Robert Juhakoski

FRISBEEGOLFMITTALAITTEEN SUUNNITTELU JA TOTEUTUS

Opinnäytetyö

Tekniikan ammattikorkeakoulututkinto

Insinööri (AMK), sähkö- ja automaatiotekniikka

2025

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Insinööri (AMK)
Tekijä/Tekijät	Robert Juhakoski
Työn nimi	Frisbeegolfmittalaitteen suunnittelu ja toteutus
Toimeksiantaja	-
Vuosi	2025
Sivut	29 sivua, liitteitä 8 sivua
Työn ohjaaja(t)	Teemu Manninen

TIIVISTELMÄ

Tämän opinnäytetyön tarkoituksena oli suunnitella ja rakentaa frisbeegolf-kiekkoon kiinnitettävä mittalaite ja mobiilisovellus sen käyttöä varten. Mittalaite olisi äärimmäisen hyödyllinen, sillä se parantaa ja nopeuttaa frisbeegolfin harjoittelua. Projektin inspiraatio oli lähtöisin kaupallisesta versiosta.

Laitteen suunnittelu aloitettiin valitsemalla ESP32:n toimintaan tarvittavat komponentit sekä mittaukseen tarvitut anturit. Koekytkennän onnistuttua komponentit integroitiin piirilevylle juottamalla ja liimaamalla. Lopuksi piirilevylle rakennettiin tukijalat ja suojakuori muovista, jotka liimattiin kiinni frisbeegolfkiekkoon. Akku sijoitettiin piirilevyn alle. Ohjelmistoa kehitettiin koko rakennusprojektin ajan. Laitteen testaus ja kalibrointi ei kuulunut työhön, ja laitteelle suoritettiin ainoastaan pienimuotoinen testaus.

Lopputuloksena syntynyt mittalaite ja mobiilisovellus saatiin toimimaan. Mobiilisovellus saavutti tavoitteet, sekä kommunikaatio laitteen ja sovelluksen välillä toimi hyvin. Kuitenkin laitteen antamien tulosten luotettavuus jäi hyvin alhaiselle tasolle, sekä laitteen käytännön toteutus oli epäammattimainen.

Laitetta on mahdollista käyttää harjoitteluun, mutta vertailukelpoista dataa se ei anna. Jatkokehitystä voisi tehdä erityisesti laitteen ohjelmakoodin osalta. Kehitystä voisi tehdä myös tukirangan, piirilevyn ja ulkokuoren osalta.

Asiasanat: mittauslaitteet, mikro-ohjaimet, ohjelmointi

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Engineering
Author (authors)	Robert Juhakoski
Thesis title	Design and Implementation of a Disc Golf Measurement device
Commissioned by	-
Time	2025
Pages	29 pages, 8 pages of appendices
Supervisor	Teemu Manninen

ABSTRACT

The purpose of this thesis was to design and build a measurement device that is attached to a disc golf disc, along with a mobile application for its use. The measuring device would be extremely useful as it enhances and speeds up disc golf practice. The inspiration for this project came from a commercial version.

Designing the project began by selecting the necessary components for the ESP32's operation and the sensors required for measurement. After successful prototyping, the components were soldered onto a circuit board. Finally, support legs and a protective casing were built from plastic, and the battery was placed under the circuit board. The software was developed throughout the entire construction project. Testing and calibration of the device were not part of the work, and only a small-scale inspection was conducted.

As a result, the measurement device and mobile application functioned. The application achieved its goals, and communication between the device and the application worked well. However, the reliability of the measurement results remained very low, and the practical implementation of the device was unprofessional.

The device can be used for training, but it does not provide comparable data. Especially the software code requires further development. The external casing, circuit board and the outer shell could also be further developed.

Keywords: measuring devices, micro controllers, programming

SISÄLLYS

1 JOHDANTO

Frisbeegolf on nopeasti kasvava urheilulaji, joka houkuttelee yhä enemmän harrastajia ja ammattilaisia ympäri maailman. Frisbeegolfin väitellysti yksi koukuttavimpia hetkiä on katsoa, kun kiekko liitelee pitkään ilmassa heiton jälkeen. Aloittelijalle tämä hetki valitettavasti kestää lyhyempään kuin kehittyneelle pelaajalle, mikä kannustaa harjoittelemaan pidempiä heittoja. Kiekon lentopituus määräytyy toki heittosuunnan perusteella, mutta pituuteen vaikuttaa lisäksi 4 tekijää: kiekon lähtönopeus, pyörimisnopeus, lentokulma sekä huojunta.

Kaikkia näitä ominaisuuksia pystyy arvioimaan silmämääräisesti, jos antaa kiekon lentää tarpeeksi kauan. Tarkkoja tietoja kiekon lennosta on kuitenkin mahdotonta saada arvioimalla. Lisäksi heitettyjen kiekkojen hakeminen ja etsiminen hidastaa harjoittelua. Käyttämällä jonkin tyyppistä mittalaitetta pystytään kiekon lentoarvot määrittämään tarkasti sekä lyhyessä ajassa. Tämä mahdollistaa kiekon heittelyn lähellä olevaan verkkoon, mikä nopeuttaa harjoittelua dramaattisesti. Se mahdollistaa myös harjoittelun paikoissa, joissa ei normaalisti voisi heitellä frisbeetä.

Itsekin lajin harrastajana kiinnostuin markkinoille pompahtaneesta uudesta mittalaitteesta, joka mullisti lajin harjoittelun, Tech Disc:stä. Tech Disc on kiekkoon pysyvästi kiinnitettävä mittalaite, joka pystyy antamaan tarkkaa palautetta heitostasi. Tällä hetkellä mittalaitteesta pyydetään suhteellisen suurta 299 euron hintaa, mikä herätti mielenkiintoni. Kiinnostuin aiheesta ja halusin lähteä tutkimaan, onko mahdollista itse rakentaa vastaavan tasoinen laite.

2 TEOREETTINEN VIITEKEHYS

2.1 Mittalaitteet urheilussa

Urheiluun liittyvät mittalaitteet ovat kehittyneet merkittävästi viime vuosikymmeninä tarjoten tarkkaa dataa, joka auttaa sekä urheilijoita että valmentajia ymmärtämään suorituksia ja kehittämään taitoja. Esimerkiksi juoksijoiden käytössä olevat kiihtyvyysanturit mittaavat askelrytmiä ja nopeutta, kun taas jalkapallossa käytettävät GPS-seurantalaitteet analysoivat pelaajien liikkumista kentällä.

Baseballissa ja golfissa on perinteisesti käytetty nopeustutkaa heittojen ja lyöntien arviointiin, mutta pelkästään heittonopeuden arviointi ei kerro koko totuutta. Teknologioiden kehittyessä nopeustutkan rinnalla on alettu käyttämään korkeanopeuskameroita. Tutkan ja kameran hybriditeknologia ”optically enhanced radar tracking (OERT)” on tällä hetkellä tarkinta dataa antavaa mittaustapa [1]. OERT-teknologiaan perustuvat laitteet saattavat maksaa jopa kymmeniä tuhansia euroja.

Frisbeegolfissa on myös perinteisesti käytetty nopeustutkaa kiekon lähtönopeuden mittaamiseen. Lähtönopeuden määrittäminen ei kuitenkaan riitä itsessään, ja siitä syystä on kehitetty edistyneempää tekniikkaa. Baseballista ja golfista poiketen frisbeegolf-kiekkoon voidaan kiinnittää antureita, mitkä mittaavat kiekon lentoa. Tämä mahdollistaa erilaisten teknologioiden syntymisen, mistä Tech Disc on erinomainen esimerkki.

Frisbeegolfkiekkoon kiinnitettävä mittalaite ei saa olla liian painava tai kookas, ettei kiekon heittäminen muutu liian vaikeaksi. Mittalaitteella varustetulla kiekolla ei kuitenkaan tarvitse olla normaalia kiekkoa vastaavaa aerodynamiikkaa, koska anturi tekee mittaukset aivan lennon alussa. Tämä vapauttaa mittalaitteen suunnittelua.

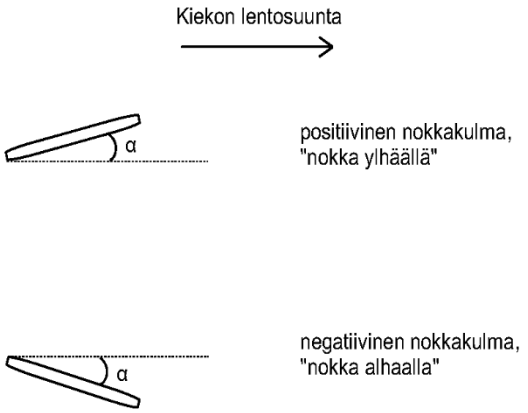
2.2 Frisbeegolfin lentoteoriaa

Frisbeegolfissa heiton onnistumista arvioidaan neljällä kriteerillä: lähtönopeus, pyörimisnopeus, lentokulmat sekä pyörimisen puhtaus. Kiekon lähtönopeus ja pyörimisnopeus määrittävät heiton tehokkuuden, ja lentokulmat sekä pyörimisen puhtaus vaikuttavat kiekon aerodynaamisiin ominaisuuksiin. Etenemis- ja pyörimisnopeus on helposti ymmärrettävissä, mutta selvitetään hieman, mitä lentokulmat ja pyörimisen puhtaus tarkoittavat.

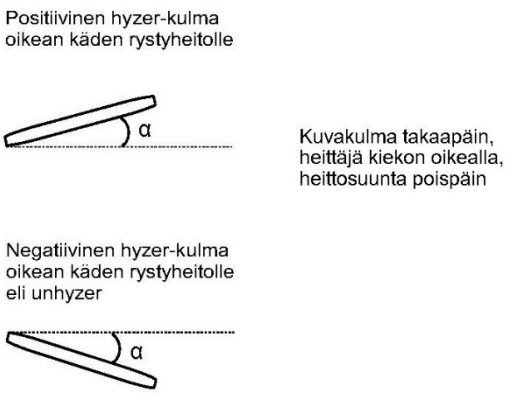
Kiekon lentäessä se muuttaa asentoaan, ja kun tarkastellaan lentokulmia, niin on kiinnostuttu lähtöhetkestä. Kiekon lentokulmat jaetaan kahteen osaan: nokkakulma sekä hyzer-kulma. Nokkakulma kuvaa kiekon etureunan suhdetta muuhun kiekkoon (kuva 1). Hyzer-kulma kuvaa kiekon sivuttaisuuntaista kallistuskulmaa. Hyzer-kulman nimitys on riippuvainen pyörimissuunnasta,

joten tarkastellaan tässä työssä ainoastaan oikean käden rystyheitosta aiheutuvaa pyörimissuuntaa (kuva 2).

