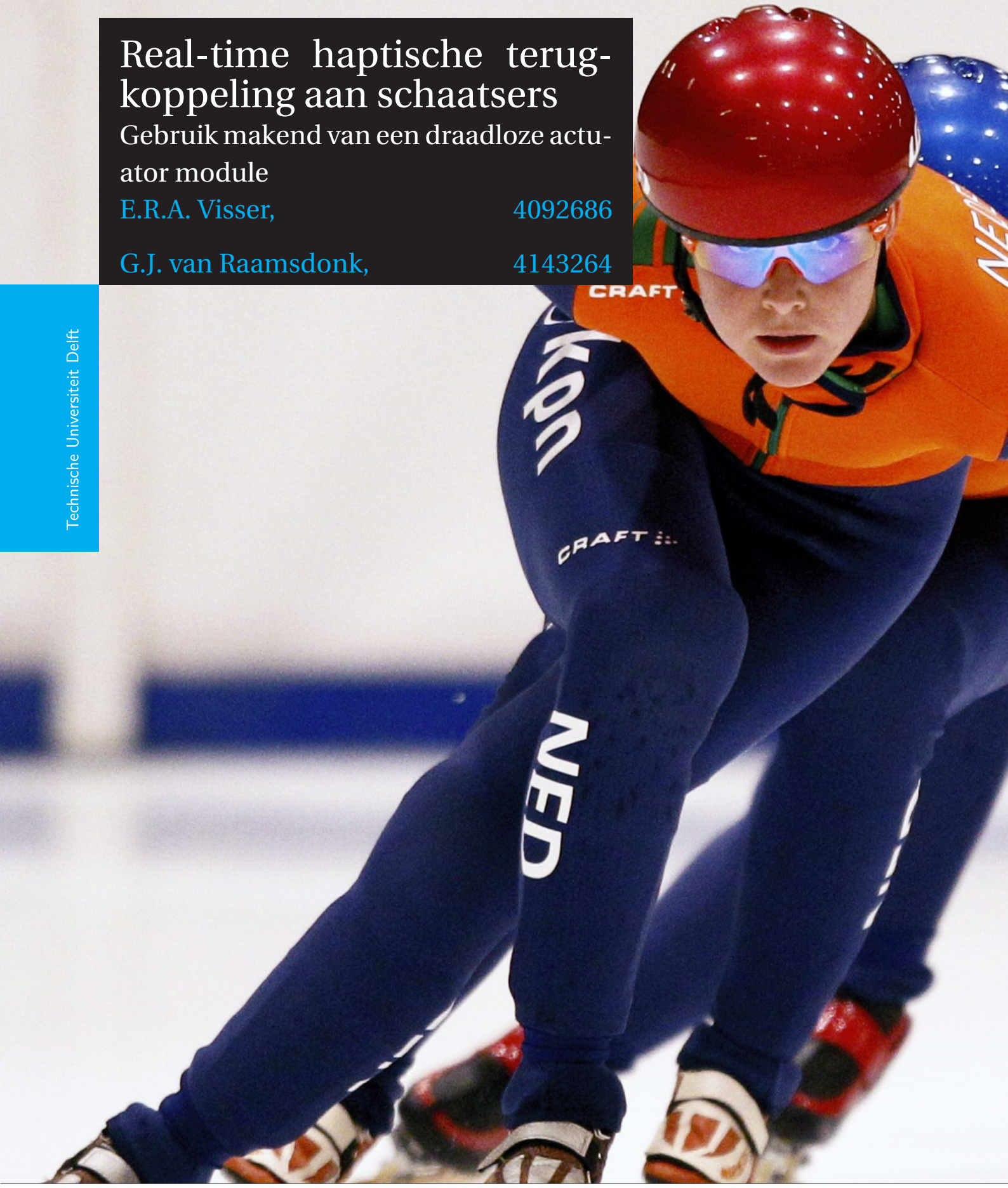
Real-time haptische terug-koppeling aan schaatsers

Gebruik makend van een draadloze actu-ator module

E.R.A. Visser, 4092686

G.J. van Raamsdonk, 4143264

Technische Universiteit Delft



TU Delft Delft University of Technology

Challenge the future

VOORWOORD

Dit document is geschreven voor de afronding van de Bachelor of Science opleiding Elektrotechniek van de technische universiteit Delft. Deze thesis is geschreven door Erwin Visser en Gert-Jan van Raamsdonk. Samen hebben zij een prototype ontwikkelt om realtime feedback te leveren in de schaatssport. Het project werd aangeboden door een externe opdrachtgever Otto den Braver van het bedrijf O'Sports. Dit project is uitgevoerd onder Prof. Dr. Kofi Makinwa . Verder gaat onze dank naar Diederik van Steen voor het delen van zijn visie en ideeën voor zijn eerdere versies en naar Ing. Jeroen Bastemeijer voor de goede begeleiding tijdens het gehele project.

E.R.A. Visser
G.J. van Raamsdonk
Delft, januari 2015

SAMENVATTING

De GonioTrainer is een draagbaar apparaat dat schaatsers helpt bij het verbeteren van hun schaatstechniek. In deze thesis wordt een manier besproken om het huidige prototype van de GonioTrainer te verbeteren en real-time terugkoppeling te geven naar de schaatser, middels een trilmotor die draadloos wordt aangestuurd. Belangrijke eigenschappen van dit systeem zijn het lage energie verbruik, de snelle draadloze communicatie over een Bluetooth Low Energy verbinding, het meten en het kleine formaat en de toekomst gerichtheid van het ontwerp. De GonioTrainer meet met een frequentie van $100Hz$ de kniehoek van de schaatser. Dit gebeurt met een resolutie van 0.087°en een standaard deviatie van 0.044°. Een algortime dat nog niet ontwikkeld is en dus ook niet in dit verslag behandeld zal worden analyseert deze kniehoek meting in real-time. Zodra hier uit komt dat deze te hoog is zal er via Bluetooth LE een signaal naar een trilmotor module gestuurd worden, welke binnen $50ms$ een trilling aan de gebruiker geeft.

Het systeem is opgebouwd uit twee apparaten. De GonioTraier, welke de hoofdmodule is, en de trilmotor, welke een submodule is. De GonioTrainer verricht de metingen, doet de analyse en beheert de Bluetooth verbinding. De trilmotor ontvangt signalen van de GonioTrainer en stuurt de trilmotor aan als dit nodig is.

De GonioTrainer zelf bestaat uit vier onderdelen. Een micro SD kaart om metingen op te slaan voor latere analyse, deze is niet geïmplementeerd in dit prototype. Een AS5600 magnetische encoder die de kniehoek van een schaatser meet met een resolutie van 0.087°. Deze wordt gelezen door een 12 bits ADC die een standaard deviatie heeft van 0.044°. Een nRF51822 Bluetooth controller voor de communicatie met de trilmotor en een STM32L152RC microcontroller die alle componenten met elkaar verbind en de analyse van de data uit voert.

De Trilmotor module bestaat uit drie onderdelen. De nRF8001 Bluetooth controller zorgt voor de draadloze communicatie met de GonioTrainer. De Picovibe trilmotor zorgt voor de terugkoppeling naar de schaatser toe en een Arduino Uno wordt gebruikt als microcontroller om beide componenten aan te sturen. De trilsterkte van de Picovibe is in te stellen in de GonioTrainer.

