_I2C_SLV4_DI       0x35   // R
78  #define MPU6050_I2C_MST_STATUS    0x36   // R
79  #define MPU6050_INT_PIN_CFG       0x37   // R/W
80  #define MPU6050_INT_ENABLE        0x38   // R/W
81  #define MPU6050_INT_STATUS        0x3A   // R
82  #define MPU6050_ACCEL_XOUT_H      0x3B   // R
83  #define MPU6050_ACCEL_XOUT_L      0x3C   // R
84  #define MPU6050_ACCEL_YOUT_H      0x3D   // R
85  #define MPU6050_ACCEL_YOUT_L      0x3E   // R
86  #define MPU6050_ACCEL_ZOUT_H      0x3F   // R
87  #define MPU6050_ACCEL_ZOUT_L      0x40   // R
88  #define MPU6050_TEMP_OUT_H        0x41   // R
89  #define MPU6050_TEMP_OUT_L        0x42   // R
90  #define MPU6050_GYRO_XOUT_H       0x43   // R
91  #define MPU6050_GYRO_XOUT_L       0x44   // R
92  #define MPU6050_GYRO_YOUT_H       0x45   // R
93  #define MPU6050_GYRO_YOUT_L       0x46   // R
94  #define MPU6050_GYRO_ZOUT_H       0x47   // R
```

```
95  #define MPU6050_GYRO_ZOUT_L       0x48   // R
96  #define MPU6050_EXT_SENS_DATA_00  0x49   // R
97  #define MPU6050_EXT_SENS_DATA_01  0x4A   // R
98  #define MPU6050_EXT_SENS_DATA_02  0x4B   // R
99  #define MPU6050_EXT_SENS_DATA_03  0x4C   // R
100 #define MPU6050_EXT_SENS_DATA_04  0x4D   // R
101 #define MPU6050_EXT_SENS_DATA_05  0x4E   // R
102 #define MPU6050_EXT_SENS_DATA_06  0x4F   // R
103 #define MPU6050_EXT_SENS_DATA_07  0x50   // R
104 #define MPU6050_EXT_SENS_DATA_08  0x51   // R
105 #define MPU6050_EXT_SENS_DATA_09  0x52   // R
106 #define MPU6050_EXT_SENS_DATA_10  0x53   // R
107 #define MPU6050_EXT_SENS_DATA_11  0x54   // R
108 #define MPU6050_EXT_SENS_DATA_12  0x55   // R
109 #define MPU6050_EXT_SENS_DATA_13  0x56   // R
110 #define MPU6050_EXT_SENS_DATA_14  0x57   // R
111 #define MPU6050_EXT_SENS_DATA_15  0x58   // R
112 #define MPU6050_EXT_SENS_DATA_16  0x59   // R
113 #define MPU6050_EXT_SENS_DATA_17  0x5A   // R
114 #define MPU6050_EXT_SENS_DATA_18  0x5B   // R
115 #define MPU6050_EXT_SENS_DATA_19  0x5C   // R
116 #define MPU6050_EXT_SENS_DATA_20  0x5D   // R
117 #define MPU6050_EXT_SENS_DATA_21  0x5E   // R
118 #define MPU6050_EXT_SENS_DATA_22  0x5F   // R
```

```
119     #define MPU6050_EXT_SENS_DATA_23   0x60   // R
120     #define MPU6050_MOT_DETECT_STATUS  0x61   // R
121     #define MPU6050_I2C_SLV0_DO        0x63   // R/W
122     #define MPU6050_I2C_SLV1_DO        0x64   // R/W
123     #define MPU6050_I2C_SLV2_DO        0x65   // R/W
124     #define MPU6050_I2C_SLV3_DO        0x66   // R/W
125     #define MPU6050_I2C_MST_DELAY_CTRL 0x67   // R/W
126     #define MPU6050_SIGNAL_PATH_RESET  0x68   // R/W
127     #define MPU6050_MOT_DETECT_CTRL    0x69   // R/W
128     #define MPU6050_USER_CTRL          0x6A   // R/W
129     #define MPU6050_PWR_MGMT_1         0x6B   // R/W
130     #define MPU6050_PWR_MGMT_2         0x6C   // R/W
131     #define MPU6050_FIFO_COUNTH        0x72   // R/W
132     #define MPU6050_FIFO_COUNTL        0x73   // R/W
133     #define MPU6050_FIFO_R_W           0x74   // R/W
134     #define MPU6050_WHO_AM_I           0x75   // R
135
136
137     // Defines for the bits, to be able to change
138     // between bit number and binary definition.
139     // By using the bit number, programming the sensor
140     // is like programming the AVR microcontroller.
141     // But instead of using "(1<<X)", or "_BV(X)",
142     // the Arduino "bit(X)" is used.
```

```
143     #define MPU6050_D0 0
144     #define MPU6050_D1 1
145     #define MPU6050_D2 2
146     #define MPU6050_D3 3
147     #define MPU6050_D4 4
148     #define MPU6050_D5 5
149     #define MPU6050_D6 6
150     #define MPU6050_D7 7
151
152     // AUX_VDDIO Register
153     #define MPU6050_AUX_VDDIO MPU6050_D7  // I2C high: 1=VDD, 0=VLOGIC
154
155     // CONFIG Register
156     // DLPF is Digital Low Pass Filter for both gyro and accelerometers.
157     // These are the names for the bits.
158     // Use these only with the bit() macro.
159     #define MPU6050_DLPF_CFG0     MPU6050_D0
160     #define MPU6050_DLPF_CFG1     MPU6050_D1
161     #define MPU6050_DLPF_CFG2     MPU6050_D2
162     #define MPU6050_EXT_SYNC_SET0 MPU6050_D3
163     #define MPU6050_EXT_SYNC_SET1 MPU6050_D4
164     #define MPU6050_EXT_SYNC_SET2 MPU6050_D5
165
166     // Combined definitions for the EXT_SYNC_SET values
```