Pyörimisen puhtaus tarkoittaa, että kiekko ei pyöri puhtaasti oman akselinsa ympäri vaan pyörimisessä on ylimääräistä heiluntaa. Tämä johtuu huonosta heittotekniikasta heikentäen kiekon aerodynamiikkaa, mikä taas vaikuttaa negatiivisesti heittopituuteen ja tarkkuuteen.



Kuva 2. Nokkakulma havainnollistettuna



Kuva 1. Hyzer-kulma havainnollistettuna

2.3 ESP32-mikrokontrollerisirun ominaisuudet

ESP32 on tehokas ja monipuolinen mikrokontrolleri, joka on suunniteltu erityisesti IoT-sovelluksiin. Sen keskeisiä ominaisuuksia ovat kaksi 32-bittistä Xtensa LX6 -prosessoria, jotka mahdollistavat moniajojen suorittamisen. ESP32 sisältää integroidut Wi-Fi- ja Bluetooth-moduulit, jotka tarjoavat nopeat ja vakaat langattomat yhteydet erilaisiin sovelluksiin, kuten anturidatan siirtoon tai etäohjaukseen. Siru tukee laajaa valikoimaa liitäntöjä, kuten GPIO, I2C, SPI ja UART, mikä tekee siitä helposti integroitavan erilaisten laitteistojen

kanssa. Lisäksi ESP32:ssa on alhainen virrankulutus ja useita virransäästötiloja, jotka tekevät siitä erinomaisen valinnan akkukäyttöisiin projekteihin. ESP32 toimii optimaalisesti 3,3 voltin käyttöjännitteellä, mutta se sietää jännitteitä välillä 3,0–3,6 volttia. [2.]

Wi-Fi-protokolla on keskeinen osa ESP32:n toiminnallisuutta, sillä se mahdollistaa nopean ja luotettavan langattoman yhteyden IoT-sovelluksissa. ESP32 tukee IEEE 802.11 b/g/n -standardeja, mikä tarkoittaa, että se toimii sekä 2,4 GHz:n taajuudella että useissa erilaisissa Wi-Fi-verkkoympäristöissä. Se tarjoaa erilaisia tiloja, kuten Station Mode (STA), Access Point Mode (AP) ja Station + Access Point Mode, jotka mahdollistavat joustavan yhteydenhallinnan. Näin ESP32 voi toimia joko verkkoon liitettynä asiakaslaitteena, langattomana tukiasemana tai molempina samanaikaisesti. [3.]

2.4 Bluetooth-tekniikka

Bluetooth-tekniikka on langattoman tiedonsiirron muoto, jossa kaksi laitetta kommunikoi käyttäen radioaaltoja. Käytetty taajuusalue on ISM (Industrial, Scientific, Medical), eli 2,402–2,480 GHz. Bluetoothista on olemassa kaksi versiota: Bluetooth Classic ja Bluetooth Low Energy. Bluetooth Low Energy on tarkoitettu nimensä mukaisesti sovelluksiin, jotka tarvitsevat pienen virrankulutuksen. [4.]

Bluetooth Classic -tiedonsiirrossa 2,402–2,480 GHz taajuusväli on jaettu 79 kanavaan. Jokaisella kanavalla on kaksi taajuutta, joista toinen edustaa nollaa ja toinen ykköstä. Vaihtamalla nopeasti kahden eri taajuuden välillä voidaan lähettää tietoa binäärikoodina. Taajuuden vaihtaminen tapahtuu erittäin nopeasti, sillä Bluetoothin tiedonsiirtonopeus on 1 megabitti sekunnissa (Mb/s), mikä tarkoittaa, että taajuus vaihtuu miljoona kertaa sekunnissa. [4; 5.]

Laitteiden välillä siirtyvä binäärikoodi on jaettu paketteihin. Jokainen paketti on koodattu niin ettei väärä laite voi vastaanottaa viestiä. Tämä mahdollistaa usean eri laitteen kommunikoinnin samalla kanavalla yhtä aikaa. [5.]

Lisäksi Bluetooth-tiedonsiirrossa on käytössä protokolla nimeltä Frequency-Hopping Spread Spectrum (FHSS). FHSS tarkoittaa, että tiedonsiirtoon käytettyä kanavaa vaihdetaan jatkuvasti. Ainoastaan laiteparin molemmat osapuolet tietävät, mille kanavalle vaihdetaan seuraavaksi, mikä parantaa tietoturvaa. FHSS-protokollassa kanavaa vaihdetaan 1 600 kertaa sekunnissa. [5.]

2.5 Projektissa käytetyt komponentit

Tässä projektissa käytettiin NodeMCU ESP32 -kehitysalustaa eli valmiiksi rakennettua piirilevyä, missä on tarvittu ESP32-mikrokontrollerisiru integroituna piirilevyyn (kuva 3). Kehitysalustassa on kiinni ESP32:n lisäksi toinen mikrosiru, minkä avulla mikrokontrolleri kommunikoi USB-väylää pitkin. Kehitysalustassa on jänniteregulaattori, joka muuttaa USB:n 5 voltin 3,3 volttiin. Lisäksi kehitysalustan kiinnittäminen laitteistoon on helpompaa verrattuna paljaaseen ESP32-piiriin. [6.]

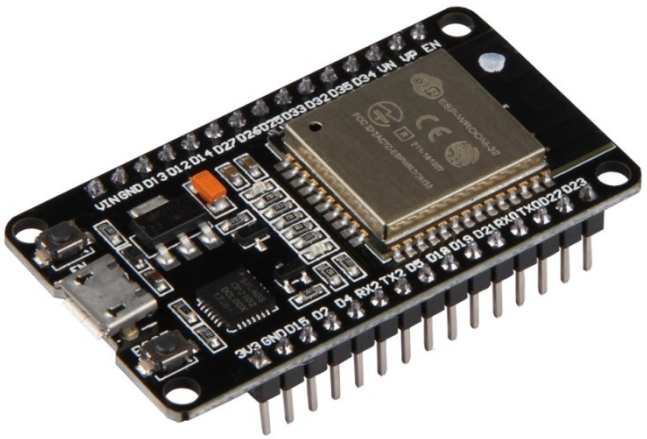
Projektissa käytetyt anturit ovat MPU-6050 ja H3LIS200DL. MPU-6050-anturi yhdistää 3-akselisen kiihtyvyysanturin ja 3-akselisen gyroskoopin yhteen kompaktiin moduuliin [5]. H3LIS200DL on 3-akselinen kiihtyvyysanturi, joka on suunniteltu mittaamaan korkeita g-voimia [8].

Mittalaitteen akuksi valittiin 3,7 voltin nimellisjännitteellä toimiva LiPo-akku 550 milliampeeritunnin kapasiteetilla, jonka koko on 5x30x40 mm. Akun antama todellinen jännite täydessä latauksessa on noin 4,2 volttia, ja jännite laskee noin 3,0 volttiin latauksen lähestyessä tyhjää [9].

NodeMCU ESP32 -kehitysalustan komponentit mukaan lukien ESP32-siru vaativat tasaisen 3,3 voltin jännitteen. Akun muuttuvan jännitteen vuoksi laitteeseen valittiin myös lineaarinen jänniteregulaattori MCP1700-3302ETO. Regulaattorin antama maksimivirta on 250 milliampeeria ja maksimi pudotusjännite on 350 millivolttia [10].

Lisäksi LiPo-akut tarvitsevat akun suojapiirin turvalliseen käyttöön ja lataamiseen [11]. Tähän laitteeseen valittiin LiPo-akuille tarkoitettu suojapiiri TP4056. TP4056-suojapiiri itsessään on pieni ja hankalasti juotettava pintaliitoskompo-

nentti. Tästä syystä projektia varten hankittiin HW-373-sovituslevy. Sovituslevy on piirikortti, joka on tehty helpottamaan pienten komponenttien käsittelyä [13, s. 90]. HW-373 sisältää pääasiassa mikro-USB-liitännän, latauksen tilasta kertovat ledit, TP-4056 suojapiirin, latausnopeuden säätövastuksen sekä ylipurkaussuojan. TP4056-piirissä on valmiiksi aseteltuna 4,2 voltin latausjännite, mikä tekee siitä turvallisen ja sopivan vaihtoehdon 3,7 voltin nimellisjännitteen LiPo-akun lataamiseen [12].



Kuva 3. NodeMCU ESP32 -kehitysalusta

2.6 Yhtälöt

Tässä kappaleessa on listattuna työn suunnitteluun ja testaukseen käytetyt yhtälöt.

Keskeiskiihtyvyys voidaan määrittää yhtälöstä 1 [14, s 92].

$$a_r = \omega^2 r \tag{1}$$

jossa	a_r	keskeiskiihtyvyys	[m/s^2]
	ω	kulmanopeus	[rad/s]

Kulmanopeus voidaan määrittää yhtälöstä 2 [14, s 92].

$$\omega = 2\pi n \quad (2)$$

jossa	ω	kulmanopeus	[rad/s]
	n	kierrostaajuus	[kierrosta/s]

Nopeus tasaisesti kiihtyvässä liikkeessä voidaan määrittää yhtälöstä 3 [14, s 91].

$$v = v_0 + at \quad (3)$$

jossa	v	nopeus hetkellä t	[m/s]
	v_0	hetkellä t=0	[m/s]
	a	(vakio)kiihtyvyys	[m/s^2]

3 FRISBEEGOLF-KIEKON MITTALAITTEEN SUUNNITTELU JA TOTEUTUS

Tässä kappaleessa käydään ensin läpi komponenttien valintaan liittyviä perusteluja ja tarkasteluja. Kappaleen edetessä selvitetään, millaisia haasteita käytännön toteutuksessa ilmeni ja mihin ratkaisuihin lopulta päädyttiin. Lisäksi tarkastellaan mobiilisovelluksen ja laitteiston ohjelmakoodia pääpiirteittäin.

3.1 Laitteiston suunnittelu

Laitteiston suunnittelun perustana oli selvittää NodeMCU ESP32 -kehitysalustan toimintaan vaadittavat komponentit ja rakentaa niistä toimiva kokonaisuus. Suunnittelun toisena päätavoitteena oli säilyttää kompakti koko.