CONTENTS

Voorwoord iii

Samenvatting v

1 Inleiding 1

2 Probleemdefinitie 3
2.1 Huidig product . . . 3
2.2 Programma van eisen . . . 3
2.2.1 Eisen vanuit het beoogde gebruik . . . 3
2.2.2 Eisen ontwerp zelf . . . 4

3 Concept systeem 5
3.1 Communicatie . . . 5
3.1.1 Beschikbare standaarden . . . 6
3.1.2 ANT+ . . . 6
3.1.3 Bluetooth Low Energy . . . 6
3.1.4 Wireless HART . . . 7
3.1.5 ZigBee . . . 8
3.1.6 Conclusie . . . 8
3.2 Goniometer . . . 9
3.2.1 Resistieve Potentiometer . . . 9
3.2.2 Resolver . . . 9
3.2.3 Optische Encoder . . . 9
3.2.4 Magnetische Encoder . . . 10
3.2.5 Conclusie . . . 10
3.3 Opslag . . . 11
3.3.1 Capaciteit . . . 11
3.3.2 Geheugentype . . . 11
3.3.3 Interface . . . 12
3.3.4 Conclusie . . . 13
3.4 Feedback module . . . 14
3.4.1 Elektrische schokken . . . 14
3.4.2 Trillingen . . . 14
3.4.3 Conclusie . . . 14
3.5 Actuator . . . 15
3.5.1 Pager . . . 15
3.5.2 Spreekspoelen . . . 15
3.5.3 Lineaire Resonantie Actuator . . . 15
3.5.4 Piëzo-elektrisch . . . 15
3.5.5 Beenmerg geleider . . . 16
3.5.6 Conclusie . . . 16
3.6 Microcontroller . . . 17
3.6.1 Microcontrollers . . . 17
3.6.2 Microprocessor . . . 17
3.6.3 Randapparatuur . . . 18
3.6.4 Keuze . . . 19
3.6.5 Conclusie . . . 20
3.7 Systeem overzicht . . . 22

4 Implementatie **23**
4.1 Communicatie . . . 23
4.1.1 Bluetooth LE protocol stack . . . 23
4.1.2 Hardware . . . 24
4.1.3 Softdevices . . . 25
4.1.4 Services . . . 26
4.1.5 Implementatie nRF8001 . . . 28
4.1.6 Implementatie nRF51822 . . . 28
4.1.7 Conclusie . . . 29
4.2 Goniometer . . . 30
4.2.1 Implementatie AS5055A . . . 30
4.2.2 Implementatie AS5600 . . . 30
4.2.3 Toepassen . . . 30
4.2.4 Conclusie . . . 32
4.3 Actuator . . . 33
4.3.1 Implementatie Pager . . . 33
4.3.2 Conclusie . . . 33

5 Meetresultaten **35**
5.1 Communicatie . . . 35
5.1.1 Opstelling . . . 35
5.1.2 Resultaten . . . 36
5.1.3 Discussie . . . 37
5.1.4 Conclusie . . . 38

6 Aanbevelingen **39**

7 Ethische aspecten **41**

8 Conclusie **43**

A Appendices **45**
A.1 Datasheet Resolver . . . 47
A.2 Datasheet PicoVice . . . 49

B C Code **51**
B.1 Bluetooth . . . 51
B.2 Encoder . . . 68
B.3 Trilmotor . . . 75

Bibliography **77**

1

INLEIDING

Om snel te kunnen schaatsen is het van belang om een goede schaatstechniek te gebruiken. Hierbij wordt er een onderscheid gemaakt in technieken voor het rechte stuk schaatsen en in de bochten[1]. De schaatser ondervindt weerstand tijdens het schaatsen, zoals lucht-, glij- en gravitatie weerstand. De grote van luchtweerstand is afhankelijk van zijn snelheid en frontale oppervlakte. Aangezien schaatsers snelheden van wel $50 km/h$ kunnen behalen loopt de luchtweerstand op tot minstens tweederde van de totale weerstand[2]. Om de luchtweerstand te verkleinen kan een schaatser zijn snelheid verminderen of het frontale oppervlak verkleinen. Aangezien schaatsers in een zo'n kort mogelijke tijd een ronde willen schaatsen is de enigste optie om het frontale oppervlak te verkleinen. Het is dus belangrijk dat je als schaatser een correcte houding hebt waarmee je de luchtweerstand vermindert. Dit wordt veelal bereikt door dieper door de knieën te zakken.

Op het huidige moment heeft de coach de rol om de schaatser te vertellen wanneer hij dieper door de knieën moet zakken. De coaches kunnen deze feedback alleen geven als de schaatser bij hem stil staat of naar hem toeschreeuwen als hij voorbij komt. Deze rol willen we weghalen bij de coaches. Een apparaat laat de schaatser constant weten of hij dieper door zijn kniën moet zakken. Door de schaatser kennis te geven over zijn houding zal hij hierop kunnen anticiperen, waardoor trainingen efficiënter zullen zijn. Het gebruik van zo'n apparaat zal weinig veranderen aan de huidige manier van trainen. De schaatser kan nog steeds zijn ronden schaatsen op het ijs waar hij voor het gebruik ook op gleed. Er hoeft geen lopende band of op het droge worden getraind. Ook worden de coaches niet buitenspel worden gezet. Zij zijn nog steeds noodzakelijk voor de training, maar zullen nu de focus kunnen leggen op andere onderdelen van de training, zoals het zwaaien van de armen.

Coaches maken vaak een video opnamen van schaatsers tijdens wedstrijden. De coaches kunnen dan terug kijken wat de houding was van de schaatser en waarop er nog getraind moet worden. Er is een vraag van de coaches of er data in de vorm van getallen gegeven kan worden. Door data te vergelijken van wedstrijden en trainingen kunnen de coaches betere feedback geven aan hun schaatsers. Ook is het mogelijk om verbetering te zien in de houding van de schaatser ten opzichte van vorige trainingen of wedstrijden. Kortom een apparaat die gebruikt kan worden tijdens trainingen en wedstrijden om de houding en daarbij meteen de sport prestatie te verbeteren heeft potentie als product op de professionele schaats markt.

2

PROBLEEMDEFINITIE

2.1. HUIDIG PRODUCT

De opdrachtgever van de thesis wilt een trainings apparaat maken om kniehoek waarden te meten en te gebruiken als feedback naar de schaatser en coaches. Het apparaat wat zij daarvoor hebben gemaakt heet een GonioTrainer ook wel hoek meter trainer. De eerste versie bestond uit het meten van de kniehoek waarna de meetwaarden werden opgeslagen. De aansturing van de goniometer en opslag werd met een RFduino gedaan. De gemeten waarden konden met een Matlab code worden geanalyseerd waarna er feedback achteraf gegeven kon worden. Vervolgens heeft de opdrachtgever een feedback element toegevoegd om de schaatser realtime te informeren over zijn kniehoek. Het implementeren om op het juiste moment feedback gegeven hun nog niet gelukt. Zij concludeerde dat dit moment per schaatser verschillend was en niet eenvoudig kon worden opgelost. Daarom is de opdrachtgever met de goniometer en analyse functie naar de professionele markt voor schaatsers gegaan.