```
167 #define MPU6050_EXT_SYNC_SET_0 (0)
168 #define MPU6050_EXT_SYNC_SET_1 (bit(MPU6050_EXT_SYNC_SET0))
169 #define MPU6050_EXT_SYNC_SET_2 (bit(MPU6050_EXT_SYNC_SET1))
170 #define MPU6050_EXT_SYNC_SET_3
171 (bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))
172 #define MPU6050_EXT_SYNC_SET_4 (bit(MPU6050_EXT_SYNC_SET2))
173 #define MPU6050_EXT_SYNC_SET_5
174 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET0))
175 #define MPU6050_EXT_SYNC_SET_6
176 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1))
177 #define MPU6050_EXT_SYNC_SET_7
178 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_
179 EXT_SYNC_SET0))
180
181 // Alternative names for the combined definitions.
182 #define MPU6050_EXT_SYNC_DISABLED     MPU6050_EXT_SYNC_SET_0
183 #define MPU6050_EXT_SYNC_TEMP_OUT_L   MPU6050_EXT_SYNC_SET_1
184 #define MPU6050_EXT_SYNC_GYRO_XOUT_L  MPU6050_EXT_SYNC_SET_2
185 #define MPU6050_EXT_SYNC_GYRO_YOUT_L  MPU6050_EXT_SYNC_SET_3
186 #define MPU6050_EXT_SYNC_GYRO_ZOUT_L  MPU6050_EXT_SYNC_SET_4
187 #define MPU6050_EXT_SYNC_ACCEL_XOUT_L MPU6050_EXT_SYNC_SET_5
188 #define MPU6050_EXT_SYNC_ACCEL_YOUT_L MPU6050_EXT_SYNC_SET_6
189 #define MPU6050_EXT_SYNC_ACCEL_ZOUT_L MPU6050_EXT_SYNC_SET_7
190
191 // Combined definitions for the DLPF_CFG values
```

```
192 #define MPU6050_DLPF_CFG_0 (0)
193 #define MPU6050_DLPF_CFG_1 (bit(MPU6050_DLPF_CFG0))
194 #define MPU6050_DLPF_CFG_2 (bit(MPU6050_DLPF_CFG1))
195 #define MPU6050_DLPF_CFG_3
196 (bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0))
197 #define MPU6050_DLPF_CFG_4 (bit(MPU6050_DLPF_CFG2))
198 #define MPU6050_DLPF_CFG_5
199 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG0))
200 #define MPU6050_DLPF_CFG_6
201 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1))
202 #define MPU6050_DLPF_CFG_7
203 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG
204 0))
205
206 // Alternative names for the combined definitions
207 // This name uses the bandwidth (Hz) for the accelometer,
208 // for the gyro the bandwidth is almost the same.
209 #define MPU6050_DLPF_260HZ  MPU6050_DLPF_CFG_0
210 #define MPU6050_DLPF_184HZ  MPU6050_DLPF_CFG_1
211 #define MPU6050_DLPF_94HZ  MPU6050_DLPF_CFG_2
212 #define MPU6050_DLPF_44HZ  MPU6050_DLPF_CFG_3
213 #define MPU6050_DLPF_21HZ  MPU6050_DLPF_CFG_4
214 #define MPU6050_DLPF_10HZ  MPU6050_DLPF_CFG_5
215 #define MPU6050_DLPF_5HZ  MPU6050_DLPF_CFG_6
216 #define MPU6050_DLPF_RESERVED MPU6050_DLPF_CFG_7
```

```
217
218     // GYRO_CONFIG Register
219     // The XG_ST, YG_ST, ZG_ST are bits for selftest.
220     // The FS_SEL sets the range for the gyro.
221     // These are the names for the bits.
222     // Use these only with the bit() macro.
223     #define MPU6050_FS_SEL0 MPU6050_D3
224     #define MPU6050_FS_SEL1 MPU6050_D4
225     #define MPU6050_ZG_ST   MPU6050_D5
226     #define MPU6050_YG_ST   MPU6050_D6
227     #define MPU6050_XG_ST   MPU6050_D7
228
229     // Combined definitions for the FS_SEL values
230     #define MPU6050_FS_SEL_0 (0)
231     #define MPU6050_FS_SEL_1 (bit(MPU6050_FS_SEL0))
232     #define MPU6050_FS_SEL_2 (bit(MPU6050_FS_SEL1))
233     #define MPU6050_FS_SEL_3 (bit(MPU6050_FS_SEL1)|bit(MPU6050_FS_SEL0))
234
235     // Alternative names for the combined definitions
236     // The name uses the range in degrees per second.
237     #define MPU6050_FS_SEL_250  MPU6050_FS_SEL_0
238     #define MPU6050_FS_SEL_500  MPU6050_FS_SEL_1
239     #define MPU6050_FS_SEL_1000 MPU6050_FS_SEL_2
240     #define MPU6050_FS_SEL_2000 MPU6050_FS_SEL_3
```

```
241
242     // ACCEL_CONFIG Register
243     // The XA_ST, YA_ST, ZA_ST are bits for selftest.
244     // The AFS_SEL sets the range for the accelerometer.
245     // These are the names for the bits.
246     // Use these only with the bit() macro.
247     #define MPU6050_ACCEL_HPF0 MPU6050_D0
248     #define MPU6050_ACCEL_HPF1 MPU6050_D1
249     #define MPU6050_ACCEL_HPF2 MPU6050_D2
250     #define MPU6050_AFS_SEL0   MPU6050_D3
251     #define MPU6050_AFS_SEL1   MPU6050_D4
252     #define MPU6050_ZA_ST      MPU6050_D5
253     #define MPU6050_YA_ST      MPU6050_D6
254     #define MPU6050_XA_ST      MPU6050_D7
255
256     // Combined definitions for the ACCEL_HPF values
257     #define MPU6050_ACCEL_HPF_0 (0)
258     #define MPU6050_ACCEL_HPF_1 (bit(MPU6050_ACCEL_HPF0))
259     #define MPU6050_ACCEL_HPF_2 (bit(MPU6050_ACCEL_HPF1))
260     #define MPU6050_ACCEL_HPF_3
261     (bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0))
262     #define MPU6050_ACCEL_HPF_4 (bit(MPU6050_ACCEL_HPF2))
263     #define MPU6050_ACCEL_HPF_7
264     (bit(MPU6050_ACCEL_HPF2)|bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL
265     _HPF0))
```

```
266
267     // Alternative names for the combined definitions
268     // The name uses the Cut-off frequency.
269     #define MPU6050_ACCEL_HPF_RESET  MPU6050_ACCEL_HPF_0
270     #define MPU6050_ACCEL_HPF_5HZ    MPU6050_ACCEL_HPF_1
271     #define MPU6050_ACCEL_HPF_2_5HZ  MPU6050_ACCEL_HPF_2
272     #define MPU6050_ACCEL_HPF_1_25HZ MPU6050_ACCEL_HPF_3
273     #define MPU6050_ACCEL_HPF_0_63HZ MPU6050_ACCEL_HPF_4
274     #define MPU6050_ACCEL_HPF_HOLD   MPU6050_ACCEL_HPF_7
275
276     // Combined definitions for the AFS_SEL values
277     #define MPU6050_AFS_SEL_0 (0)
278     #define MPU6050_AFS_SEL_1 (bit(MPU6050_AFS_SEL0))
279     #define MPU6050_AFS_SEL_2 (bit(MPU6050_AFS_SEL1))
280     #define MPU6050_AFS_SEL_3
281     (bit(MPU6050_AFS_SEL1)|bit(MPU6050_AFS_SEL0))
282
283     // Alternative names for the combined definitions
284     // The name uses the full scale range for the accelerometer.
285     #define MPU6050_AFS_SEL_2G  MPU6050_AFS_SEL_0
286     #define MPU6050_AFS_SEL_4G  MPU6050_AFS_SEL_1
287     #define MPU6050_AFS_SEL_8G  MPU6050_AFS_SEL_2
288     #define MPU6050_AFS_SEL_16G MPU6050_AFS_SEL_3
289
```