3.1.1 Mikrokontrolleri

Suunnittelu aloitettiin mikrokontrollerista. Oli selvää, että tarvittaisiin mikrokontrolleri laskentatehoksi laitteeseen. Alkuperäinen suunnitelma oli käyttää Arduino Nanoa, mutta lopulta päädyttiin kuitenkin ESP32-mikrokontrolleriin, koska siinä on sisäänrakennettuna bluetooth sekä Wi-Fi, eikä näin ollen tarvita

lisämoduuleja. ESP32-mikrokontrolleri on itsessään pintaliitoskomponentti, joten projektiin hankittiin NodeMCU ESP32 kehitysalusta, mihin kyseinen mikrokontrolleri ja sen kätevään käyttöön tarvittavat oheiskomponentit ovat valmiiksi integroituna.

3.1.2 Anturit

Mikrokontrolleri itsessään ei tee muuta kuin prosessoi ja lähettää tietoa, joten tarvittiin projektiin myös anturi. Projektiin valittiin MPU-6050 kiihtyvyysanturi sen halvan hinnan ja saatavuuden takia. Anturin valmistaja ilmoittaa sen 3-akselisen gyroskoopin pystyvän mittaamaan tarkasti maksimissaan 2000°/s pyörimisnopeuteen asti. MPU-6050 sisältää myös 3-akselisen kiihtyvyysanturin, joka pystyy mittaamaan maksimissaan 16 g:n kiihtyvyyksiä luotettavasti. Seuraavana tehty matemaattinen tarkastelu MPU-6050-anturin soveltuvuudesta projektiin.

Ensiksi tutkittiin pyörimisnopeuden mittaamista. Tech Discillä yksi korkeimmista ikinä mitatuista kiekon pyörimisnopeuksista on 1705 kierrosta minuutissa [15]. Käytetään tätä arvoa anturin tarkasteluun. Kun se muutetaan asteiksi sekunneissa, niin voi tehdä vertauksen anturivalmistajan antamaan tietoon.

$$1705\ kierrosta/min * (\frac{360°}{60s}) = 10230°/s$$

$$10230°(/s)/2000°(/s) = 5{,}115$$

Kuten huomataan, niin MPU-6050:n gyroskoopin mittauskyvyn raja ylittyy yli viisinkertaisesti. Todennäköisesti MPU6050:n gyroskooppi ei pystyisi mittaamaan edes aloittelijan tuottamia pyörimisnopeuksia tarkasti.

Vastaavasti tarkasteltiin MPU-6050:n kiihtyvyysanturin kokemaa keskeiskiihtyvyyttä kiekon pyöriessä tasaisella nopeudella. Tarkasteluun käytettiin keskeiskiihtyvyyden kaavaa (1), jonka käyttöä varten määritettiin ensin kulmanopeus kaavalla (2). Tarkastelussa oletettiin, että anturi on sijoitettu 1 cm etäisyydelle kiekon keskipisteestä.

$\omega = 2\pi n = 2 * \pi * (\frac{1705\ kierrosta/min}{60s})$ =178,54… rad/s (2)

$a_r = \omega^2 * r = \left(178{,}54 \ldots \frac{\mathrm{rad}}{\mathrm{s}}\right)^2 * 0{,}01\mathrm{m} = 318{,}79 \ldots \mathrm{m}/s^2$ (1)

Kiihtyvyys muutettiin g-voimiksi

$318{,}79 \ldots (\mathrm{m}/s\text{^}2\ )/(9{,}81\ \mathrm{m}/s\text{^}2\ ) = 32{,}49 \ldots g$

Huomataan, ettei MPU-6050 pysty tuottamaan tarkkaa dataa näin kovassa pyörimisnopeuksissa 1 cm etäisyydellä keskipisteestä.

Seuraavaksi laskettiin etäisyys r, jolla anturi saisi enintään olla kiekon keskipisteestä, kun ehtoina on 1705 kierroksen minuuttinopeus ilman 16 g:n ylittämistä. Laskussa käytettiin aiemmin saamaamme kulmanopeutta 178,54… rad/s.

$a_r = \omega^2 * r => r = \frac{a_r}{\omega^2} = \frac{16 * \frac{9{,}81m}{s^2}}{(178{,}54 \ldots \frac{rad}{s})^2} = 0,004923 \ldots m \approx 4{,}92mm$ (1)

Tarkastelun perusteella voidaan todeta MPU-6050 anturin pystyvän teoriassa mittaamaan jopa korkeimpia frisbeegolfissa havaittavia pyörimisnopeuksia tarkasti. Ottaen huomioon, että mikrosirun koko on 4x4x0,9 mm, niin fyysisesti on mahdollista asetella anturi 4,92 mm rajan sisäpuolelle.

Seuraavaksi tarkastelussa pohdittiin lähtönopeuden mittaamista. Lähtönopeus on teoriassa mahdollista laskea käyttäen MPU6050-anturin tuottamaa kiihtyvyysdataa, mutta ongelmaksi muodostuu anturin kyky sietää g-voimia. YouTube-sivustolla JoeV Disc Golf-kanavalla tehdyssä tarkastelussa arvioidaan kiekon kokevan 150 g:n kiihtyvyyttä suurimmillaan heiton aikana, kun heiton lähtönopeus on arviolta 67 mailia tunnissa [16].

Tarkastelun jälkeen projektiin hankittiin myös H3LIS200DL-kiihtyvyysanturi. Kyseinen anturi pystyy valmistajan mukaan mittaamaan jopa 200 g:n kiihtyvyyttä ilman saturaatiota. H3LIS200DL-kiihtyvyysanturi on kuitenkin suhteellisen epätarkka. Anturin tarkkuudeksi ilmoitetaan 200 g:n mittausalueella 1560 mg/digit [8]. Tämä tarkoittaa, että hetkellinen kiihtyvyyden arvo voi heittää jopa 1,56*9,81 m/s^2 = 15,30 m/s^2 todellisesta kiihtyvyyden arvosta. Epätarkkuutta

pyrittiin korjaamaan ohjelmallisesti laitteiston koodissa, sekä epätarkkuus otettiin huomioon tulosten arvioinnissa.

3.1.3 Virtalähde

Mittalaitteen virtalähteen valinnan perustana oli komponenttien vaatima 3,3 voltin nimellisjännite. Anturit kestävät hieman laajempaakin jänniteskaalaa, mutta mikrokontrolleri ja sen kehitysalusta toimivat optimaalisesti 3,3 voltin jännitteellä [6; 7; 8]. Laitteiston painon ollessa määrittävä tekijä valittiin pienin saatavilla ollut 3,7 voltin akku eli 550:n milliampeeritunnin LiPo-akku.

3.1.4 Jänniteregulaattori

Kuvassa 4 on havainnollistettuna 3,7 voltin nimellisjännitteisen LiPo-akun jännitteen muuttuminen suhteessa latauksen määrään. Jotta saataisiin muutettua jännite tasaiseksi, tarvittiin jänniteregulaattori. Jänniteregulaattori ottaa tasavirtaa tietyllä jännitevälillä, ja muuttaa jännitteen halutuksi. Lineaarinen jänniteregulaattori pudottaa aina jonkin verran saamaansa jännitettä, ja tätä kutsutaan pudotusjännitteeksi [13, s. 242].

Pudotusjännite koituu ongelmaksi, koska esimerkiksi 0,5 voltin pudotusjännitteellä varustettu regulaattori vaatii 3,8 volttia toimittaakseen laitteistolle vaaditun 3,3 volttia. Kuvasta 4 nähdään, että akku tuottaa 3,8 volttia tai enemmän ainoastaan ollessaan noin 40 % latauksessa tai enemmän. Kuvasta 4 huomataan myös, että akun tuottama jännite pysyy suurimman osan latauksestaan

yli 3,7 voltissa. Tämä tarkoittaa, että optimaalinen pudotusjännite jänniteregulaattorille saisi olla maksimissaan noin 0,4 volttia, kun tarkoituksena on tuottaa laitteistolle 3,3 voltin käyttöjännite.

Lipo Voltage Chart						
Voltage	1S	2S	3S	4S	5S	6S
Voltage	3.7V	7.4V	11.1V	14.8V	18.5V	22.2V
Fully Charged Voltage	4.2V	8.4V	12.6V	16.8V	21V	25.2V
Relationship of Voltage and Capacity						
Capacity %	1S	2S	3S	4S	5S	6S
100	4.2	8.4	12.6	16.8	21	25.2
95	4.15	8.3	12.45	16.6	20.75	24.9
90	4.11	8.22	12.33	16.45	20.56	24.67
85	4.08	8.16	12.25	16.33	20.41	24.49
80	4.02	8.05	12.07	16.09	20.11	24.14
75	3.98	7.97	11.95	15.93	19.92	23.9
70	3.95	7.91	11.86	15.81	19.77	23.72
65	3.91	7.83	11.74	15.66	19.57	23.48
60	3.87	7.75	11.62	15.5	19.37	23.25
55	3.85	7.71	11.56	15.42	19.27	23.13
50	3.84	7.67	11.51	15.34	19.18	23.01
45	3.82	7.63	11.45	15.26	19.08	22.89
40	3.8	7.59	11.39	15.18	18.98	22.77
35	3.79	7.57	11.36	15.14	18.93	22.72
30	3.77	7.53	11.3	15.06	18.83	22.6
25	3.75	7.49	11.24	14.99	18.73	22.48
20	3.73	7.45	11.18	14.91	18.63	22.36
15	3.71	7.41	11.12	14.83	18.54	22.24
10	3.69	7.37	11.06	14.75	18.44	22.12
5	3.61	7.22	10.83	14.43	18.04	21.65
0	3.27	6.55	9.82	13.09	16.37	19.64

Kuva 4. LiPo-akun lähtöjännite latauksen muuttuessa [9]

Ongelmana jänniteregulaattorin valinnassa tulee vastaan niiden kyky syöttää virtaa. Pudotusjännitteen pienentyessä myös komponentin kyky syöttää virtaa pienenee. Johtuen mikrokontrollerien vaikeasti arvioitavasta virrankäytöstä hankittiin varmuuden vuoksi 2 jänniteregulaattoria. MCP1700-3303ETO oli pääasiallinen valinta jänniteregulaattoriksi sen pienen, 0,35 voltin pudotusjännitteen takia. Kyseisen regulaattorin 250 mA virrankestoisuuden riittävyys aiheutti epäilyksiä [10], joten hankittiin varalle myös toinen regulaattori, LM2937ET-3,3:n.