2.2. PROGRAMMA VAN EISEN

De opdrachtgever van dit project is O'Sports. Zij zijn ook de fabrikant en handelaar. O'Sports wilt een vernieuwing van de GonioTrainer. Hierbij moeten de functies van hoek meten en meetwaarden opslaan worden behouden. Hier moet de functie van real-time feedback naar de schaatser aan toe gevoegd worden, zonder dat dat de huidige functies verloren gaan. Het is dus een opvolger van de huidige GonioTrainer.De doelgroep van de GonioTrainer is de professionele wedstrijdschaatser. Dit zijn schaatsers die participeren in nationale kampioenschappen en internationale wedstrijden. Het is gewenst dat de GonioTrainer uitbreidbaar is, met bijvoorbeeld extra sensoren, voor gebruikers die een betere analyse wensen.

2.2.1. EISEN VANUIT HET BEOOGDE GEBRUIK

De GonioTrainer wordt ontworpen om de coach te ondersteunen in zijn taak. De primaire functies zijn wedstrijdanalyses en trainingshulp door middel van real-time feedback. De GonioTrainer zal door de gebruikers een keer per week gebruikt gaan worden. De GonioTrainer zal tijdens trainingen en wedstrijden gebruikt worden. Hierdoor is het belangrijk dat deze comfortabel te dragen is en de gebruiker niet hindert. De hoofdmodule van de GonioTrainer moet eenvoudig te bevestigen zijn, d.w.z. binnen een minuut. Het bevestigen moet zonder externe hulpmiddelen (zoals tape) mogelijk zijn. De GonioTrainer zal op de knie gedragen worden, zie figuur 2.1. Submodules kunnen gedraagd worden waar de sporter de feedback wilt ontvangen. De hoofdmodule wordt bevestigt door een elastische band om de boven en onderbeen van de schaatser over het pak heen. Aan deze band wordt door middel van polycarbonaat een arm naar knie getrokken. De GonioTrainer wordt op het midden van deze arm geplaatst en drukt tegen de knie aan. De submodule, ook wel de real-time feedback, zal ook door een band op de bovenbeen of arm worden bevestigt. Afhankelijk van drie gemeten kniehoeken wordt wel of geen feedback gegeven aan de schaatser.

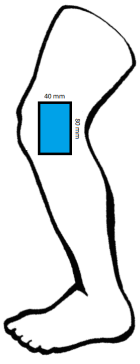


Figure 2.1: Hoofdmodule van de GonioTrainer wordt tegen de knie geplaatst

2.2.2. EISEN ONTWERP ZELF

De GonioTrainer moet kunnen communiceren met een computer, tablet of smartphone. Dit moet bij de gangbare modellen zonder toevoegingen mogelijk zijn. De GonioTrainer moet spat waterdicht zijn. Ook moet deze robuust genoeg zijn om na een val van en schaatser probleemloos te blijven functioneren. Onderdelen om de GonioTrainer te bouwen moeten zo gekozen worden, dat er geen licentie kosten verbonden aan zijn. Het is gewenst dat er per onderdeel meerdere leveranciers zijn die het zelfde, of een vergelijkbaar onderdeel verkopen. De software om een kniehoek te analyseren en feedback te geven voor schaatsers zal van te voren geïnstalleerd zijn. Bij het toepassen van de GonioTrainer in andere sporten zal de eindgebruiker thuis de software kunnen vervangen. De kostprijs van alle componenten bij elkaar mag maximaal € 150 bedragen. De GonioTrainer heeft als maximale afmetingen $80mm$ x $40mm$ x $20mm$ (l x b x h) zie figuur 2.1 en een maximaal gewicht van $150g$. De GonioTrainer zal een maximale meet onnauwkeurigheid hebben van 0.5°. Het kleinst meetbare verschil in hoek is gesteld op minimaal 0.1°. De GonioTrainer zal de kniehoek minstens 100 keer per seconde moeten meten en moet minimaal 8 uur achter elkaar te gebruiken zijn. De tijd die het mag doen om data te meten, versturen en daadwerkelijk feedback te geven is maximaal $50ms$. De gemeten kniehoeken dienen voor een uitgebreide analyse tijdens de training binnen één minuut van de GonioTrainer afgehaald te kunnen worden.Voor de realisatie van dit project is twee maanden de tijd. Hierin moet een functioneel prototype tot stand komen. Kortom de volgende systeem specificaties kunnen uit het programma van eisen worden samengevat in de tabel 2.1.

Meetfrequentie	$100Hz$
Resolutie	0.1°
Meet nauwkeurigheid	0.5°
Reactie tijd	$50ms$
Gebruiks tijd	8 uur
Afmetingen	$80mm$ x $40mm$ x $20mm$
Gewicht	$150g$
Kosten	Ca. €150

Table 2.1: Systeem specificaties GonioTrainer

3

CONCEPT SYSTEEM

Het vernieuwen van de GonioTrainer is opgesplitst in losse onderdelen, zie figuur 3.1. In deze thesis zullen alleen de volgende delen worden onderzocht en geïmplementeerd worden om een prototype te maken. Door deze prototype kan de functionaliteit worden getest op een eenvoudige beheersbare omgeving. Hierbij zijn de onderdelen zo gekozen dat deze ook te gebruiken zijn voor het definitieve product. De losse onderdelen zijn de hoekmeter, data opslag, processor(1), communicatie, processor(2) en de trilmotor als actuator.

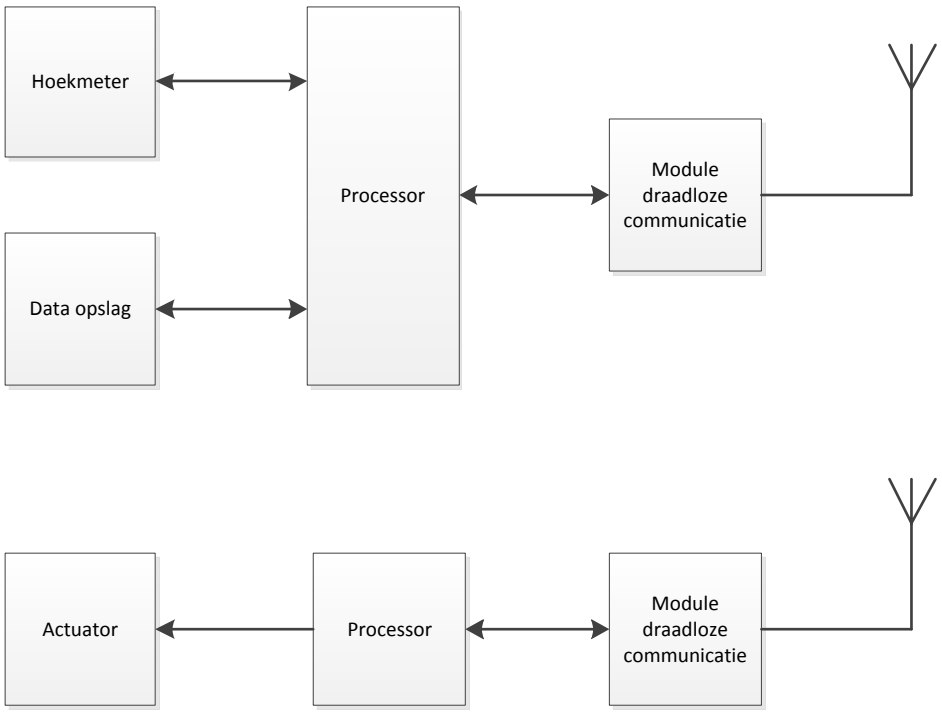


Figure 3.1: Een overzicht van de onderdelen van het concept systeem

3.1. COMMUNICATIE

De GonioTrainer moet in staat zijn om draadloos te communiceren met een feedback module. De GonioTrainer en de feedback module zullen zich altijd op het lichaam bevinden, dus de afstand hiertussen zal nooit meer dan $2mr$ bedragen. Verder moeten twee GonioTrainers met elkaar gekoppeld kunnen worden

om gelijktijdig metingen mogelijk te maken. Ook deze twee GonioTrainers zullen zich op het zelfde lichaam bevinden dus hier geldt hetzelfde bereik. Uit toekomstperspectief is het wenselijk dat de draadloze technologie compatibel is met veel gebruikte consumenten elektronica, zoals smartphones, tablets en laptops. De hoeveelheid data die verstuurd dient te worden over de draadloze verbinding varieert van commando's van enkele bytes tot meetgegevens van $2MB$. Voor het aansturen van de feedback module is het belangrijk dat de vertraging van de verbinding minder dan $50ms$ bedraagt.