```
290     // FIFO_EN Register
291     // These are the names for the bits.
292     // Use these only with the bit() macro.
293     #define MPU6050_SLV0_FIFO_EN  MPU6050_D0
294     #define MPU6050_SLV1_FIFO_EN  MPU6050_D1
295     #define MPU6050_SLV2_FIFO_EN  MPU6050_D2
296     #define MPU6050_ACCEL_FIFO_EN MPU6050_D3
297     #define MPU6050_ZG_FIFO_EN    MPU6050_D4
298     #define MPU6050_YG_FIFO_EN    MPU6050_D5
299     #define MPU6050_XG_FIFO_EN    MPU6050_D6
300     #define MPU6050_TEMP_FIFO_EN  MPU6050_D7
301
302     // I2C_MST_CTRL Register
303     // These are the names for the bits.
304     // Use these only with the bit() macro.
305     #define MPU6050_I2C_MST_CLK0  MPU6050_D0
306     #define MPU6050_I2C_MST_CLK1  MPU6050_D1
307     #define MPU6050_I2C_MST_CLK2  MPU6050_D2
308     #define MPU6050_I2C_MST_CLK3  MPU6050_D3
309     #define MPU6050_I2C_MST_P_NSR MPU6050_D4
310     #define MPU6050_SLV_3_FIFO_EN MPU6050_D5
311     #define MPU6050_WAIT_FOR_ES   MPU6050_D6
312     #define MPU6050_MULT_MST_EN   MPU6050_D7
313
```

```
314 // Combined definitions for the I2C_MST_CLK
315 #define MPU6050_I2C_MST_CLK_0 (0)
316 #define MPU6050_I2C_MST_CLK_1  (bit(MPU6050_I2C_MST_CLK0))
317 #define MPU6050_I2C_MST_CLK_2  (bit(MPU6050_I2C_MST_CLK1))
318 #define MPU6050_I2C_MST_CLK_3
319 (bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
320 #define MPU6050_I2C_MST_CLK_4  (bit(MPU6050_I2C_MST_CLK2))
321 #define MPU6050_I2C_MST_CLK_5
322 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0))
323 #define MPU6050_I2C_MST_CLK_6
324 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1))
325 #define MPU6050_I2C_MST_CLK_7
326 (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C
327 _MST_CLK0))
328 #define MPU6050_I2C_MST_CLK_8  (bit(MPU6050_I2C_MST_CLK3))
329 #define MPU6050_I2C_MST_CLK_9
330 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK0))
331 #define MPU6050_I2C_MST_CLK_10
332 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1))
333 #define MPU6050_I2C_MST_CLK_11
334 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C
335 _MST_CLK0))
336 #define MPU6050_I2C_MST_CLK_12
337 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2))
338 #define MPU6050_I2C_MST_CLK_13
339 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C
340 _MST_CLK0))
```

```
341 #define MPU6050_I2C_MST_CLK_14
342 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C
343 _MST_CLK1))
344 #define MPU6050_I2C_MST_CLK_15
345 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C
346 _MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
347
348 // Alternative names for the combined definitions
349 // The names uses I2C Master Clock Speed in kHz.
350 #define MPU6050_I2C_MST_CLK_348KHZ MPU6050_I2C_MST_CLK_0
351 #define MPU6050_I2C_MST_CLK_333KHZ MPU6050_I2C_MST_CLK_1
352 #define MPU6050_I2C_MST_CLK_320KHZ MPU6050_I2C_MST_CLK_2
353 #define MPU6050_I2C_MST_CLK_308KHZ MPU6050_I2C_MST_CLK_3
354 #define MPU6050_I2C_MST_CLK_296KHZ MPU6050_I2C_MST_CLK_4
355 #define MPU6050_I2C_MST_CLK_286KHZ MPU6050_I2C_MST_CLK_5
356 #define MPU6050_I2C_MST_CLK_276KHZ MPU6050_I2C_MST_CLK_6
357 #define MPU6050_I2C_MST_CLK_267KHZ MPU6050_I2C_MST_CLK_7
358 #define MPU6050_I2C_MST_CLK_258KHZ MPU6050_I2C_MST_CLK_8
359 #define MPU6050_I2C_MST_CLK_500KHZ MPU6050_I2C_MST_CLK_9
360 #define MPU6050_I2C_MST_CLK_471KHZ MPU6050_I2C_MST_CLK_10
361 #define MPU6050_I2C_MST_CLK_444KHZ MPU6050_I2C_MST_CLK_11
362 #define MPU6050_I2C_MST_CLK_421KHZ MPU6050_I2C_MST_CLK_12
363 #define MPU6050_I2C_MST_CLK_400KHZ MPU6050_I2C_MST_CLK_13
364 #define MPU6050_I2C_MST_CLK_381KHZ MPU6050_I2C_MST_CLK_14
365 #define MPU6050_I2C_MST_CLK_364KHZ MPU6050_I2C_MST_CLK_15
```

```
366
367     // I2C_SLV0_ADDR Register
368     // These are the names for the bits.
369     // Use these only with the bit() macro.
370     #define MPU6050_I2C_SLV0_RW MPU6050_D7
371
372     // I2C_SLV0_CTRL Register
373     // These are the names for the bits.
374     // Use these only with the bit() macro.
375     #define MPU6050_I2C_SLV0_LEN0    MPU6050_D0
376     #define MPU6050_I2C_SLV0_LEN1    MPU6050_D1
377     #define MPU6050_I2C_SLV0_LEN2    MPU6050_D2
378     #define MPU6050_I2C_SLV0_LEN3    MPU6050_D3
379     #define MPU6050_I2C_SLV0_GRP     MPU6050_D4
380     #define MPU6050_I2C_SLV0_REG_DIS MPU6050_D5
381     #define MPU6050_I2C_SLV0_BYTE_SW MPU6050_D6
382     #define MPU6050_I2C_SLV0_EN      MPU6050_D7
383
384     // A mask for the length
385     #define MPU6050_I2C_SLV0_LEN_MASK 0x0F
386
387     // I2C_SLV1_ADDR Register
388     // These are the names for the bits.
389     // Use these only with the bit() macro.
```