LM2937ET-3,3 olisi toiminnaltaan varma valinta, sillä mahdolliset virtapiikit, mitkä estäisivät MCP1700-regulaattorin käytön, eivät olisi ongelma tälle regulaattorille. LM2937ET-3,3:n käyttö projektissa kuitenkin pienentäisi akun käyttöaikaa merkittävästi johtuen sen 0,5 voltin maksimi pudotusjännitteestä. Kyseinen regulaattori kestää 500 mA virtaa. [17.]

3.1.5 Ulkokuori ja kiinnitys

Ulkokuori suunniteltiin tehtävän muovista käyttäen 3D-tulostinta. Ulkokuoren tarkkaa suunnittelua ei kuitenkaan lähdetty tekemään, koska ensin täytyisi tietää laitteiston lopullinen koko. Koekytkennän onnistumisen jälkeen tarkoituksena oli tilata valmis piirikortti, mihin komponentit saisi kiinnittyä tiiviisti. Tämän jälkeen ulkokuoren tarkemman suunnittelun ja toteutuksen voisi aloittaa. Todennäköisesti liimaaminen olisi paras vaihtoehto mittalaitteen kiinnittämiselle kiekkoon.

3.2 Laitteiston toteuttaminen

3.2.1 Kytkentä

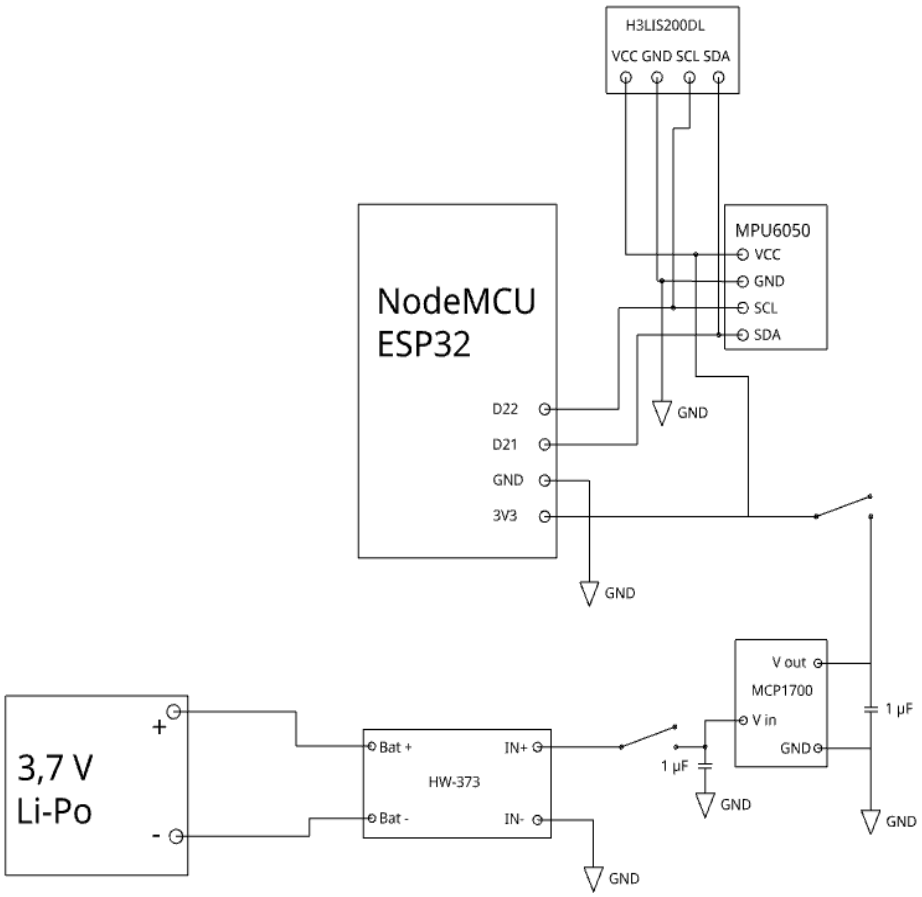
Ensimmäisenä tehtiin koekytkentä leipälevylle, eli alustalle, johon on helppo kiinnittää komponentteja väliaikaisesti. Kuvassa 5 on havainnollistettuna käytetty kytkentä. Kaikki komponentit jakavat yhteisen maatason, ja jänniteregulaattorin tuottama 3,3 voltin käyttöjännite on kytketty NodeMCU ESP32-kehitysalustan 3v3-pinniin sekä MPU6050- ja H3LIS200DL-kiihtyvyysantureiden VCC-pinneihin [6; 7; 8].

Anturit ja mikrokontrolleri kommunikoivat I2C-väylää pitkin, eli ne yhdistetään kahdella johdolla SCL ja SDA. SCL-liitinten välillä kulkee kellosignaali ja SDA-liitinten välillä datasignaali [13, s. 94]. Laitteiston kiihtyvyysantureissa SDA- ja SCL-pinnit on nimetty vastaamaan näitä nimiä, ja mikrokontrollerin SDA- ja SCL-pinnit ovat D21 sekä D22 [6]. Molemmat anturit ovat liitettyinä omilla johdoillaan samaan väylään, ja ohjelma käyttää eri osoitteita tiedonsiirtoon.

Virtalähde on kytketty HW-373:n BAT (-) ja BAT (+) -pinneihin, ja HW-373:n IN (+) - ja IN (-) -pinneistä positiivinen on kytketty syöttämään jänniteregulaattorin

syöttöpuolta, ja negatiivinen on kytketty yleiseen maahan. [18.] Jänniteregulaattorin tulopuolelle on kytketty virtalähde, ja lähtöpuoli on kytketty syöttämään laitteiston käyttöjännitettä. Regulaattorin maadoituspinni on kytketty yleiseen maahan, sekä syöttö ja lähtöpuolet on kytketty 1 mikrofaradin kondensaattoreiden läpi maahan virtapiikkien kontrolloimiseksi [10].

Lisäksi kytkennässä on kaksi fyysistä on/off-kytkintä. Ensimmäinen on laitteen virtakytkin, joka sijaitsee akun ja HW-373-piirin välissä. Virrankatkaisun lisäksi kytkimen tehtävä on eristää akku muusta laitteistosta latauksen ajaksi. Toinen on/off-kytkin on tarkoitettu laitteen ohjelmointia varten, ja se on sijoitettu eristämään MCP1700-regulaattori NodeMCU ESP32 -piiristä. Ohjelmointia varten NodeMCU ESP32-piiriin on kytkettävä USB-johto, mikä aiheuttaisi takavirran muodostumisen MCP1700-regulaattorin ulostuloon mahdollisesti vahingoittaen komponenttia.



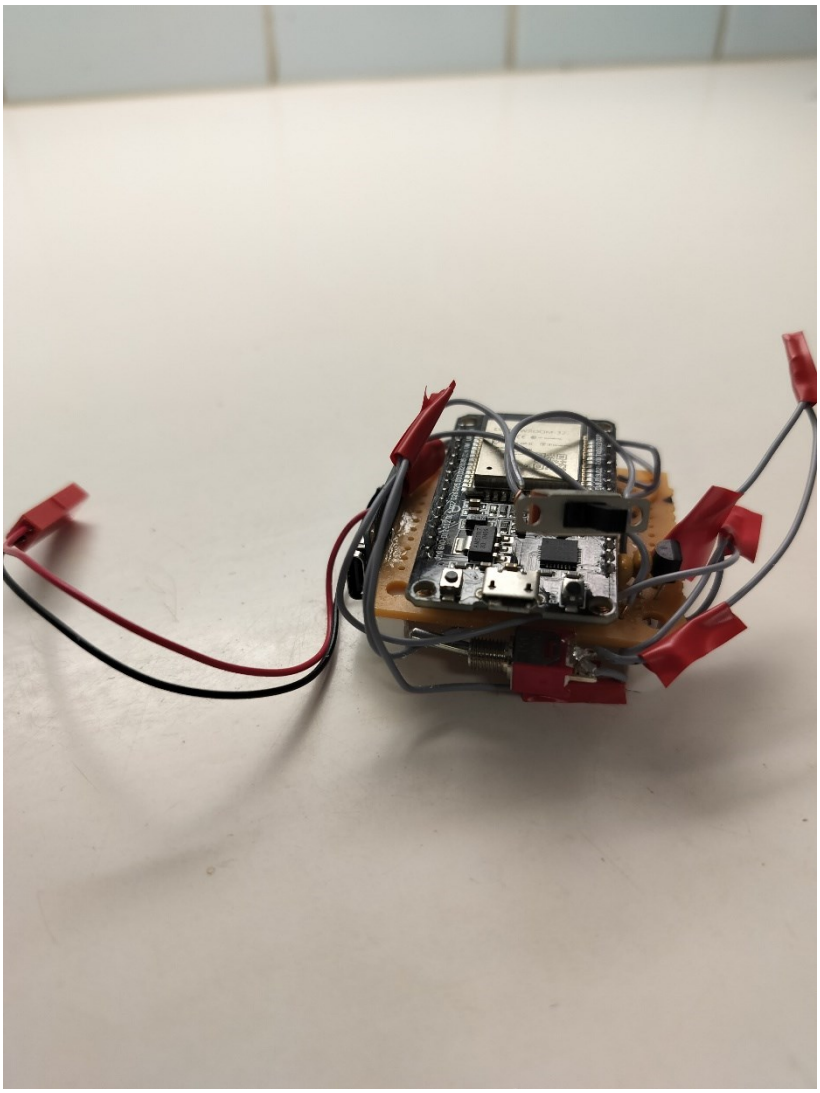
Kuva 5. Mittalaitteen kytkentäkaavio

3.2.2 Integrointi piirilevylle

Koekytkennän onnistuttua oltiin valmiita juottamaan komponentit piirilevylle. Projektissa käytettiin juotostäplillä varustettua tyhjää piirilevyä, johon kaikki komponentit kiinnitettiin.

Pienten komponenttien juottaminen osoittautui erittäin hankalaksi. Aluksi yritettiin tehdä tarpeelliset komponenttien liitokset juotostinaa käyttäen piirilevyn pohjaan. Pitkän juotostyön jälkeen laitetta testattaessa anturit eivät kuitenkaan lähettäneet dataa, vaikka komponenttien ledit syttyivät palamaan. Vianmääritysprosessin aikana laitteisto jouduttiin purkamaan useaan kertaan osiin, ja lopulta H3LIS200DL-kiihtyvyysanturi rikkoontui prosessissa. Liika juotosten purkaminen sulatti piirilevyn käyttökelvottomaksi. Noin 30 euron komponentille ei tilattu korvaavaa anturia, vaan projekti tehtiin loppuun käyttäen vain toista kiihtyvyysanturia.