3.1.1. BESCHIKBARE STANDAARDEN

Er zijn veel standaarden voor draadloze communicatie. Om een standaard te kiezen die geschikt is voor onze toepassing hebben we 18 populaire standaarden met elkaar vergeleken en gekeken welke het beste is voor ons. In tabel 3.1 zijn een aantal eigenschappen van de verschillende standaarden te zien. Het gewenste bereik van de communicatie is 2 tot $10m$. Verder is een snelheid van enkele kilobytes per seconde al genoeg om de gewenste resultaten te verkrijgen. Het energieverbruik moet laag zijn omdat de draadloze modules via een batterij of accu gevoed zullen worden. Een globale analyse van de beschikbare technologieën laat 4 kandidaten over die interessant zijn voor onze toepassing. Dit zijn ANT+, Bluetooth LE, Wireless HART en ZigBee.

3.1.2. ANT+

ANT en ANT+ zijn gepatenteerde draadloze technologieën. De technologie wordt vooral veel toegepast in sport- en gezondheidsapplicaties. Het wordt dan vooral gebruikt om sensoren met een basis station te laten communiceren. ANT+ ondersteunt een aantal netwerkmogelijkheden. Het is in staat om mesh, ster en point-to-point netwerken te creëren. Ook kan het opereren in broadcast modus, maar hiervoor moet de ontvangende kant wel actief aan het luisteren zijn. De snelheid van een verbinding is gemiddeld $20kb/s$. De vertraging in een verbinding is minder dan een milliseconde. Deze lage vertragingstijd wordt alleen bereikt als het ontvangende apparaat constant aan het luisteren is. Dit zorgt voor meer energie verbruik bij het ontvangende apparaat. Het maximale bereik is ongeveer $30m$. Het energie verbruik van een verbinding kan het beste uitgedrukt worden in watt per bit. Bij ANT ligt dit energieverbruik op $0.71\mu W/b$. De piekstroom die een verbinding vraagt is $17mA$. Deze stroom kan zonder problemen geleverd worden door een knoopcel. ANT+ is een gepatenteerde technologie, dus het is enkel verkrijgbaar in kant en klare communicatie chips. Dit maakt de implementatie vaak erg eenvoudig. ANT+ begint ook langzaam geïmplementeerd te worden in consumenten elektronica. Ongeveer 100 smartphone modellen zijn compatibel met ANT+ [3] en voor de PC kan een ANT+ adapter voor ongeveer € 10 aangeschaft worden [4].

3.1.3. BLUETOOTH LOW ENERGY

Bluetooth Low Energy (LE) is begonnen als een project van Nokia. Daar heette het toen nog Wibree. Ondertussen heeft het ook al de namen Bluetooth Smart, Bluetooth Ultra Low-Power en Bluetooth v4.0 gehad. Het is een technologie die vooral wordt toegepast in draadloze muizen, toetsenborden, audioapparatuur en ook in smartphones, tablets en laptops. Bluetooth LE ondersteunt broadcasting, mesh, en point-to-point netwerken. Het kan ook in een ster network gebruikt worden waarbij een host-device met 8 verschillende apparaten verbonden kan zijn. De snelheid van een Bluetooth LE verbinding is gemiddeld $305kb/s$. De vertraging in de verbinding is ongeveer $2.5ms$. Het bereik van een verbinding is bij een laag energieverbruik ongeveer 10 meter. Het is mogelijk om het zendvermogen groter te maken en dan kan een bereik tot 300 meter behaald worden. Een Bluetooth LE verbinding verbruikt gemiddeld $0.153microW/b$. De piekstroom van een Blutooth LE module is $12.5mA$. Dit is goed geschikt voor een knoopcel. Bluetooth is een open standaard. Dit betekent dat de specificaties om een Bluetooth apparaat te maken vrij beschikbaar zijn en iedereen deze mag gebruiken. Hierdoor zijn er veel chips en microcontrollers op de markt die Bluetooth LE geïmplementeerd hebben. Deze beschikken vaak over een application programming interface (API) die het eenvoudig maakt om een draadloze verbinding via Bluetooth LE te realiseren. Ook is de Bluetooth standaard ruim vertegenwoordigd in allerlei consumenten elektronica wat de compatibiliteit ten goede komt [4].

Technology or standard	Frequency	Range	Features	Common applications
ANT+	2.4 GHz	<10 m	Low power	Health, sports monitoring
Bluetooth	2.4 GHz	<10 m, up to 100 m with higher power	Low-power version available	Wireless headsets, audio apps
Cellular	Common cellular bands	Several km	Longer range	M2M
IEEE 802.15.4	2.4 GHz	<10 m	Multiple protocols available	Wireless networks
IEEE 802.22	470 to 768 MHz	Many miles	Designed for white spaces, cognitive radio	Broadband, backhaul, not yet used
ISA100a	2.4 GHz	<10 m	Extra security and reliability	Industrial monitoring and control
Infrared (IrDA)	800 to 1000 m	<1 m	Security, high speed	Remote control, data transfer
ISM band	Part 15 frequencies	<10 m	Low cost, simplicity	Monitoring and control
NFC	13.56 MHz	<30 cm	Security	Payment, access
RFID	125 kHz, 13.56 MHz, 902 to 928 MHz	<1 m	Low cost	Tracking, inventory, access
6LoWPAN	2.4 GHz	<10 m	Internet access	Monitor and control via Internet
UWB	3.1 to 10.6 GHz	<10 m	Low power, high-speed data	Video transfer
Wi-Fi	2.4 and 5 GHz	<100 m	High speed, ubiquity	Local networks, Internet access, broadband
Wireless HART	2.4 GHz	<10 m	HART protocol	Industrial monitoring and control
WirelessHD	60 GHz	<10 m	Very high speed	Video transfer
WirelessUSB	2.4 GHz	<10 m	Proprietary protocol	HID
ZigBee	2.4 GHz	<10 m	Mesh networks	Home, industry monitoring and control
Z-Wave	908.42 MHz	<30 m	Simple protocol	Home monitoring and control

Table 3.1: Overzicht veel gebruikte draadloze standaarden.