```
390     #define MPU6050_I2C_SLV1_RW MPU6050_D7
391
392     // I2C_SLV1_CTRL Register
393     // These are the names for the bits.
394     // Use these only with the bit() macro.
395     #define MPU6050_I2C_SLV1_LEN0    MPU6050_D0
396     #define MPU6050_I2C_SLV1_LEN1    MPU6050_D1
397     #define MPU6050_I2C_SLV1_LEN2    MPU6050_D2
398     #define MPU6050_I2C_SLV1_LEN3    MPU6050_D3
399     #define MPU6050_I2C_SLV1_GRP     MPU6050_D4
400     #define MPU6050_I2C_SLV1_REG_DIS MPU6050_D5
401     #define MPU6050_I2C_SLV1_BYTE_SW MPU6050_D6
402     #define MPU6050_I2C_SLV1_EN      MPU6050_D7
403
404     // A mask for the length
405     #define MPU6050_I2C_SLV1_LEN_MASK 0x0F
406
407     // I2C_SLV2_ADDR Register
408     // These are the names for the bits.
409     // Use these only with the bit() macro.
410     #define MPU6050_I2C_SLV2_RW MPU6050_D7
411
412     // I2C_SLV2_CTRL Register
413     // These are the names for the bits.
```

```
414     // Use these only with the bit() macro.
415     #define MPU6050_I2C_SLV2_LEN0    MPU6050_D0
416     #define MPU6050_I2C_SLV2_LEN1    MPU6050_D1
417     #define MPU6050_I2C_SLV2_LEN2    MPU6050_D2
418     #define MPU6050_I2C_SLV2_LEN3    MPU6050_D3
419     #define MPU6050_I2C_SLV2_GRP     MPU6050_D4
420     #define MPU6050_I2C_SLV2_REG_DIS MPU6050_D5
421     #define MPU6050_I2C_SLV2_BYTE_SW MPU6050_D6
422     #define MPU6050_I2C_SLV2_EN      MPU6050_D7
423
424     // A mask for the length
425     #define MPU6050_I2C_SLV2_LEN_MASK 0x0F
426
427     // I2C_SLV3_ADDR Register
428     // These are the names for the bits.
429     // Use these only with the bit() macro.
430     #define MPU6050_I2C_SLV3_RW MPU6050_D7
431
432     // I2C_SLV3_CTRL Register
433     // These are the names for the bits.
434     // Use these only with the bit() macro.
435     #define MPU6050_I2C_SLV3_LEN0    MPU6050_D0
436     #define MPU6050_I2C_SLV3_LEN1    MPU6050_D1
437     #define MPU6050_I2C_SLV3_LEN2    MPU6050_D2
```

```
438     #define MPU6050_I2C_SLV3_LEN3    MPU6050_D3
439     #define MPU6050_I2C_SLV3_GRP     MPU6050_D4
440     #define MPU6050_I2C_SLV3_REG_DIS MPU6050_D5
441     #define MPU6050_I2C_SLV3_BYTE_SW MPU6050_D6
442     #define MPU6050_I2C_SLV3_EN      MPU6050_D7
443
444     // A mask for the length
445     #define MPU6050_I2C_SLV3_LEN_MASK 0x0F
446
447     // I2C_SLV4_ADDR Register
448     // These are the names for the bits.
449     // Use these only with the bit() macro.
450     #define MPU6050_I2C_SLV4_RW MPU6050_D7
451
452     // I2C_SLV4_CTRL Register
453     // These are the names for the bits.
454     // Use these only with the bit() macro.
455     #define MPU6050_I2C_MST_DLY0     MPU6050_D0
456     #define MPU6050_I2C_MST_DLY1     MPU6050_D1
457     #define MPU6050_I2C_MST_DLY2     MPU6050_D2
458     #define MPU6050_I2C_MST_DLY3     MPU6050_D3
459     #define MPU6050_I2C_MST_DLY4     MPU6050_D4
460     #define MPU6050_I2C_SLV4_REG_DIS MPU6050_D5
461     #define MPU6050_I2C_SLV4_INT_EN  MPU6050_D6
```

```
462     #define MPU6050_I2C_SLV4_EN      MPU6050_D7
463
464     // A mask for the delay
465     #define MPU6050_I2C_MST_DLY_MASK 0x1F
466
467     // I2C_MST_STATUS Register
468     // These are the names for the bits.
469     // Use these only with the bit() macro.
470     #define MPU6050_I2C_SLV0_NACK MPU6050_D0
471     #define MPU6050_I2C_SLV1_NACK MPU6050_D1
472     #define MPU6050_I2C_SLV2_NACK MPU6050_D2
473     #define MPU6050_I2C_SLV3_NACK MPU6050_D3
474     #define MPU6050_I2C_SLV4_NACK MPU6050_D4
475     #define MPU6050_I2C_LOST_ARB  MPU6050_D5
476     #define MPU6050_I2C_SLV4_DONE MPU6050_D6
477     #define MPU6050_PASS_THROUGH  MPU6050_D7
478
479     // I2C_PIN_CFG Register
480     // These are the names for the bits.
481     // Use these only with the bit() macro.
482     #define MPU6050_CLKOUT_EN       MPU6050_D0
483     #define MPU6050_I2C_BYPASS_EN   MPU6050_D1
484     #define MPU6050_FSYNC_INT_EN    MPU6050_D2
485     #define MPU6050_FSYNC_INT_LEVEL MPU6050_D3
```

```
486     #define MPU6050_INT_RD_CLEAR    MPU6050_D4
487     #define MPU6050_LATCH_INT_EN    MPU6050_D5
488     #define MPU6050_INT_OPEN        MPU6050_D6
489     #define MPU6050_INT_LEVEL       MPU6050_D7
490
491     // INT_ENABLE Register
492     // These are the names for the bits.
493     // Use these only with the bit() macro.
494     #define MPU6050_DATA_RDY_EN    MPU6050_D0
495     #define MPU6050_I2C_MST_INT_EN MPU6050_D3
496     #define MPU6050_FIFO_OFLOW_EN  MPU6050_D4
497     #define MPU6050_ZMOT_EN        MPU6050_D5
498     #define MPU6050_MOT_EN         MPU6050_D6
499     #define MPU6050_FF_EN          MPU6050_D7
500
501     // INT_STATUS Register
502     // These are the names for the bits.
503     // Use these only with the bit() macro.
504     #define MPU6050_DATA_RDY_INT   MPU6050_D0
505     #define MPU6050_I2C_MST_INT    MPU6050_D3
506     #define MPU6050_FIFO_OFLOW_INT MPU6050_D4
507     #define MPU6050_ZMOT_INT       MPU6050_D5
508     #define MPU6050_MOT_INT        MPU6050_D6
509     #define MPU6050_FF_INT         MPU6050_D7
```