Lopulta pitkän vianmääritysprosessin jälkeen jäljellä oleva anturi saatiin kommunikoimaan mikrokontrollerin kanssa. Toimivassa versiossa juotostinalla piirilevyn pohjaan tehdyt liitokset korvattiin johtimilla. Johtimien yhdistäminen tehtiin niin, että paljaat päät kierrettiin 4–6 kertaa toistensa ympäri ja lopuksi juotettiin yhteen. Paljaiden liitosten päälle lisättiin sähköteippiä. Lopullinen laitteisto on havainnollistettuna kuvassa 6.



Kuva 6. Mittalaite ilman suojakuorta

3.2.3 Tukirakenne ja kiinnitys

Piirilevyn kiinnitys hoidettiin lopulta leikkaamalla muovisista seinäkiinnikkeistä sopivat palat, jotka liimattiin piirilevyn reunoihin. Akku kiinnitettiin kangasteipillä kiekon pohjaan, ja piirilevy aseteltiin akun päälle siten, että piirilevyn ja akun väliin jäi hieman ilmaa. Ulkokuori tehtiin noin 1 mm paksusta, lieriön muotoisesta muovipakkauksesta. Ulkokuoreen tehtiin aukko laitteen lataamista ja käyttöä varten. Lopullinen laitteisto kiekkoon kiinnitettynä on kuvassa 7.

Kuva 7. Valmis mittalaite kiinnitettynä kiekkoon

3.2.4 Ohjelmakoodi

Koodin toiminnallisuus on selitetty tässä kappaleessa, ja ohjelmakoodi kokonaisuudessaan on nähtävissä liitteestä 1. Koodi on tuotettu Arduino IDE -ohjelmointiympäristössä käyttäen Arduinon omaa hieman helpotettua C++ -koodikieltä [13, s. 77]. Ohjelmakoodi perustuu kiihtyvyysanturin antamaan dataan.

Ohjelma tunnistaa heiton alkaneeksi, kun kiihtyvyys nousee tarpeeksi suureksi. Ensiksi ohjelma määrittää kiekon lähtönopeuden perustuen kiekon ko-

kemaan kiihdytykseen heittäjän kädestä. Kiekon vapauduttua heittäjän kädestä ohjelma määrittää pyörimisnopeuden perustuen kiekon kokemaan keskeiskiihtyvyyteen. Lyhyen ilmalennon aikana ohjelma määrittää myös kiekon lentokulmat sekä pyörimisen puhtauden määrän asteina. Lopuksi ohjelma lähettää laskemansa tiedot mobiilisovellukseen.

Koodissa käytettiin valmista MPU6050-anturille rakennettua koodia, jonka on julkaissut S. James Remington -niminen käyttäjä github.com-sivustolla [19]. MPU-6050-Fusion-tiedoston MPU6050_Mahony_calgyro.ino-niminen koodi sisältää Mahony-suodattimen, joka on räätälöity MPU6050-kiihtyvyysanturille. Mahony-suodatin käyttää kvarternio-matematiikkaa objektin asennon määrittämiseen tarkasti [20]. Mahony-suodattimen lopputuotteena on laitteen sen hetkiset kallistumis-, kääntymis- ja nyökkäyskulmat, joiden perusteella ohjelma määrittää lentoarvoja kiekolle.

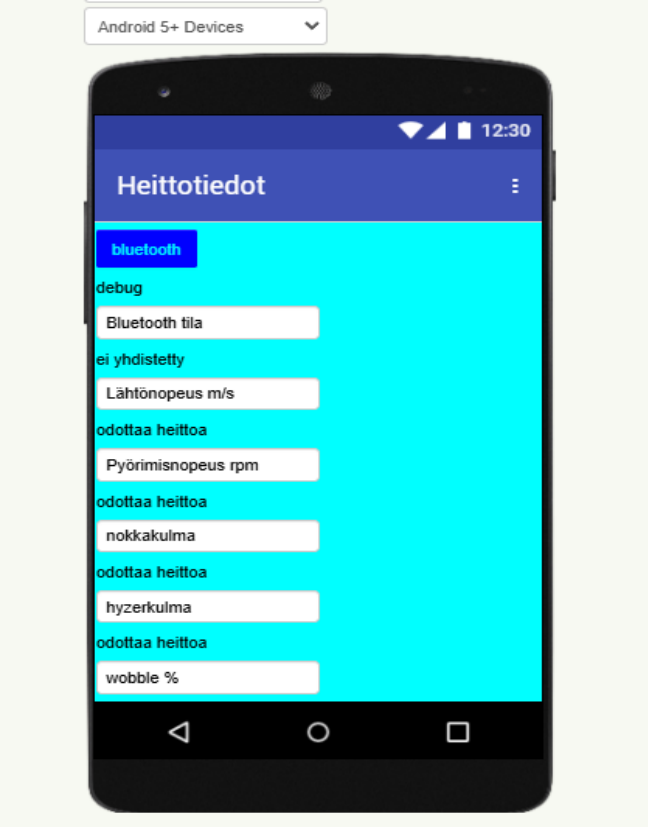
Kiekon lähtönopeus määritetään koodissa yhtälöllä 3. Kokonaiskiihtyvyyttä integroidaan 0,0025 s välein lähtönopeudeksi. Kiekon lennonaikainen nokkakulma sekä hyzer-kulma määritetään siltä hetkeltä, kun kiekko on juuri irronnut heittäjän kädestä, käyttäen mahony-suodattimen tarjoamaa dataa kallistuskulmista. Koodi on rakennettu sen oletuksen varaan, että kiekkoa pidetään heitettäessä aina samasta kohtaa. Pyörimisen puhtaus lasketaan z-akselin kokeman kiihtyvyyden heilahtelun perusteella, koska tasaisessa pyörimisessä z-akseli pysyy samassa tasossa.

3.2.5 Mobiilisovellus

Mobiilisovellus toteutettiin MIT App Inventorilla. MIT App Inventor on selainpohjainen vapaasti käytettävä visuaalinen ohjelmointiympäristö, joka on tarkoitettu mobiilisovellusten ohjelmointiin [21].

Ohjelmoitu sovellus on hyvin yksinkertainen, ja sen toiminnallisuuksiin ei lisätty ylimääräisiä ominaisuuksia. Ohjelmassa on yksi nappi, jota painamalla aukeaa lista puhelimeen liitetyistä bluetooth-laitepareista. Valitsemalla listasta jonkun laitteen ohjelma menee takaisin aloitusnäyttöön, ja bluetooth tilasta kertova teksti muuttuu ”ei yhdistetty” -tekstistä ”yhdistetty”-tekstiin, mikäli bluetooth-yhteys onnistui.

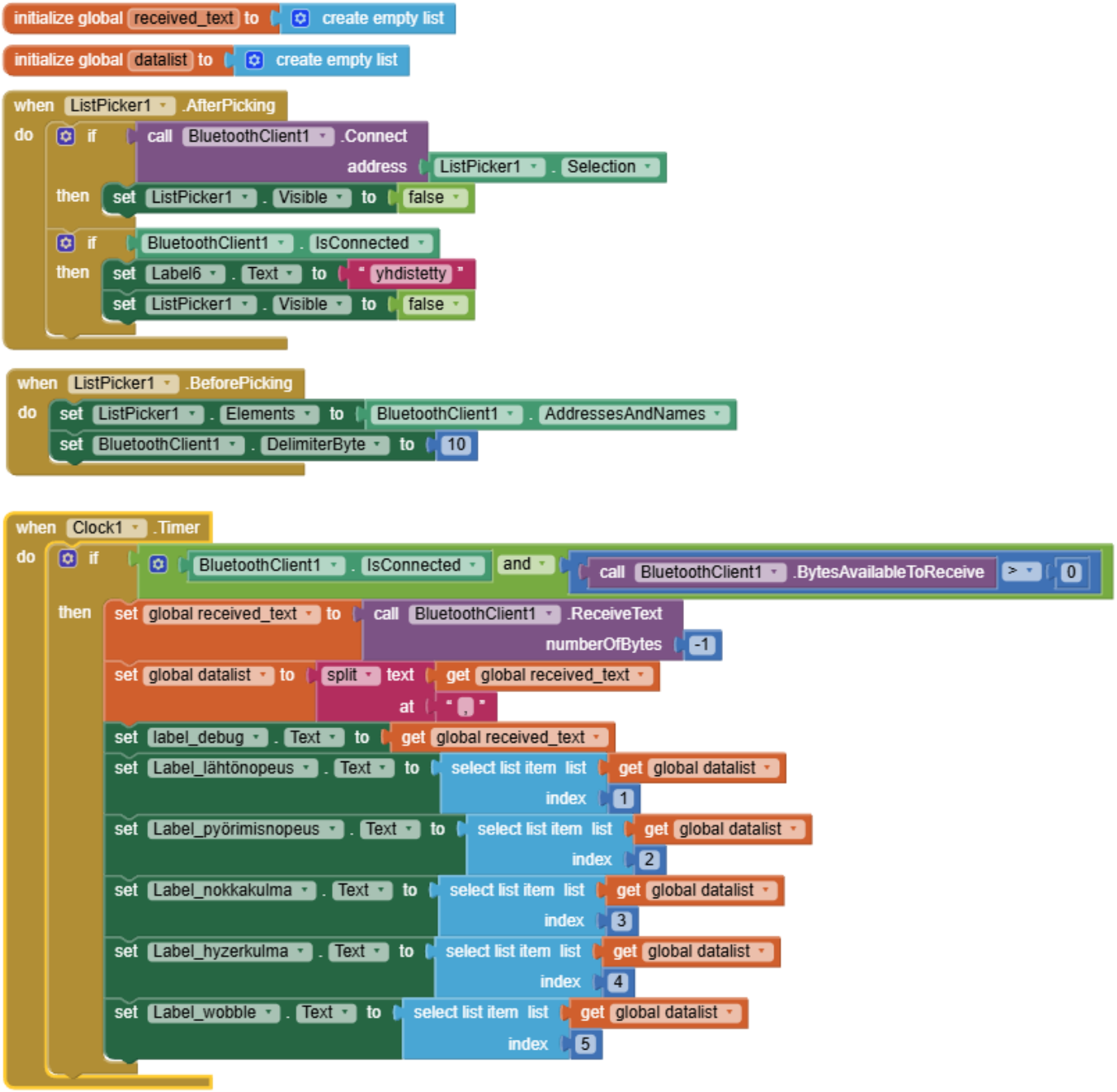
Ohjelmakoodi lähettää sovellukselle 5 tietoa pilkuilla erotettuina. Ohjelma purkaa viestin perustuen pilkkuihin, ja sitten asettelee tiedot näkyviin niille tarkoitettuihin paikkoihin. Kuvassa 8 on poiminta MIT App Inventorin simuloimasta mobiilinäkymästä, missä ”odottaa heittoa” -tekstien paikalle ilmaantuu heiton tiedot, kun sovellus vastaanottaa ne laitteistolta.