3.1.4. WIRELESS HART

Wireless HART is de draadloze variant van het HART protocol. Het wordt vooral toegepast in industriële sensor monitoring om processen in fabrieken te controleren. Wireless HART ondersteunt alleen ster en mesh netwerk typologieën. De snelheid van een Wireless HART verbinding is gemiddeld $250kb/s$. De vertraging in de verbinding is ongeveer $30ms$. Het bereik is rond de 10 meter.Een Wireless HART verbinding verbruikt gemiddeld 0.68µ W/b. De piekstroom hierbij bedraagt $29mA$. Dit is net mogelijk om te voeden via een knoopcel. Dit lage energieverbruik is echter met een updatetijd van 1 tot $3s$. Als er vaker informatie over verbinding gestuurd wordt neemt het energie verbruik toe [4] [5].

3.1.5. ZIGBEE

ZigBee is een standard die in 2002 gedefinieerd is door de ZigBee Alliance, welke is opgericht door een groep van 16 bedrijven. Het is ontworpen voor industriele sensor monitoring, zoals in smart meters en domotica. ZigBee ondersteunt mesh, ster en point-to-point netwerken. In de ontwikkeling is vooral rekening gehouden met de ondersteuning van grote mesh netwerken, waardoor netwerken met maximaal 65.000 knooppunten ondersteunt worden. Dit is handig voor grootschalige sensornetwerken. De snelheid van een ZigBee verbinding is gemiddeld $100kb/s$. De vertraging van een verbinding is $20ms$. Het normale bereik van een verbinding is rond de 10 meter. Dit kan met een hoger energieverbruik uitgebreid worden tot 100 meter. Het energie verbruik bij een ZigBee verbinding ligt hoger dan de andere drie technologieën. Gemiddeld verbruikt een ZigBee verbinding $185.9\mu W/b$. Ook het piekvermogen ligt hoger, namelijk $40mA$. Dit vermogen is te hoog om ondersteund te worden door een knoopcel [6] [4].

3.1.6. CONCLUSIE

Een overzicht van de details per technologie is te vinden in tabel 3.2. Uit de vier onderzochte technologieën zijn er twee die niet bruikbaar zijn. Dit zijn Wireless HART en ZigBee. ZigBee heeft een te hoog energie verbruik. Hierdoor kan deze niet gevoed worden met een knoopcel. Ook is de technologie in alle opzichten ondergeschikt, met uitzondering van de uitgebreide netwerkmogelijkheden. Deze netwerkmogelijkheden zijn echter niet nodig voor de GonioTrainer. Wireless HART is geoptimaliseerd voor sensornetwerken die elke 1 tot $3s$ een update sturen. Voor verbindingen die meer communicatie nodig hebben, zoals een feedback module die binnen enkele miliseconde aangestuurd moet worden is dit minder geschikt.

ANT+ en Bluetooth Low Energie zijn heel erg vergelijkbaar. Beiden hebben een zeer laag energie verbruik, een lage vertraging en een toereikend bereik. Bluetooth LE is iets beter in het energie verbruik en het bereik. Op de vertraging scoort ANT+ weer beter. De snelheid van de verbindingen is ook vergelijkbaar, maar ook hier is Bluetooth LE iets sneller. ANT+ wordt al veel toegepast in de sport wereld. Ook Bluetooth LE is een bewezen standaard voor sport toepassingen. ANT+ bezig is met zijn intrede in de consumenten elektronica, maar Bluetooth wordt al veel langer standaard geïmplementeerd in smartphones, tablets en laptops. Dit maakt deze technologie veel geschikter voor het toekomstperspectief waarin de GonioTrainer draadloos zal moeten communiceren met deze apparaten voor een gebruiksvriendelijker geheel. Daarom is Bluetooth Low Energie de meest geschikte draadloze technologie voor de GonioTrainer.

Technologie	**Toepasingen**	**Netwerk mogelijkheden**	**Snelheid**	**Vertraging**	**Bereik**	**Energie**
ANT+	Sport gezondheid	Star Tree Mesh Peer-to-peer	1 Mb/s	<1 ms	10 meter	Gemiddeld: 0,71uW/b Piek: 17 mA
Bluetooth Low Energy	Sport Gezondheid Beveiliging Thuis media	8 Device star Peer-to-peer	250 kb/s	2,5 ms	10 meter	Gemiddeld: 0,153uW/b Piek: 12,5 mA
Wireless HART	Industrieel Sensor monitoring Actuator control	Star Mesh	250 kb/s	30 ms	10 meter	Gemiddeld: 0,68 uW/b Piek: 29 mA
ZigBee	Domotica Industrieel	Star Tree Mesh (65000 nodes) point-to-point point-to-multipoint	250 kb/s	20 ms	10 meter	Gemiddeld: 185,9uW/b Piek: 40 mA

Table 3.2: Details per draadloze technologie.

3.2. GONIOMETER

De GonioTrainer zal in de eerste plaats een kniehoek moeten meten voordat er realtime-feedback gegeven kan worden. Een sensor die hoeken kan meten wordt een Goniometer genoemd. De GonioTrainer heeft een Goniometer in zich en zal tegen de knie worden gedrukt. De Goniometer zal een verschil in hoek moeten kunnen meten van 0.1°en een onnauwkeurigheid van 0.5°. De hoek zal met een frequentie van minstens $100Hz$ moeten worden uitgelezen. De volgende vier Goniometers zijn nader onderzocht. Dit zijn de resistieve potentiometer, resolver, optische- en magnetische encoders.

3.2.1. RESISTIEVE POTENTIOMETER

Een resistieve potentiometer bestaat uit een weerstand met drie uiteindes. De eerste is verbonden aan een spanningsbron. De tweede aan een aardpunt en de derde is verbonden aan een beweegbare contact. Dit contact beweegt tussen de twee eerste verbindingen waardoor zijn spanning verandert ten opzichte van het aardpunt. Deze spanning vertelt de verandering in positie, zie figuur 3.2. De resolutie bij een resistieve potentiometer word bepaalt door de verandering van spanning ten opzichte van de verandering in hoek. Theoretisch gezien kan de resolutie oneindig zijn: zolang er een verschil in spanning wordt waargenomen bij een veranderende hoek kan een potentiometer dit waarnemen.

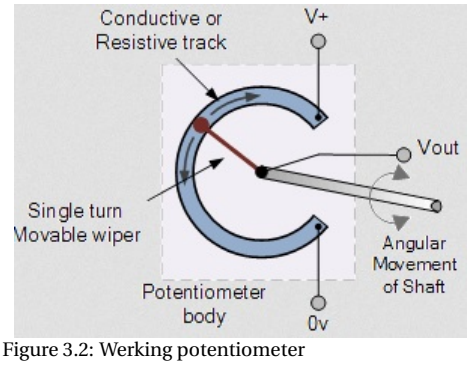


Figure 3.2: Werking potentiometer

Meet nauwkeurigheid of tolerantie kan bij een standaard resistieve potentiometer tussen de 0.01°en 2°zitten afhankelijk van de grote en materiaal van de weerstand en het verschil in hoek. Het meet nauwkeurigheid wordt bepaalt door een omzetter van analoog naar digitaal, deze omzetter zal de resolutie ook verminderen. Een omzetter kan uitgelezen worden met een frequentie van enkele megahertz en heeft resoluties van 5 tot 24bits. Het is voor de potentiometer van noodzaak dat er continu een verbinding is tussen weerstand en contact. Door ongewenste trillingen of invloed van water kan er contact verlies(open tak) optreden waardoor er geen hoek gemeten kan worden [7].