```
510
511     // MOT_DETECT_STATUS Register
512     // These are the names for the bits.
513     // Use these only with the bit() macro.
514     #define MPU6050_MOT_ZRMOT MPU6050_D0
515     #define MPU6050_MOT_ZPOS  MPU6050_D2
516     #define MPU6050_MOT_ZNEG  MPU6050_D3
517     #define MPU6050_MOT_YPOS  MPU6050_D4
518     #define MPU6050_MOT_YNEG  MPU6050_D5
519     #define MPU6050_MOT_XPOS  MPU6050_D6
520     #define MPU6050_MOT_XNEG  MPU6050_D7
521
522     // IC2_MST_DELAY_CTRL Register
523     // These are the names for the bits.
524     // Use these only with the bit() macro.
525     #define MPU6050_I2C_SLV0_DLY_EN MPU6050_D0
526     #define MPU6050_I2C_SLV1_DLY_EN MPU6050_D1
527     #define MPU6050_I2C_SLV2_DLY_EN MPU6050_D2
528     #define MPU6050_I2C_SLV3_DLY_EN MPU6050_D3
529     #define MPU6050_I2C_SLV4_DLY_EN MPU6050_D4
530     #define MPU6050_DELAY_ES_SHADOW MPU6050_D7
531
532     // SIGNAL_PATH_RESET Register
533     // These are the names for the bits.
```

```
534 // Use these only with the bit() macro.
535 #define MPU6050_TEMP_RESET  MPU6050_D0
536 #define MPU6050_ACCEL_RESET MPU6050_D1
537 #define MPU6050_GYRO_RESET  MPU6050_D2
538
539 // MOT_DETECT_CTRL Register
540 // These are the names for the bits.
541 // Use these only with the bit() macro.
542 #define MPU6050_MOT_COUNT0      MPU6050_D0
543 #define MPU6050_MOT_COUNT1      MPU6050_D1
544 #define MPU6050_FF_COUNT0       MPU6050_D2
545 #define MPU6050_FF_COUNT1       MPU6050_D3
546 #define MPU6050_ACCEL_ON_DELAY0 MPU6050_D4
547 #define MPU6050_ACCEL_ON_DELAY1 MPU6050_D5
548
549 // Combined definitions for the MOT_COUNT
550 #define MPU6050_MOT_COUNT_0 (0)
551 #define MPU6050_MOT_COUNT_1 (bit(MPU6050_MOT_COUNT0))
552 #define MPU6050_MOT_COUNT_2 (bit(MPU6050_MOT_COUNT1))
553 #define MPU6050_MOT_COUNT_3
554 (bit(MPU6050_MOT_COUNT1)|bit(MPU6050_MOT_COUNT0))
555
556 // Alternative names for the combined definitions
557 #define MPU6050_MOT_COUNT_RESET MPU6050_MOT_COUNT_0
```

```
558
559     // Combined definitions for the FF_COUNT
560     #define MPU6050_FF_COUNT_0 (0)
561     #define MPU6050_FF_COUNT_1 (bit(MPU6050_FF_COUNT0))
562     #define MPU6050_FF_COUNT_2 (bit(MPU6050_FF_COUNT1))
563     #define MPU6050_FF_COUNT_3
564     (bit(MPU6050_FF_COUNT1)|bit(MPU6050_FF_COUNT0))
565
566     // Alternative names for the combined definitions
567     #define MPU6050_FF_COUNT_RESET MPU6050_FF_COUNT_0
568
569     // Combined definitions for the ACCEL_ON_DELAY
570     #define MPU6050_ACCEL_ON_DELAY_0 (0)
571     #define MPU6050_ACCEL_ON_DELAY_1 (bit(MPU6050_ACCEL_ON_DELAY0))
572     #define MPU6050_ACCEL_ON_DELAY_2 (bit(MPU6050_ACCEL_ON_DELAY1))
573     #define MPU6050_ACCEL_ON_DELAY_3
574     (bit(MPU6050_ACCEL_ON_DELAY1)|bit(MPU6050_ACCEL_ON_DELAY0))
575
576     // Alternative names for the ACCEL_ON_DELAY
577     #define MPU6050_ACCEL_ON_DELAY_0MS MPU6050_ACCEL_ON_DELAY_0
578     #define MPU6050_ACCEL_ON_DELAY_1MS MPU6050_ACCEL_ON_DELAY_1
579     #define MPU6050_ACCEL_ON_DELAY_2MS MPU6050_ACCEL_ON_DELAY_2
580     #define MPU6050_ACCEL_ON_DELAY_3MS MPU6050_ACCEL_ON_DELAY_3
581
```

```
582     // USER_CTRL Register
583     // These are the names for the bits.
584     // Use these only with the bit() macro.
585     #define MPU6050_SIG_COND_RESET MPU6050_D0
586     #define MPU6050_I2C_MST_RESET  MPU6050_D1
587     #define MPU6050_FIFO_RESET     MPU6050_D2
588     #define MPU6050_I2C_IF_DIS     MPU6050_D4   // must be 0 for MPU-6050
589     #define MPU6050_I2C_MST_EN     MPU6050_D5
590     #define MPU6050_FIFO_EN        MPU6050_D6
591
592     // PWR_MGMT_1 Register
593     // These are the names for the bits.
594     // Use these only with the bit() macro.
595     #define MPU6050_CLKSEL0      MPU6050_D0
596     #define MPU6050_CLKSEL1      MPU6050_D1
597     #define MPU6050_CLKSEL2      MPU6050_D2
598     #define MPU6050_TEMP_DIS     MPU6050_D3    // 1: disable temperature sensor
599     #define MPU6050_CYCLE        MPU6050_D5    // 1: sample and sleep
600     #define MPU6050_SLEEP        MPU6050_D6    // 1: sleep mode
601     #define MPU6050_DEVICE_RESET MPU6050_D7    // 1: reset to default values
602
603     // Combined definitions for the CLKSEL
604     #define MPU6050_CLKSEL_0 (0)
605     #define MPU6050_CLKSEL_1 (bit(MPU6050_CLKSEL0))
```

```
606     #define MPU6050_CLKSEL_2 (bit(MPU6050_CLKSEL1))
607     #define MPU6050_CLKSEL_3 (bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))
608     #define MPU6050_CLKSEL_4 (bit(MPU6050_CLKSEL2))
609     #define MPU6050_CLKSEL_5 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL0))
610     #define MPU6050_CLKSEL_6 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1))
611     #define MPU6050_CLKSEL_7
612     (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))
613
614     // Alternative names for the combined definitions
615     #define MPU6050_CLKSEL_INTERNAL    MPU6050_CLKSEL_0
616     #define MPU6050_CLKSEL_X           MPU6050_CLKSEL_1
617     #define MPU6050_CLKSEL_Y           MPU6050_CLKSEL_2
618     #define MPU6050_CLKSEL_Z           MPU6050_CLKSEL_3
619     #define MPU6050_CLKSEL_EXT_32KHZ   MPU6050_CLKSEL_4
620     #define MPU6050_CLKSEL_EXT_19_2MHZ MPU6050_CLKSEL_5
621     #define MPU6050_CLKSEL_RESERVED    MPU6050_CLKSEL_6
622     #define MPU6050_CLKSEL_STOP        MPU6050_CLKSEL_7
623
624     // PWR_MGMT_2 Register
625     // These are the names for the bits.
626     // Use these only with the bit() macro.
627     #define MPU6050_STBY_ZG       MPU6050_D0
628     #define MPU6050_STBY_YG       MPU6050_D1
629     #define MPU6050_STBY_XG       MPU6050_D2
```