Kuva 8. MIT App inventorin simuloima mobiilinäkymä

MIT App Inventorin käyttö ei vaatinut aikaisempaa kokemusta koodaamisesta, sillä ohjelman luonnissa käytettiin visuaalista ohjelmointitapaa. Youtuben MoThunderz-nimisen käyttäjän tekemältä opasvideolta pystyttiin poimimaan

lähes kaikki ohjelmaan tarvittavat tiedot [22]. Kuvassa 9 on nähtävillä visuaalinen koodi MIT App Inventorin ohjelmointiympäristöstä poimittuna.



Kuva 9. MIT App Inventorin visuaalinen blokkeihin perustuva ohjelmointityyli

Koodissa on ensiksi määritelty kaksi muuttujaa ”received_text”-vastaanotetulle tiedolle ja ”datalist”-puretulle tietolistalle. Sen jälkeen on luotu ”list picker”-toiminnolla nappi, jolla avataan laiteparilista, ja otetaan yhteys laitteeseen. Viimeisenä on ”Clock” -toiminnon ympärille rakennettu logiikkafunktio, joka aktivoituu, kun mittalaite lähettää tietoa bluetoothilla. Funktio purkaa viestin osiin ja sitten tulostaa oikean viestin osan oikeaan paikkaan.

4 LAITTEISTON TESTAAMINEN

Laitteen testaaminen ja kalibrointi rajattiin työn ulkopuolelle, mutta perustoiminnan selvittämiseksi suoritettiin pienimuotoista testaamista koko projektin ajan. Tässä kappaleessa käydään läpi käytetyt testausmenetelmät ja saavutetut tulokset.

Testaus aloitettiin komponenttitasolla erilaisilla koekytkennöillä leipälevyllä. Ensin varmistettiin anturien ja mikrokontrollerien toiminta, minkä jälkeen tehtiin täysi koekytkentä kuvan 5 mukaisesti. Mittalaitteen kokoamisen jälkeen siirryttiin ohjelmiston testaukseen.

Ohjelmiston testaus aloitettiin asettamalla kiekko eri asentoihin ja seuraamalla kiihtyvyysdatan käyttäytymistä. Kehitystyön edetessä testit muuttuivat dynaamisemmiksi ja alkoivat sisältää kiekon heiluttelua. Lopulta kiekkoa heiteltiin asuinhuoneistoon rakennetussa improvisoidussa heittostudiossa. Studio suunniteltiin siten, että tiukasti viritetty lakana pysäytti kiekon, ja lattia pehmustettiin putoamiskohdasta.

Laitteiston toiminnan syvällisemmäksi ymmärtämiseksi testejä varten kehitettiin useita eri ohjelmistokoodeja. Merkittävin testausmenetelmä oli heiton aikaisten kiihtyvyysarvojen tallentaminen ohjelmallisesti. Näitä tietoja analysoimalla voitiin hienosäätää heiton ja ilmalennon tunnistukseen liittyviä kiihtyvyyden raja-arvoja.

Lopulliselle ohjelmistolle suoritettiin lyhyt testaus, jonka tavoitteena oli selvittää laitteiston antamien tietojen vaihtelua samanlaisten heittojen välillä. Ensin testattiin lähtönopeuden ja pyörimisnopeuden määritys. Kiekkoa heitettiin ensin viisi kertaa peräkkäin kevyellä voimalla ja sen jälkeen viisi kertaa kovalla voimalla. Taulukoissa 1 ja 2 on havainnollistettuna saadut testitulokset.

Lentokulmien ja pyörimisen puhtauden määrittäminen jäi niin huonolle tasolle, ettei niille suoritettu tarkempaa testaamista. Suurimpana ongelmana oli, että

muuttamalla hyzer- tai nokkakulmaa niin myös toinen muuttui samalla. Pyörimisen puhtauden määrittämiseen käytetty menetelmä ei tuottanut järkeviä tuloksia.

Taulukko 1. Lähtönopeuden testitulokset (m/s)

Heitto	1	2	3	4	5
kevyt heitto	37,6	37,3	36,7	45,0	37,2
kova heitto	47,5	46,0	50,4	47,7	54,8

Taulukko 2. Pyörimisnopeuden testitulokset (rpm)

Heitto	1	2	3	4	5
kevyt heitto	371,5	377,6	394,0	426,6	406,1
kova heitto	558,8	530,4	597,6	522,5	536,1

Taulukosta 1 huomataan, että lähtönopeuksissa on pientä heiluntaa, mutta laite tekee selvän eron kevyiden ja kovien heittojen välille. Lähtönopeuden määritys kovissa heitoissa ei kuitenkaan ole kovin luotettava johtuen korkeiden g-voimien mittauksen puutteesta. Taulukosta 2 nähdään, että pyörimisnopeuden määritys toimii systemaattisesti, mutta sisältää jonkin verran vaihtelua heittojen välillä. Vaikka testien luotettavuus ei olekaan täydellinen, voidaan silti tehdä johtopäätös, että laite toimii systemaattisesti, mutta kohtalaisen suurella heilunnalla.

5 POHDINTA JA JOHTOPÄÄTÖKSET

Opinnäytetyön tavoitteena oli suunnitella ja rakentaa mittalaite frisbeegolfiin, sekä ohjelmoida mobiilisovellus sen käyttöön. Mittalaitteen osalta tavoite osoittautui hyvin kunnianhimoiseksi. Mittalaitteen rakentamiseen liittyi useita käytännön haasteita, mitä ei osattu ottaa huomioon projektia aloittaessa.

Tuloksena syntynyt mittalaite ja sen käyttöön suunniteltu mobiilisovellus saatiin toimimaan. Komponentit olivat yhteensopivat, sekä kommunikointi mittalaitteen ja mobiilisovelluksen välillä onnistui kiitettävästi. Lopputuloksessa oli kuitenkin useita puutteita.

Suurimpana puutteena voidaan pitää ohjelmakoodin kyvyttömyyttä tuottaa tarkkoja lentoarvoja. Vaikka lentoarvojen määrittäminen onkin teoriassa mahdollista ainoastaan kiihtyvyysantureita käyttämällä, niin käytännössä se osoittautui erittäin haastavaksi. Kiekon jatkuva pyöriminen vaikeuttaa lentoarvojen määrittämistä dramaattisesti, ja lopulta projektin lentoarvoja määrittävän koodin osalta jouduttiin tekemään selkeitä kompromisseja.

Toinen suuri haaste ilmeni komponenttien integroimisessa piirilevylle. Pienien juotosten tekeminen käsin osoittautui erittäin hankalaksi, ja lopputuloksena projektin arvokkain yksittäinen komponentti vaurioitui käyttökelvottomaksi. Toisen kiihtyvyysanturin pois jättäminen heikensi entisestään mittalaitteen kykyä tuottaa luotettavaa tietoa, mutta laitteen luotettavuus olisi joka tapauksessa jäänyt suhteellisen alhaiselle tasolle.

Haasteet käytännön toteutuksessa johtivat siihen, että laitteiston liitokset jäivät jossain kohtaa puutteellisiksi. Gyroskooppia ei saatu toimimaan lopullisessa laitteistossa, mikä johtui mitä todennäköisimmin huonoista liitoksista. Lisäksi kovien heittojen välissä laite täytyi lähes joka kerta yhdistää uudestaan, mikä myös kertoi liitosten heikkoudesta.

Lopputuloksena syntynyt mittalaite oli ulkoiselta toteutukseltaan viimeistelemätön ja ohjelmistonsa osalta puutteellinen, eli jatkokehitykselle jäi runsaasti varaa. Laitteesta saisi siistimmän, kun ulkokuoren tekisi 3D-tulostimella ja yhdistäisi piirilevyn tukijalat ulkokuoreen. Laitteen toimintavarmuutta voisi lisätä ja kokoa pienentää huomattavasti, kun käytettäisiin sovituslevyttömiä komponentteja, ja juottaminen tehtäisiin koneellisesti. Piirilevynä tulisi myös tietenkin käyttää tehtaalta tilattua erikseen suunniteltua piirilevyä.

Jatkokehityksessä olisi myös hyödyllistä kokeilla magnetometrin lisäämistä kiihtyvyysantureiden rinnalle. Magnetometrillä voisi mahdollisesti määrittää etenemissuunnan, mikä helpottaisi lentokulmien määrittämistä. Lisäksi laitteeseen olisi hyvä rakentaa painonapilla toimiva virtapainike transistorista ja MOSFETistä sekä korvata takavirralta suojaava kytkin schottky-diodilla.

Ohjelmistojen osalta tärkein jatkokehityskohde olisi ehdottomasti lentokulmien ja pyörimisen puhtauden tarkempi määrittäminen. Tässä projektissa käytetty menetelmä nokka- ja hyzer-kulmien laskemiseen on epätarkka ja edellyttää, että kiekkoa pidetään aina samasta kohdasta. Muidenkin lentoarvojen määrittämistä voisi kehittää tarkemmaksi ja ohjelmaan voisi lisätä toiminnallisuuksia kuten heittojen tallentamisen ja vertailun.

Tuloksena syntynyt laite ei tuota vertailukelpoista tietoa, mutta sitä voidaan käyttää harjoittelussa heiton tehokkuuden arviointiin. Pyörimisnopeuden määritys toimii systemaattisesti, ja on odotettavissa, että se toimii hyvin myös todella voimakkaissa heitoissa. Lähtönopeuden määrittäminen toimisi muuten hyvin, mutta korkeiden g-voimien mittauksen puuttumisen vuoksi kovien heittojen mittaaminen on mahdotonta.

Frisbeegolf-mittalaitteen suunnittelun ja rakentamisen jälkeen kävi ilmi, että laitteen fyysinen toteutus on yhtä lailla haastavaa kuin ohjelmiston suunnittelu. Sähkötekninen suunnittelu ja koodin perusrakenteen kirjoittaminen tapahtui suhteellisen helposti, mutta lentoarvoja määrittävä koodin osuus ja käytännön toteutus koekytkennän jälkeen osoittautuivat todella haastaviksi.