3.2.2. RESOLVER

Een resolver is een draaibare transformator die een hoek meet door middel van twee stator spoelen die 90°van elkaar vandaan zitten.. Er zal een AC bron gebruikt worden om de rotor spoel een magnetisch veld te laten opwekken. De twee stator spoelen (SIN en COS) zullen een geïnduceerde spanning opwekken waaruit de hoek bepaalt kan worden, zie figuur 3.3. Een standaard resolver heeft een frequentie tussen de 2 en $3KHz$ en kan een hoek verplaatsing meten van 40°. De kleinst meetbare hoek die gemeten kan worden ligt tussen de 0.2 tot 0.6 met een meet nauwkeurigheid van 0.05 tot 0.2°, zie Appendix A.1 voor uitgebreide uitleg van een resolver. De gemeten hoek van een resolver zal als analoog uitgangssignaal gegeven worden.

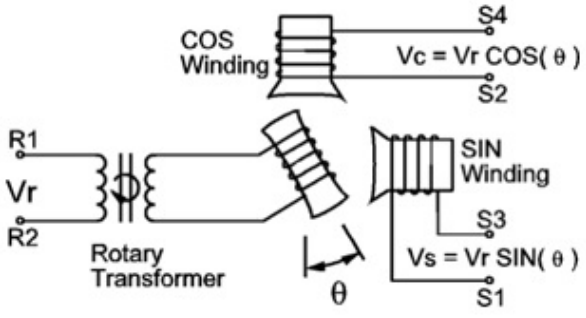


Figure 3.3: Werking resolver

3.2.3. OPTISCHE ENCODER

Een optische encoder werkt op basis van een licht- bron en sensor. Licht wordt uitgezonden op een draaischijf waar wit of zwarte delen staan, zie figuur 3.4. Het witte deel zal het licht reflecteren, terwijl zwart absorbeert.

Afhankelijk van het reflecterende licht wordt er met de sensor een digitaal uitgangssignaal gegeven. Optische encoders worden gebruikt wanneer de precisie belangrijker is dan robuustheid. De optische encoders hebben een resolutie van 0.003 tot 0.1°. Daarnaast kan de optische encoder met meer dan $1MHz$ waardes uitlezen [8]. Het is belangrijk dat het reflecterende licht goed ontvangen wordt. Bij verschuiving van draaischijf of stof tussen de lichtbron en sensor zal de reflectie worden geblokkeerd. Trillingen van 20µm zullen een verschuiving van de sensor veroorzaken waardoor verkeerde waarden worden gelezen.

3.2.4. MAGNETISCHE ENCODER

Een magnetische encoder heeft dezelfde werkingsprincipe als een optische encoder. Sensoren meten een verandering in dit geval een veranderend magnetisch veld. De verandering die wordt veroorzaakt door een draaiende magneet boven een draaischaaf met polen bepaalt de hoek verplaatsing, zie figuur 3.4. In plaats van lichtval wordt er een magnetisch veld loodrecht boven de draaischijf gehouden. Magnetische encoders hebben een nauwkeurigheid van 0.1 tot 0.83°en een lees snelheid van $50KHz$[8]. Bij een resolutie van minimaal 12bits zal de kleinst mogelijk hoek die gemeten kan worden 0.08°zijn. Magnetisch encoders zijn zijn ten opzichte van een optische encoder beter bestand tegen bevuiling of ongewenste trilling.

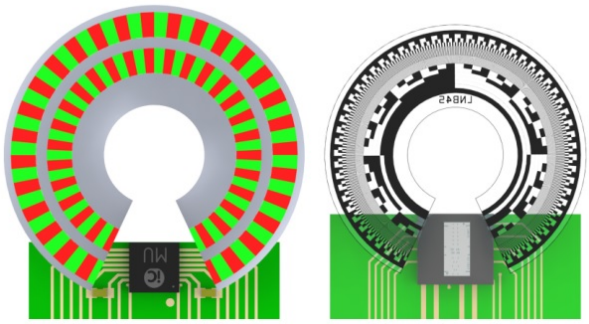


Figure 3.4: Links een magnetische draaischijf met noord en zuid polen paren, Rechts een optische draaischijf met witte en zwarte reflecterende delen

3.2.5. CONCLUSIE

Elke Goniometer kan de hoek meten met meer dan $100Hz$. De Goniometer moet ook hoeken onderscheiden van ten minste van 0.1°. Een resolver kan zo'n kleine hoek niet onderscheiden. De omgeving waar de GonioTrainer gebruikt zal bestaan uit veel ongewenste trillingen. De verbinding kan verloren gaan tussen de weerstand en het contact bij een resistieve potentiometer. De omgeving is te ruig voor een resistieve potentiometer om te gebruiken als Goniometer. De optische encoder is ook te kwetsbaar tegen deze ruige omgeving. Een resolver zal ook een analoog naar digitaal signaal omzetter nodig hebben om het signaal verder te gebruiken voor de GonioTrainer. Een magnetische encoder kan een de hoek nauwkeurig meten tussen de 0.1 tot 0.83°en valt onder de eis van 0.5°. Ook is hij tegen ongewenste trillingen ten opzichte van een optische encoder of resistieve potentiometer.

3.3. OPSLAG

Een van de functies van de GonioTrainer is dat deze in staat is om van een wedstrijd of training een volledige meting te verrichten die geanalyseerd kan worden. Op basis van deze analyse kan de coach advies geven aan de schaatser. Voor de analyse van de meting zijn er berekeningen nodig die te complex zijn om door de GonioTrainer uitgevoerd te worden. Deze berekening worden daarom uitbesteed aan een computer, smartphone of tablet. Hiervoor is de onbewerkte (ruwe) meet data van de GonioTrainer nodig. De GonioTrainer moet dus in staat zijn om gedurende een hele wedstrijd of training de ruwe meet data op te slaan. Tijdens de trainingen is het gewenst dat de schaatser snel een analyse van zijn techniek krijgt. Het is dus belangrijk dat de data op een eenvoudige en binnen één minuut van de GonioTrainer afgehaald kan worden.

3.3.1. CAPACITEIT

Er moet voldoende opslagcapaciteit aanwezig zijn om een hele training te kunnen meten. Het is gewenst dat het mogelijk is om meerdere trainingen te meten zonder data van de GonioTrainer te verwijderen. Dit vergroot het gebruiksgemak.

Momenteel wordt er gemeten met een 12 bits Goniometer. De meetfrequentie is $100Hz$. De duur van een training wordt geschat op 1 uur. Als de GonioTrainer een gehele training zal meten komt dit neer op

$$12 * 100 * (60 * 60) = 4320000bits = 540000Bytes = 527,34kB. \tag{3.1}$$

Tegenwoordig zijn geheugens van enkele gigabytes relatief goedkoop [9]. Uit deze eerste berekening blijkt dus het doel van meerdere trainingen achter elkaar meten goed haalbaar is. Als de ruwe data wat meer opslagruimte in beslag neemt is dit nog geen probleem.