```
630     #define MPU6050_STBY_ZA       MPU6050_D3
631     #define MPU6050_STBY_YA       MPU6050_D4
632     #define MPU6050_STBY_XA       MPU6050_D5
633     #define MPU6050_LP_WAKE_CTRL0 MPU6050_D6
634     #define MPU6050_LP_WAKE_CTRL1 MPU6050_D7
635
636     // Combined definitions for the LP_WAKE_CTRL
637     #define MPU6050_LP_WAKE_CTRL_0 (0)
638     #define MPU6050_LP_WAKE_CTRL_1 (bit(MPU6050_LP_WAKE_CTRL0))
639     #define MPU6050_LP_WAKE_CTRL_2 (bit(MPU6050_LP_WAKE_CTRL1))
640     #define MPU6050_LP_WAKE_CTRL_3
641     (bit(MPU6050_LP_WAKE_CTRL1)|bit(MPU6050_LP_WAKE_CTRL0))
642
643     // Alternative names for the combined definitions
644     // The names uses the Wake-up Frequency.
645     #define MPU6050_LP_WAKE_1_25HZ MPU6050_LP_WAKE_CTRL_0
646     #define MPU6050_LP_WAKE_2_5HZ  MPU6050_LP_WAKE_CTRL_1
647     #define MPU6050_LP_WAKE_5HZ    MPU6050_LP_WAKE_CTRL_2
648     #define MPU6050_LP_WAKE_10HZ   MPU6050_LP_WAKE_CTRL_3
649
650
651     // Default I2C address for the MPU-6050 is 0x68.
652     // But only if the AD0 pin is low.
653     // Some sensor boards have AD0 high, and the
```

```
654     // I2C address thus becomes 0x69.
655     #define MPU6050_I2C_ADDRESS 0x68
656
657
658     // Declaring an union for the registers and the axis values.
659     // The byte order does not match the byte order of
660     // the compiler and AVR chip.
661     // The AVR chip (on the Arduino board) has the Low Byte
662     // at the lower address.
663     // But the MPU-6050 has a different order: High Byte at
664     // lower address, so that has to be corrected.
665     // The register part "reg" is only used internally,
666     // and are swapped in code.
667     typedef union accel_t_gyro_union
668     {
669       struct
670       {
671         uint8_t x_accel_h;
672         uint8_t x_accel_l;
673         uint8_t y_accel_h;
674         uint8_t y_accel_l;
675         uint8_t z_accel_h;
676         uint8_t z_accel_l;
677         uint8_t t_h;
```

```
678         uint8_t t_l;
679         uint8_t x_gyro_h;
680         uint8_t x_gyro_l;
681         uint8_t y_gyro_h;
682         uint8_t y_gyro_l;
683         uint8_t z_gyro_h;
684         uint8_t z_gyro_l;
685       } reg;
686       struct
687       {
688         int x_accel;
689         int y_accel;
690         int z_accel;
691         int temperature;
692         int x_gyro;
693         int y_gyro;
694         int z_gyro;
695       } value;
696     };
697
698     // Use the following global variables and access functions to help store the overall
699     // rotation angle of the sensor
700     unsigned long last_read_time;
701     float         last_x_angle;  // These are the filtered angles
```

```
float       last_y_angle;
float       last_z_angle;
float       last_gyro_x_angle;  // Store the gyro angles to compare drift
float       last_gyro_y_angle;
float       last_gyro_z_angle;

void set_last_read_angle_data(unsigned long time, float x, float y, float z, float x_gyro,
float y_gyro, float z_gyro) {
  last_read_time = time;
  last_x_angle = x;
  last_y_angle = y;
  last_z_angle = z;
  last_gyro_x_angle = x_gyro;
  last_gyro_y_angle = y_gyro;
  last_gyro_z_angle = z_gyro;
}

inline unsigned long get_last_time() {return last_read_time;}
inline float get_last_x_angle() {return last_x_angle;}
inline float get_last_y_angle() {return last_y_angle;}
inline float get_last_z_angle() {return last_z_angle;}
inline float get_last_gyro_x_angle() {return last_gyro_x_angle;}
inline float get_last_gyro_y_angle() {return last_gyro_y_angle;}
inline float get_last_gyro_z_angle() {return last_gyro_z_angle;}
```

```
726
727     //  Use the following global variables and access functions
728     //  to calibrate the acceleration sensor
729     float    base_x_accel;
730     float    base_y_accel;
731     float    base_z_accel;
732
733     float    base_x_gyro;
734     float    base_y_gyro;
735     float    base_z_gyro;
736
737
738     int read_gyro_accel_vals(uint8_t* accel_t_gyro_ptr) {
739       // Read the raw values.
740       // Read 14 bytes at once,
741       // containing acceleration, temperature and gyro.
742       // With the default settings of the MPU-6050,
743       // there is no filter enabled, and the values
744       // are not very stable.  Returns the error value
745
746       accel_t_gyro_union* accel_t_gyro = (accel_t_gyro_union *) accel_t_gyro_ptr;
747
748       int error = MPU6050_read (MPU6050_ACCEL_XOUT_H, (uint8_t *) accel_t_gyro,
749     sizeof(*accel_t_gyro));
```

```
750
751     // Swap all high and low bytes.
752     // After this, the registers values are swapped,
753     // so the structure name like x_accel_l does no
754     // longer contain the lower byte.
755     uint8_t swap;
756     #define SWAP(x,y) swap = x; x = y; y = swap
757
758     SWAP ((*accel_t_gyro).reg.x_accel_h, (*accel_t_gyro).reg.x_accel_l);
759     SWAP ((*accel_t_gyro).reg.y_accel_h, (*accel_t_gyro).reg.y_accel_l);
760     SWAP ((*accel_t_gyro).reg.z_accel_h, (*accel_t_gyro).reg.z_accel_l);
761     SWAP ((*accel_t_gyro).reg.t_h, (*accel_t_gyro).reg.t_l);
762     SWAP ((*accel_t_gyro).reg.x_gyro_h, (*accel_t_gyro).reg.x_gyro_l);
763     SWAP ((*accel_t_gyro).reg.y_gyro_h, (*accel_t_gyro).reg.y_gyro_l);
764     SWAP ((*accel_t_gyro).reg.z_gyro_h, (*accel_t_gyro).reg.z_gyro_l);
765
766     return error;
767   }
768
769   // The sensor should be motionless on a horizontal surface
770   //  while calibration is happening
771   void calibrate_sensors() {
772     int                 num_readings = 10;
773     float               x_accel = 0;
```