Projektin useiden haasteitten takia päädyin oppimaan valtavan määrän uusia asioita ammattitaitoni kannalta. Ymmärrän pienelektroniikasta huomattavasti enemmän, sekä kiihtyvyysanturin ja gyroskoopin toiminta selkeni aivan uudella tavalla. Päädyin opettelemaan mikrokontrollerin ohjelmointia kymmeniä tunteja sekä opin ohjelmoimaan yksinkertaisia mobiilisovelluksia. Opinnäytetyöprosessin aikana kohtasin useasti ylitsepääsemättömältä tuntuvia esteitä, joista lopulta aina pääsi kuitenkin eteenpäin. Opin, ettei pidä luovuttaa, vaikka eteneminen tuntuu toivottomalta.

LÄHTEET

1. TrackMan. What is a golf launch monitor? Blogi. Päivitetty 26.9.2023. Saatavissa: https://www.trackman.com/blog/golf/what-is-a-golf-launch-monitor [viitattu 16.1.2025].

2. Espressif Systems. Tietolomake. WWW-dokumentti. 2023. Saatavissa: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf [viitattu 24.1.2025].

3. ESP32 A feature-rich MCU with integrated Wi-Fi and Bluetooth connectivity for a wide-range of applications. Espressif Systems. WWW-dokumentti. Päivitetty 2024. Saatavissa: https://www.espressif.com/en/products/socs/esp32 [viitattu 16.1.2025].

4. Learn About Bluetooth. Bluetooth SIG inc. WWW-dokumentti. Päivitetty 2025. Saatavissa: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/ [viitattu 14.3.2025].

5. Branch Education. How does Bluetooth Work? Youtube. Videoleike. Julkaistu 20.5.2021. Saatavissa: https://www.youtube.com/watch?v=1I1vxu5qIUM [viitattu 14.3.2025].

6. Joy-IT. Joy-IT. Tietolomake. WWW-dokumentti. Päivitetty 26.9.2018. Saatavissa: https://cdn-reichelt.de/documents/datenblatt/A300/SBC-NODEMCU-ESP32-DATASHEET_V1.2.pdf [viitattu 18.1.2025].

7. InvenSense Inc. Tietolomake. WWW-dokumentti. Päivitetty 19.8.2013. Saatavissa: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf [viitattu 18.1.2025].

8. STMicroelectronics NV. Tietolomake. PDF-dokumentti. 2015. Saatavissa: https://www.st.com/resource/en/datasheet/h3lis200dl.pdf [viitattu 29.1.2025].

9. Gerald. Useful Overview of Lipo Battery Voltage. Blogi. Päivitetty 26.4.2024. Saatavissa: https://www.ufinebattery.com/blog/useful-overview-of-lipo-battery-voltage/ [viitattu 20.1.2025].

10. Microchip Technology Inc. Tietolomake. WWW-dokumentti. Päivitetty 12/2020. Saatavissa: https://eu.mouser.com/datasheet/2/268/MCP1700_Data_Sheet_20001826F-3442024.pdf [viitattu 20.1.2025].

11. Gerald. Learn About Charging Lipo Batteries. Blogi. Päivitetty 22.2.2024. Saatavissa: https://www.ufinebattery.com/blog/learn-about-charging-lipo-batteries/ [viitattu 20.1.2025].

12. NanJing Top Power ASIC Corp. Tietolomake. WWW-dokumentti. s.a. Saatavissa: https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf [viitattu 20.1.2025].

13. Silvonen, K. Elektroniikka ja sähkötekniikka. Helsinki. Gaudeamus. 2018.

14. Mäkelä, M., Soininen, L., Tuomola, S. & Öistämö, J. Tekniikan kaavasto. 22. painos. Tampere. Tammertekniikka. 2021.

15. Overthrow Disc Golf. Highest Spin in the UNIVERSE | Zach Nash. Youtube. Videoleike. Julkaistu 10.9.2024. Saatavissa: https://www.youtube.com/watch?v=7CoTW91HYG4 [viitattu 20.1.2025].

16. JoeV Disc Golf. How much FORCE in a Disc Golf Throw? | Joe V Disc Golf Analysis |. Youtube. Videoleike. Julkaistu 20.12.2021. Saatavissa: https://www.youtube.com/watch?v=x4wnbKYfPLs [viitattu 25.1.2025].

17. Texas Insntruments. Tietolomake. WWW-dokumentti. Päivitetty 6/2014. Saatavissa: https://www.ti.com/lit/ds/symlink/lm2937.pdf?ts=1737626193900&ref_url=https%253A%252F%252Fwww.mouser.co.uk%252F [viitattu 24.1.2025].

18. Teddy Tech. 18650 Battery Charger Module - TP4056 Basic Review. Youtube. Videoleike. Julkaistu 4.6.2019. Saatavissa: https://www.youtube.com/watch?v=yjlozGCROBY [viitattu 24.1.2025].

19. S. James Remington. MPU-6050-Fusion. Zip-tiedosto. Päivitetty 18.11.2023. Saatavissa: https://github.com/jremington/MPU-6050-Fusion [viitattu 13.2.2025].

20. Teddy Lai. IMU Mahony filter explanation. Blogi. Päivitetty 22.4.2024. Saatavissa: https://medium.com/@k66115704/imu-mahony-filter-explanation-1ae75bf033ab [viitattu 13.2.2025].

21. MIT App Inventor. About Us. Blogi. 2024. Saatavissa: https://appinventor.mit.edu/about-us [viitattu 11.2.2025].

22. MoThunderz. Create your own App! Control an ESP32 (Arduino) via Bluetooth - Part 2. Youtube. Videoleike. Julkaistu 5.8.2021. Saatavissa: https://www.youtube.com/watch?v=OvWd_xZ12E4&t=524s [viitattu 11.2.2025].