Veel computer standaarden gaan uit van 8 bits datablokken (bytes), zoals het ASCII systeem [10]. De 12 bits datablokken vormen geen standaard en zij dus lastiger in te lezen. Het is voor het schrijven van een analyseprogramma een stuk makkelijker als de ruwe data opgeslagen is en als decimale waarde in ASCII tekst staat. Aangezien berekening 1 indiceert dat er nog geen gebrek is aan opslagruimte maakt het niet uit dat de ruwe data meer ruimte inneemt. Ook is het fijn als de waarde gescheiden zijn door een komma. Dan wordt en comma seperated value (csv) bestand gecreëerd en dit wordt door veel programma's en libraries als standaard input geaccepteerd. Een 12 bits waarde is in decimale notatie maximaal $2\hat{1}2 = 4096$. Er zijn dus 4 ASCII tekens nodig om een meetwaarde op te slaan. Als de komma die de waarden van elkaar scheid meegenomen wordt komt het totaal op 5 tekens per meting. Een non-stop meting van een uur wordt dan

$$5 * 8 * 100 * (60 * 60) = 14400000bit = 1800000byte = 1757,81kB. \tag{3.2}$$

Het toekomstperspectief van de GonioTrainer is dat er meer sensoren aangesloten zullen worden om een beter meting te verkrijgen. De additie van een accelerometer is erg waarschijnlijk. Om zeker te zijn dat dit systeem ook in de toekomst blijft werken wordt in de laatste berekening een drie-as accelerometer meegenomen die voor elke as een 12 bits waarde geeft, dus inclusief komma ook 5 ASCII tekens. Ook wordt hier bij elke meting een unix timestamp toevoegt welke 32 bits is, dus 11 tekens inclusief komma. De grootte van een meting van een uur met dit systeem bedraagt

$$(5 + 3 * 5 + 11) * 8 * 100 * (60 * 60) = 89280000bit = 11160000byte = 10,64MB. \tag{3.3}$$

Op een geheugen van $512MB$ past dus $512/10.64 = 48,12$ uur aan ruwe meet data. Dit is ruim voldoende om te voldoen aan de eis dat minimaal 1 volledige training gemeten moet kunnen worden. Ook wordt hiermee voldaan aan de wens dat meerdere trainingen achter elkaar gemeten kan worden zonder dat het wissen van data noodzakelijk is.

3.3.2. GEHEUGENTYPE

Er zijn veel verschillen manieren om digitale data op te slaan. Het grootste verschil ligt in vluchtig (volatile) en niet-vluchtig (non-volatile) geheugen. Vluchtig geheugen wordt veel gebruikt als werkgeheugen. Het heeft de eigenschap dat het continu gevoed moet worden met energie, anders gaat de opgeslagen data verloren. Bij niet-vluchtig daarentegen blijft de opgeslagen data bewaard als de voedingsspanning van het geheugenelement verwijdert wordt. Dit type geheugen wordt meestal gebruikt voor opslag van data die voor langere periode bewaard moet blijven. Voor de opslag van de meet data van de GonioTrainer is dus niet-vluchtig geheugen nodig. Er zijn veel verschillende vormen van niet-vluchtig geheugen. Hieronder worden de varianten vergeleken die qua afmeting, gewicht en capaciteit geschikt zijn voor de GonioTrainer.

EEPROM

De meest gebruikte variant van niet-vluchtig geheugen in kleine toepassingen is electrically eraseble read only memory (EEPROM). Dit type geheugen gebruikt floating gate transistoren om bits op te slaan. Bij deze opslagtechniek wordt door middel van “Fowler-Nordheim tunneling” of “hot-carrier injection” de lading op de floating-gate van de transistor verandert, waardoor deze een digitale ‘1’ of ‘0’ representeert. In deze thesis zal niet dieper op deze techniek in worden gegaan.

Het voordeel van EEPROM is dat in het geheugen individuele bytes beschreven en verwijdert kunnen worden. Het nadeel is dat EEPROM-chips die op de markt aangeboden worden een capaciteit van enkele bytes tot een paar megabyte hebben. Een ander nadeel is dat de geheugencellen maar 10 duizend tot 1 miljoen keer herschreven kunnen worden. Bij elke schrijf actie blijven elektronen achter op de floating gate. Hierdoor wordt het verschil tussen een logische 0 en een logische 1 bij elke keer dat een cel herschreven wordt kleiner. Uiteindelijk is het verschil zodanig klein dat er geen duidelijk onderscheid meer is en de geheugencel onbruikbaar is.

FLASH

Flash is een EEPROM soort. De fysische opslagtechniek is het zelfde. Flash gebruikt dus ook floating gate transistoren. Het verschil zit in de manier waarop het geheugen gestructureerd is. Bij EEPROM kan elke byte afzonderlijk beschreven worden. In flash is het geheugen opgedeeld in blokken en kunnen enkel complete blokken beschreven or verwijdert worden. Hierdoor is het mogelijk om een veel grotere geheugencapaciteit aan te bieden. De capaciteit varieert van enkele megabytes tot tientallen gigabytes. Ook is de schrijfsnelheid groter dan EEPROM door deze structuur. Het nadeel is dat altijd een geheel blok aangepast moet worden, wat voor een snellere slijtage zorgt. Het aantal schrijfcycli ligt dus lager bij flashgeheugen.

Er zijn twee typen flash opslag, dat zijn NAND- en NOR-Flash. Het verschil zit in de schakeling van transistoren die een bit opslaat. Beide typen hebben voor en nadelen. NOR-Flash kan erg snel gelezen worden, maar het schrijven en wissen van data duurt erg lang vergeleken met NAND-Flash. De opslagdichtheid van NOR-Flash ligt gemiddeld een factor 100 lager dan die van NAND-Flash [11] [12]. Dit zorgt ervoor dat NOR-Flash ideaal is als bijvoorbeeld programmageheugen. Dit hoeft niet vaak beschreven te worden en is vaak niet groot, maar moet wel erg snel uitgelezen worden. Voor bulk data opslag is NAND-Flash meer geschikt omdat dit sneller overschreven kan worden en de capaciteit hiervan hoger is.

FRAM

Ferroelectric RAM (FRAM) is een geheugenelement dat het zelfde gestructureerd is als Dynamic RAM (DRAM). Bij DRAM wordt een bit opgeslagen op een enkele transistor, waarvan de waarde bewaard blijft door een dielectricum (capaciteit) die daaraan gekoppeld is. Omdat deze capaciteit langzaam leeg loopt moet de waarde ongeveer elke 64 ms ververst worden. Dit zorgt ervoor dat de data verloren gaat zodra er langer dan 64 ms geen energie wordt geleverd aan een DRAM module. FRAM verschilt van DRAM doordat het geen dielectricum, maar een ferro-elektrische laag gebruikt om de waarde van de transistor op te slaan. Deze laag verliest zijn ferro-elektriciteit niet vanzelf, dus blijft de waarde bewaard, ook als er geen energie meer wordt geleverd. Het voordeel van FRAM is dat het sneller kan schrijven, minder energie verbruikt en veel meer schijfcycli aan kan dan flash. Het nadeel van FRAM is dat het een erg jonge technologie is en daarom nog relatief duur. Ook is de capaciteit niet toereikend.

CONCLUSIE

Er zijn nog meer technologieën die niet-vluchtige opslag mogelijk maken zoals Magnetoresistive RAM (MRAM), Phase-change RAM (PRAM) en non-volatile Static RAM (nvSRAM). Deze technologieën begeven zich echter allemaal nog in een beginstadium en zijn daarom nog erg duur met een lage opslagcapaciteit.

De technische verschillen tussen EEPROM en flash zijn erg klein, maar door de geheugenstructuur biedt flash een opslagcapaciteit die veel groter is dan EEPROM. Hierdoor is flash de enige geschikte technologie voor de GonioTrainer. Flash is met zijn snelle schrijfsnelheid, hoge capaciteit, grote geheugendichtheid en lage kosten de meest geschikte opslagtechnologie.