```
774     float           y_accel = 0;
775     float           z_accel = 0;
776     float           x_gyro = 0;
777     float           y_gyro = 0;
778     float           z_gyro = 0;
779     accel_t_gyro_union    accel_t_gyro;
780
781     //Serial.println("Starting Calibration");
782
783     // Discard the first set of values read from the IMU
784     read_gyro_accel_vals((uint8_t *) &accel_t_gyro);
785
786     // Read and average the raw values from the IMU
787     for (int i = 0; i < num_readings; i++) {
788       read_gyro_accel_vals((uint8_t *) &accel_t_gyro);
789       x_accel += accel_t_gyro.value.x_accel;
790       y_accel += accel_t_gyro.value.y_accel;
791       z_accel += accel_t_gyro.value.z_accel;
792       x_gyro += accel_t_gyro.value.x_gyro;
793       y_gyro += accel_t_gyro.value.y_gyro;
794       z_gyro += accel_t_gyro.value.z_gyro;
795       delay(100);
796     }
797     x_accel /= num_readings;
```

```
798       y_accel /= num_readings;
799       z_accel /= num_readings;
800       x_gyro /= num_readings;
801       y_gyro /= num_readings;
802       z_gyro /= num_readings;
803
804       // Store the raw calibration values globally
805       base_x_accel = x_accel;
806       base_y_accel = y_accel;
807       base_z_accel = z_accel;
808       base_x_gyro = x_gyro;
809       base_y_gyro = y_gyro;
810       base_z_gyro = z_gyro;
811
812       //Serial.println("Finishing Calibration");
813     }
814
815     //Lisäyksiä koodiin-------
816     #include<Servo.h>
817
818     Servo servo1;
819     Servo servo2;
820     Servo servo3;
821
```

```
822   int digi=11;
823
824
825   //-------------------------
826   void setup()
827   {
828     int error;
829     uint8_t c;
830
831
832     Serial.begin(19200);
833     /*
834     Serial.println(F("InvenSense MPU-6050"));
835     Serial.println(F("June 2012"));
836     */
837     // Initialize the 'Wire' class for the I2C-bus.
838     Wire.begin();
839
840
841     // default at power-up:
842     //    Gyro at 250 degrees second
843     //    Acceleration at 2g
844     //    Clock source at internal 8MHz
845     //    The device is in sleep mode.
```

```
846     //
847
848     error = MPU6050_read (MPU6050_WHO_AM_I, &c, 1);
849     /*
850     Serial.print(F("WHO_AM_I : "));
851     Serial.print(c,HEX);
852     Serial.print(F(", error = "));
853     Serial.println(error,DEC);
854     */
855
856     // According to the datasheet, the 'sleep' bit
857     // should read a '1'. But I read a '0'.
858     // That bit has to be cleared, since the sensor
859     // is in sleep mode at power-up. Even if the
860     // bit reads '0'.
861     error = MPU6050_read (MPU6050_PWR_MGMT_2, &c, 1);
862     /*
863     Serial.print(F("PWR_MGMT_2 : "));
864     Serial.print(c,HEX);
865     Serial.print(F(", error = "));
866     Serial.println(error,DEC);
867     */
868
869     // Clear the 'sleep' bit to start the sensor.
```

```
870       MPU6050_write_reg (MPU6050_PWR_MGMT_1, 0);
871
872       //Initialize the angles
873       calibrate_sensors();
874       set_last_read_angle_data(millis(), 0, 0, 0, 0, 0, 0);
875
876       //----Lisäyksiä koodiin----
877       pinMode (11, INPUT);
878       servo1.attach(8);
879       servo2.attach(9);
880       servo3.attach(10);
881       //-------------------------
882     }
883
884
885     void loop()
886     {
887       //Lisäyksiä koodiin-----
888        digi=map(pulseIn(11,HIGH,25000),1199,1834,180,0);
889       //-----------
890
891       int error;
892       double dT;
893       accel_t_gyro_union accel_t_gyro;
```

```
894
895       /*
896       Serial.println(F(""));
897       Serial.println(F("MPU-6050"));
898       */
899
900       // Read the raw values.
901       error = read_gyro_accel_vals((uint8_t*) &accel_t_gyro);
902
903       // Get the time of reading for rotation computations
904       unsigned long t_now = millis();
905
906     /*
907       Serial.print(F("Read accel, temp and gyro, error = "));
908       Serial.println(error,DEC);
909
910
911       // Print the raw acceleration values
912       Serial.print(F("accel x,y,z: "));
913       Serial.print(accel_t_gyro.value.x_accel, DEC);
914       Serial.print(F(", "));
915       Serial.print(accel_t_gyro.value.y_accel, DEC);
916       Serial.print(F(", "));
917       Serial.print(accel_t_gyro.value.z_accel, DEC);
```

```
918       Serial.println(F(""));
919     */
920
921       // The temperature sensor is -40 to +85 degrees Celsius.
922       // It is a signed integer.
923       // According to the datasheet:
924       //   340 per degrees Celsius, -512 at 35 degrees.
925       // At 0 degrees: -512 - (340 * 35) = -12412
926     /*
927       Serial.print(F("temperature: "));
928       dT = ( (double) accel_t_gyro.value.temperature + 12412.0) / 340.0;
929       Serial.print(dT, 3);
930       Serial.print(F(" degrees Celsius"));
931       Serial.println(F(""));
932
933
934       // Print the raw gyro values.
935       Serial.print(F("raw gyro x,y,z : "));
936       Serial.print(accel_t_gyro.value.x_gyro, DEC);
937       Serial.print(F(", "));
938       Serial.print(accel_t_gyro.value.y_gyro, DEC);
939       Serial.print(F(", "));
940       Serial.print(accel_t_gyro.value.z_gyro, DEC);
941       Serial.print(F(", "));
```

```
942       Serial.println(F(""));
943     */
944
945       // Convert gyro values to degrees/sec
946       float FS_SEL = 131;
947       /*
948       float gyro_x = (accel_t_gyro.value.x_gyro - base_x_gyro)/FS_SEL;
949       float gyro_y = (accel_t_gyro.value.y_gyro - base_y_gyro)/FS_SEL;
950       float gyro_z = (accel_t_gyro.value.z_gyro - base_z_gyro)/FS_SEL;
951       */
952       float gyro_x = (accel_t_gyro.value.x_gyro - base_x_gyro)/FS_SEL;
953       float gyro_y = (accel_t_gyro.value.y_gyro - base_y_gyro)/FS_SEL;
954       float gyro_z = (accel_t_gyro.value.z_gyro - base_z_gyro)/FS_SEL;
955
956
957       // Get raw acceleration values
958       //float G_CONVERT = 16384;
959       float accel_x = accel_t_gyro.value.x_accel;
960       float accel_y = accel_t_gyro.value.y_accel;
961       float accel_z = accel_t_gyro.value.z_accel;
962
963       // Get angle values from accelerometer
964       float RADIANS_TO_DEGREES = 180/3.14159;
965      //  float accel_vector_length = sqrt(pow(accel_x,2) + pow(accel_y,2) + pow(accel_z,2));
```