ARDUINOKOODI Liite 1

```
#include <Wire.h>
#include <BluetoothSerial.h>



BluetoothSerial SerialBT;

//Kopioitu S.J. Remington 3/2020
int MPU_addr = 0x68;

int cal_gyro = 1;  //set to zero to use gyro calibration offsets below.

// vvvvvvvvvvvvvvvvvv  VERY VERY IMPORTANT vvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
//These are the previously determined offsets and scale factors for accel-
erometer and gyro for
// a particular example of an MPU-6050. They are not correct for other ex-
amples.
//The AHRS will NOT work well or at all if these are not correct

float A_cal[6] = {265.0, -80.0, -700.0, 0.994, 1.000, 1.014}; // 0..2 off-
set xyz, 3..5 scale xyz
// the code will work, but not as accurately, for an uncalibrated accel-
erometer. Use this line instead:
// float A_cal[6] = {0.0, 0.0, 0.0, 1.000, 1.000, 1.000}; // 0..2 offset
xyz, 3..5 scale xyz

float G_off[3] = { -499.5, -17.7, -82.0}; //raw offsets, determined for
gyro at rest
#define gscale ((250/32768.0)*(PI/180.0))  //gyro 250 LSB per d/s -> rad/s

// ^^^^^^^^^^^^^^^^^^^ VERY VERY IMPORTANT ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

// GLOBALLY DECLARED, required for Mahony filter
// vector to hold quaternion
float q[4] = {1.0, 0.0, 0.0, 0.0};

// Free parameters in the Mahony filter and fusion scheme,
// Kp for proportional feedback, Ki for integral
float Kp = 30.0;
float Ki = 0.0;

// Notes: with MPU-9250, angles start oscillating at Kp=40. Ki does not
seem to help and is not required.
// with MPU-6050, some instability observed at Kp=100, Kp=30 works well

// char s[60]; //snprintf buffer, if needed for debug

// globals for AHRS loop timing
unsigned long now_ms, last_ms = 0; //millis() timers
```

```
// print interval
unsigned long print_ms = 1000; //print angles every "print_ms" milliseconds
float yaw, pitch, roll; //Euler angle output

//Kopiointi päättyy

//heiton tunnistus
bool throw_detected = false;
bool vaihe1 = false;
bool in_air = false;
bool flight_ended = false;
unsigned long throwDetectTime = 0;
unsigned long heiton_esto;

//lentoarvojen määritys
float pyörimisnopeus;
float pyörimisnopeus_rpm;
float wobble_degree;
float maxZ = -9999;
float minZ = 9999;
float nokkakulma_lähtö;
float hyzerkulma_lähtö;
float nopeus_xy = 0.0;
float sum = 0;
float average_acc = 0;

//yleisiä
unsigned long lastPrintTime = 0;
const unsigned long printInterval = 100;
const float sensor_radius = 0.005;
const float disc_radius = 0.106;
float dt = 0.0025;  // 400 hz

//in_air taulukko
const int dataSize = 100;  // Maksimikoko taulukolle
float acc_data[dataSize];  // Taulukko kiihtyvyysarvoille
int data_index = 0;        // Indeksi taulukkoon
unsigned long prev_time = 0;
int valid_samples;
float sum2;

void setup() {

  Wire.begin();
  Serial.begin(9600);
  SerialBT.begin("ESP32_BT", true);
  Serial.println("starting");
  Wire.setClock(400000);
  //MPU6050 alustus
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);
```

```
  Wire.write(0);
  Wire.endTransmission(true);
  //MPU6050 256 Hz BW
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x1A);
  Wire.write(0x07);

  Wire.endTransmission();
  //MPU6050 16g-tilaan
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x1C);
  Wire.write(0x30);
  Wire.endTransmission(true);
}

void loop()
{
  //Kopioitu S.J. Remington 3/2020
  static unsigned int i = 0; //loop counter
  static float deltat = 0;  //loop time in seconds
  static unsigned long now = 0, last = 0; //micros() timers
  static long gsum[3] = {0};
  //raw data
  int16_t ax, ay, az;
  int16_t gx, gy, gz;
  int16_t Tmp; //temperature

  //scaled data as vector
  float Axyz[3];
  float Gxyz[3];

  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 14); // request a total of 14 registers
  int t = Wire.read() << 8;
  ax = t | Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  t = Wire.read() << 8;
  ay = t | Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  t = Wire.read() << 8;
  az = t | Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  t = Wire.read() << 8;
  Tmp = t | Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  t = Wire.read() << 8;
  gx = t | Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  t = Wire.read() << 8;
  gy = t | Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  t = Wire.read() << 8;
  gz = t | Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)

  // calibrate gyro upon startup. SENSOR MUST BE HELD STILL (a few seconds)
  i++;
```

```
  if (cal_gyro) {

    gsum[0] += gx; gsum[1] += gy; gsum[2] += gz;
    if (i == 500) {
      cal_gyro = 0;  //turn off calibration and print results

      for (char k = 0; k < 3; k++) G_off[k] = ((float) gsum[k]) / 500.0;

      Serial.print("G_Off: ");
      Serial.print(G_off[0]);
      Serial.print(", ");
      Serial.print(G_off[1]);
      Serial.print(", ");
      Serial.print(G_off[2]);
      Serial.println();
    }
  }

  // normal AHRS calculations

  else {
    Axyz[0] = (float) ax;
    Axyz[1] = (float) ay;
    Axyz[2] = (float) az;

    //apply offsets and scale factors from Magneto
    for (i = 0; i < 3; i++) Axyz[i] = (Axyz[i] - A_cal[i]) * A_cal[i + 3];

    Gxyz[0] = ((float) gx - G_off[0]) * gscale; //250 LSB(d/s) default to
radians/s
    Gxyz[1] = ((float) gy - G_off[1]) * gscale;
    Gxyz[2] = ((float) gz - G_off[2]) * gscale;

    //  snprintf(s,sizeof(s),"mpu raw
%d,%d,%d,%d,%d,%d",ax,ay,az,gx,gy,gz);
    //  Serial.println(s);

    now = micros();
    deltat = (now - last) * 1.0e-6; //seconds since last update
    last = now;

    Mahony_update(Axyz[0], Axyz[1], Axyz[2], Gxyz[0], Gxyz[1], Gxyz[2],
deltat);

    // Compute Tait-Bryan angles.
    // In this coordinate system, the positive z-axis is down toward Earth.
    // Yaw is the angle between Sensor x-axis and Earth magnetic North
    // (or true North if corrected for local declination, looking down on
the sensor
    // positive yaw is counterclockwise, which is not conventional for NED
navigation.
```

```
    // Pitch is angle between sensor x-axis and Earth ground plane, toward
the
    // Earth is positive, up toward the sky is negative. Roll is angle be-
tween
    // sensor y-axis and Earth ground plane, y-axis up is positive roll.
These
    // arise from the definition of the homogeneous rotation matrix con-
structed
    // from quaternions. Tait-Bryan angles as well as Euler angles are
    // non-commutative; that is, the get the correct orientation the rota-
tions
    // must be applied in the correct order which for this configuration is
yaw,
    // pitch, and then roll.
    // http://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Eu-
ler_angles
    // which has additional links.

    roll  = atan2((q[0] * q[1] + q[2] * q[3]), 0.5 - (q[1] * q[1] + q[2] *
q[2]));
    pitch = asin(2.0 * (q[0] * q[2] - q[1] * q[3]));
    //conventional yaw increases clockwise from North. Not that the MPU-
6050 knows where North is.
    yaw   = -atan2((q[1] * q[2] + q[0] * q[3]), 0.5 - ( q[2] * q[2] + q[3]
* q[3]));
    // to degrees
    yaw   *= 180.0 / PI;
    if (yaw < 0) yaw += 360.0; //compass circle
    //ccrrect for local magnetic declination here
    pitch *= 180.0 / PI;
    roll *= 180.0 / PI;

    now_ms = millis(); //time to print? SerialBT.print("acc_xy
");       SerialBT.print(yaw, 0);       SerialBT.print(", ");
    if (now_ms - last_ms >= print_ms) {
      last_ms = now_ms;
      // print angles for serial plotter...
      //  Serial.print("ypr ");
    }
  }
  //Kopiointi päättyy

  //kiihtyvyyden skaalaus
  float scaleFactor = 0.00243;
  float axyz_X = Axyz[0] * scaleFactor;
  float axyz_Y = Axyz[1] * scaleFactor;
  float axyz_Z = Axyz[2] * scaleFactor;

  // x ja y akselin kiihtyvyyksien yhdistäminen
  float acc_xy = sqrt(pow(axyz_X, 2) + pow(axyz_Y, 2));

  // Heiton tunnistus ja duplikaation esto
```

```
if (!throw_detected && acc_xy > 40 && millis() - heiton_esto > 5000) {
  throw_detected = true;
  heiton_esto = millis();
  throwDetectTime = millis();
}

// Ilmalennon aloitus
if (throw_detected && acc_xy > 60){
  vaihe1 = true;
}

if(vaihe1 && 55 > acc_xy){
  throw_detected = false;
  in_air = true;
  vaihe1 = false;
}

// Ilmalennon lopetus
if (in_air && (acc_xy > 80)) {
  in_air = false;
  flight_ended = true;
}


//pakotettu lopetus
if ((throw_detected || in_air) && millis() - throwDetectTime > 750){
  flight_ended = true;
  throw_detected = false;
}

//Intergoidaan nopeutta kiihtyvyydestä
if (throw_detected) {
  nopeus_xy += acc_xy * dt;
}

if (in_air) {
      if (millis() - prev_time >= dt) {  // Tallenna dt välein
          prev_time = millis();

          if (data_index < dataSize) {
              acc_data[data_index] = acc_xy;
              data_index++;
          } else {
              Serial.println("Taulukko täynnä!");
          }
      }

}


//Lennon aikaiset mittaukset
if (in_air) {
```

```
    throw_detected = false;



   // lasketaan hyzerkulma ja nokkakulma mahony-suodattimen dataan
   hyzerkulma_lähtö = pitch / 2;
   nokkakulma_lähtö = (abs(roll) * (-0.692) + 124.6) * (abs(roll) / roll);

   //lasketaan wobble
   if (abs(axyz_Z) > maxZ){
      maxZ = axyz_Z;
    }

   if (abs(axyz_Z) < minZ){
      minZ = axyz_Z;
    }



   wobble_degree = abs(maxZ - minZ - 0.3) / 9.81 * 90;

  }
  if (flight_ended) {
   in_air = false;
   throw_detected = false;


    int valid_samples = 0;
    float sum2 = 0.0;

    // Lasketaan suodatettu summa (jätetään pois ensimmäiset 15 ja viimei-
set 5 arvoa)
    for (int i = 15; i < data_index - 5; i++) {
        sum2 += acc_data[i];  // Lisätään suodatetut arvot sum2:een
        valid_samples++;  // Lasketaan, kuinka monta arvoa lisättiin
    }

    // Lasketaan keskiarvo (varmistetaan, että valid_samples > 0)
    average_acc = (valid_samples > 0) ? sum2 / valid_samples : 0;

    // Lasketaan pyörimisnopeus
    pyörimisnopeus = sqrt(average_acc / sensor_radius);
    pyörimisnopeus_rpm = pyörimisnopeus * 60.0 / (2 * M_PI);


   String data = String(nopeus_xy) + "," +
                 String(pyörimisnopeus_rpm) + "," +
                 String(nokkakulma_lähtö) + "," +
                 String(hyzerkulma_lähtö) + "," +
                 String(wobble_degree);
```

```
                SerialBT.println(data); // Lähetetään koko rivi yhdellä
komennolla
    // Nollataan seuraavaa heittoa varten
    throwDetectTime = 0;
    flight_ended = false;
    nopeus_xy = 0;
    data_index = 0;  // Nollataan indeksi uuden tallennuksen aloittamiseksi
    wobble_degree = 0;
  }
}

//Kopioitu S.J. Remington 3/2020

//-------------------------------------------------------------------------
-------------------------
// Mahony scheme uses proportional and integral filtering on
// the error between estimated reference vector (gravity) and measured one.
// Madgwick's implementation of Mayhony's AHRS algorithm.
// See: http://www.x-io.co.uk/node/8#open_source_ahrs_and_imu_algorithms
//
// Date      Author      Notes
// 29/09/2011 SOH Madgwick    Initial release
// 02/10/2011 SOH Madgwick  Optimised for reduced CPU load
// 07/09/2020 SJR minor edits
//-------------------------------------------------------------------------
-------------------------
// IMU algorithm update

void Mahony_update(float ax, float ay, float az, float gx, float gy, float
gz, float deltat) {
  float recipNorm;
  float vx, vy, vz;
  float ex, ey, ez;  //error terms
  float qa, qb, qc;
  static float ix = 0.0, iy = 0.0, iz = 0.0;  //integral feedback terms
  float tmp;

  // Compute feedback only if accelerometer measurement valid (avoids NaN
in accelerometer normalisation)
  tmp = ax * ax + ay * ay + az * az;

  // ignore accelerometer if false (tested OK, SJR)
  if (tmp > 0.0)
  {

    // Normalise accelerometer (assumed to measure the direction of gravity
in body frame)
    recipNorm = 1.0 / sqrt(tmp);
    ax *= recipNorm;
    ay *= recipNorm;
    az *= recipNorm;
```

```
    // Estimated direction of gravity in the body frame (factor of two di-
vided out)
    vx = q[1] * q[3] - q[0] * q[2];
    vy = q[0] * q[1] + q[2] * q[3];
    vz = q[0] * q[0] - 0.5f + q[3] * q[3];

    // Error is cross product between estimated and measured direction of
gravity in body frame
    // (half the actual magnitude)
    ex = (ay * vz - az * vy);
    ey = (az * vx - ax * vz);
    ez = (ax * vy - ay * vx);

    // Compute and apply to gyro term the integral feedback, if enabled
    if (Ki > 0.0f) {
      ix += Ki * ex * deltat;  // integral error scaled by Ki
      iy += Ki * ey * deltat;
      iz += Ki * ez * deltat;
      gx += ix;  // apply integral feedback
      gy += iy;
      gz += iz;
    }

    // Apply proportional feedback to gyro term
    gx += Kp * ex;
    gy += Kp * ey;
    gz += Kp * ez;
  }

  // Integrate rate of change of quaternion, given by gyro term
  // rate of change = current orientation quaternion (qmult) gyro rate

  deltat = 0.5 * deltat;
  gx *= deltat;   // pre-multiply common factors
  gy *= deltat;
  gz *= deltat;
  qa = q[0];
  qb = q[1];
  qc = q[2];

  //add qmult*delta_t to current orientation
  q[0] += (-qb * gx - qc * gy - q[3] * gz);
  q[1] += (qa * gx + qc * gz - q[3] * gy);
  q[2] += (qa * gy - qb * gz + q[3] * gx);
  q[3] += (qa * gz + qb * gy - qc * gx);

  // Normalise quaternion
  recipNorm = 1.0 / sqrt(q[0] * q[0] + q[1] * q[1] + q[2] * q[2] + q[3] *
q[3]);
  q[0] = q[0] * recipNorm;
  q[1] = q[1] * recipNorm;
```

```
  q[2] = q[2] * recipNorm;
  q[3] = q[3] * recipNorm;
}
```