```
966     float accel_angle_y = atan(-1*accel_x/sqrt(pow(accel_y,2) +
967   pow(accel_z,2)))*RADIANS_TO_DEGREES;
968     float accel_angle_x = atan(accel_y/sqrt(pow(accel_x,2) +
969   pow(accel_z,2)))*RADIANS_TO_DEGREES;
970
971     float accel_angle_z = 0;
972
973     // Compute the (filtered) gyro angles
974     float dt =(t_now - get_last_time())/1000.0;
975     float gyro_angle_x = gyro_x*dt + get_last_x_angle();
976     float gyro_angle_y = gyro_y*dt + get_last_y_angle();
977     float gyro_angle_z = gyro_z*dt + get_last_z_angle();
978
979     // Compute the drifting gyro angles
980     float unfiltered_gyro_angle_x = gyro_x*dt + get_last_gyro_x_angle();
981     float unfiltered_gyro_angle_y = gyro_y*dt + get_last_gyro_y_angle();
982     float unfiltered_gyro_angle_z = gyro_z*dt + get_last_gyro_z_angle();
983
984     // Apply the complementary filter to figure out the change in angle - choice of alpha is
985     // estimated now.  Alpha depends on the sampling rate...
986     float alpha = 0.96;
987     float angle_x = alpha*gyro_angle_x + (1.0 - alpha)*accel_angle_x;
988     float angle_y = alpha*gyro_angle_y + (1.0 - alpha)*accel_angle_y;
989     float angle_z = gyro_angle_z;  //Accelerometer doesn't give z-angle
```

```
990
991     // Update the saved data with the latest values
992     set_last_read_angle_data(t_now, angle_x, angle_y, angle_z, unfiltered_gyro_angle_x,
993   unfiltered_gyro_angle_y, unfiltered_gyro_angle_z);
994   /*
995     // Send the data to the serial port
996     Serial.print(F("DEL:"));            //Delta T
997     Serial.print(dt, DEC);
998     Serial.print(F("#ACC:"));           //Accelerometer angle
999     Serial.print(accel_angle_x, 2);
1000    Serial.print(F(","));
1001    Serial.print(accel_angle_y, 2);
1002    Serial.print(F(","));
1003    Serial.print(accel_angle_z, 2);
1004    Serial.print(F("#GYR:"));
1005    Serial.print(unfiltered_gyro_angle_x, 2);      //Gyroscope angle
1006    Serial.print(F(","));
1007    Serial.print(unfiltered_gyro_angle_y, 2);
1008    Serial.print(F(","));
1009    Serial.print(unfiltered_gyro_angle_z, 2);
1010    */
1011    Serial.print(F("#FIL:"));           //Filtered angle
1012    Serial.print(angle_x, 2);
1013    Serial.print(F(","));
```

```
1014      Serial.print(angle_y, 2);
1015      Serial.print(F(","));
1016      Serial.print(angle_z, 2);
1017      Serial.print(F(","));
1018      Serial.print(digi);
1019      Serial.print(F(","));
1020      Serial.print(angle_x+digi);
1021      Serial.println(F(""));
1022
1023      //----Lisäyksiä koodiin----
1024      servo1.write(angle_x);
1025      servo2.write(angle_y);
1026      servo3.write(angle_z);
1027      //-------------------------
1028
1029      // Delay so we don't swamp the serial port
1030      delay(5);
1031    }
1032
1033
1034    // ----------------------------------------------------
1035    // MPU6050_read
1036    //
1037    // This is a common function to read multiple bytes
```

```
1038    // from an I2C device.
1039    //
1040    // It uses the boolean parameter for Wire.endTransMission()
1041    // to be able to hold or release the I2C-bus.
1042    // This is implemented in Arduino 1.0.1.
1043    //
1044    // Only this function is used to read.
1045    // There is no function for a single byte.
1046    //
1047    int MPU6050_read(int start, uint8_t *buffer, int size)
1048    {
1049      int i, n, error;
1050
1051      Wire.beginTransmission(MPU6050_I2C_ADDRESS);
1052      n = Wire.write(start);
1053      if (n != 1)
1054        return (-10);
1055
1056      n = Wire.endTransmission(false);    // hold the I2C-bus
1057      if (n != 0)
1058        return (n);
1059
1060      // Third parameter is true: relase I2C-bus after data is read.
1061      Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);
```

```
1062      i = 0;
1063      while(Wire.available() && i<size)
1064      {
1065        buffer[i++]=Wire.read();
1066      }
1067      if ( i != size)
1068        return (-11);
1069
1070      return (0);  // return : no error
1071    }
1072
1073
1074    // ----------------------------------------------------------
1075    // MPU6050_write
1076    //
1077    // This is a common function to write multiple bytes to an I2C device.
1078    //
1079    // If only a single register is written,
1080    // use the function MPU_6050_write_reg().
1081    //
1082    // Parameters:
1083    //   start : Start address, use a define for the register
1084    //   pData : A pointer to the data to write.
1085    //   size  : The number of bytes to write.
```

```
1086    //
1087    // If only a single register is written, a pointer
1088    // to the data has to be used, and the size is
1089    // a single byte:
1090    //   int data = 0;        // the data to write
1091    //   MPU6050_write (MPU6050_PWR_MGMT_1, &c, 1);
1092    //
1093    int MPU6050_write(int start, const uint8_t *pData, int size)
1094    {
1095      int n, error;
1096
1097      Wire.beginTransmission(MPU6050_I2C_ADDRESS);
1098      n = Wire.write(start);        // write the start address
1099      if (n != 1)
1100        return (-20);
1101
1102      n = Wire.write(pData, size);  // write data bytes
1103      if (n != size)
1104        return (-21);
1105
1106      error = Wire.endTransmission(true); // release the I2C-bus
1107      if (error != 0)
1108        return (error);
1109
```

```
1110      return (0);         // return : no error
1111    }
1112
1113    // ------------------------------------------------------
1114    // MPU6050_write_reg
1115    //
1116    // An extra function to write a single register.
1117    // It is just a wrapper around the MPU_6050_write()
1118    // function, and it is only a convenient function
1119    // to make it easier to write a single register.
1120    //
1121    int MPU6050_write_reg(int reg, uint8_t data)
1122    {
1123      int error;
1124
1125      error = MPU6050_write(reg, &data, 1);
1126
1127      return (error);
1128    }
```