3.3.3. INTERFACE

Flash geheugen is beschikbaar in via verschillende interfaces. Het geheugen zelf zit altijd ingepakt in een intergrated circuit (IC). Deze IC’s kunnen direct aan een microcontroller gekoppeld worden. Het is echter

ook mogelijk om de IC's te benaderen via een module, zoals in USB-sticks en SD-kaarten gebeurt.

FLASH IC'S

Als het flash geheugen als IC's geïntegreerd wordt is het geheugenelement direct aan de microcontroller gekoppeld. Dit heeft als voordeel dat er minder componenten nodig zijn en de snelste schrijfsnelheid bereikt kan worden. Ook zullen de kosten lager zijn dan andere oplossingen. Het nadeel van deze implementatie is dat het lastig is om de data in het flash geheugen te delen met nadere apparaten. Omdat het geheugen geïntegreerd is in het systeem moet er een aparte interface gemaakt worden die de data voor andere apparaten toegankelijk maakt. Er zijn standaarden om de communicatie tussen het geheugen en een ander apparaat mogelijk te maken, zoals de USB mass storage device class (MSDC) [13]. Dit zorgt echter wel voor extra complexiteit in de microcontroller en de software die hierop draait, omdat er een USB interface aanwezig dient te zijn en de MSDC geïmplementeerd moet worden.

SECURE DIGITAL

Er zijn een aantal standaarden die een interface voor het flash geheugen geïmplementeerd hebben. Voorbeelden hiervan zijn CompactFlash en SmartMedia, maar de meest bekende en meest gebruikte standaard is wel die van Secure Digital. Een SD-kaart biedt de voordelen van NAND-Flash met een eenvoudige compatibiliteit met verschillende systemen. Veel computers en laptop's hebben een SD-kaart lezer ingebouwd. Als dit niet het geval is kan een kaartlezer voor enkele euro's aangeschaft worden. Ook tablets en smartphones zijn grootschalig uitgerust met een SD-kaart slot.

Er zijn drie verschillende verpakkingen waarin de SD-kaart aangeboden wordt. Dit zijn SD, mini-SD en micro-SD. De laatste heeft een oppervlakte van slechts 1,65cm^2, een gewicht van 0,35 gram en wordt veel in draagbare apparaten gebruikt.

Een SD-kaart is via verschillende protocollen te benaderen. Voor hoge snelheden zijn er de 1 bit en 4 bit SD protocollen. Deze geven wel meer complexiteit in de software. De andere mogelijkheid is om de SD-kaart aan te sturen via een Serial Peripheral Interface (SPI). De lees en schrijf snelheid van de kaart ligt in dit geval lager dan wanneer de SD protocollen gebruikt worden, maar omdat SPI door veel microcontroller ondersteunt wordt is dit een stuk eenvoudiger. Ook is het SPI protocol een open standaard waar de SD protocollen eigendom zijn van de SD Association. Over het SPI protocol is dus veel meer vrij toegankelijke informatie en hoeven ook geen licentiekosten betaald te worden.

3.3.4. CONCLUSIE

Voor het prototype van de GonioTrainer is het beter om de opslag via een SD-kaart te laten verlopen welke met een SPI protocol aangestuurd wordt. Dit is wel duurder dan het gebruik van een flash IC en geeft ook een lagere lees en schrijf snelheid op de GonioTrainer zelf. De implementatie is echter eenvoudiger en de lees snelheid van een SD-kaart die in een computer of laptop gestoken wordt ligt hoger dan wanneer dit via de microcontroller van de flash chip zou moeten. Als de planning het toelaat kan een poging gedaan worden om de SD-kaart via de MSDC interface te benaderen vanaf een computer of laptop. Als dit haalbaar blijkt kan de SD-kaart in een vervolg model vervangen worden door een flash IC.

3.4. FEEDBACK MODULE

De GonioTrainer zal real-time feedback moeten leveren aan de schaatser. Daarvoor zal de GonioTrainer één van de zintuigen moeten prikkelen. Er zal een keuze gemaakt moeten worden tussen zien, horen, ruiken, proeven of voelen. Wanneer een gevoel wordt gebruikt om de schaatser te informeren noemen we dit haptische feedback. Haptische feedback is de meest voorkomende vorm in biomedische toepassingen[14]. Tijdens het schaatsen zijn de audiovisuele waarnemingen van een schaatser al veelvuldig belast met informatie, zoals aanwijzingen van de coach, schaatsers in de buurt, het startsignaal en dergelijke. De tastzin is echter zeer weinig belast en vormt hierdoor een ideaal kanaal om de schaatser feedback te geven. In deze toepassing heeft haptische feedback dus de voorkeur.Er zijn twee vormen van haptische feedback onderzocht. Dit zijn de elektrische schokken en trillingen.

3.4.1. ELEKTRISCHE SCHOKKEN

Een elektrische schok ervaar je als een elektrische stroom in of uit het lichaam loopt. De reactie op een schok is afhankelijk van veel factoren. Eén van de belangrijkste factor is de toestand van de huid. Wanneer de huid nat is van zweet of water zal de stroom beter geleiden, terwijl een droge huid meer weerstand kan bieden. Een natte huid heeft bijvoorbeeld een weerstand van zo'n $100 k\Omega$ en een droge $1 M\Omega$. Bij een zelfde spanning zal er bij een natte huid een stroom lopen die duizend keer zo groot is [15]. Andere factoren zijn de plek op het lichaam waar de schok gegeven wordt of je gevoelig ben en zelfs man of vrouw bent. Zo zal een schok het sterkst aanvoelen als je een vrouw bent én de schok bij een zenuwuiteinde wordt gegeven [16].

3.4.2. TRILLINGEN

Trilling is een constante beweging uit een evenwichtsstand die zich herhaalt. De snelheid, tijdsduur en sterkte van de trilling bepalen hoe een trilling ervaart wordt. Onderzoek van Prof. Miwa T heeft aangetoond dat trillingen van bepaalde snelheid na een tijdsduur geen toename in ervaring oplevert. Oftewel de trilling wordt nauwelijks meer opgemerkt. Zo zal een trilling tussen de 2-60 Hz na $2s$ terwijl een trilling van 60-200 Hz al na $0.8s$ als gewoon opgemerkt[17] worden. Het is ook afhankelijk welke persoon de trillingen ontvangt. Een tik tegen de schouders is een vriendschappelijke gebaar bij mannen, terwijl dit voor vrouwen als pijnlijk ervaren wordt. Iedereen persoon ervaart trillingen anders. Uit onderzoek blijkt dat snelheid of frequentie van een trilling meer effect heeft dan de amplitude op de ervaring. Een stijging in frequentie heeft een significant effect, terwijl de amplitude weinig uitmaakten[18].

3.4.3. CONCLUSIE

Haptische feedback geven afhankelijk van hun kniehoek is extreem complex. Geen persoon is gelijk en kan elke ervaring anders beleven dan de persoon naast hem. Echter er is wel een duidelijk verschil tussen een elektrische schok en trillingen. De ervaring van een elektrische schok is afhankelijk van de conditie van de huid. Een droge of natte huid laat een andere stroom door het lichaam trekken. Een trilling is constant ongeacht of de schaatser zweet of niet. Wel zal de frequentie van de trilling afgesteld moeten worden per de schaatser voor wat hij of zij goed opmerkt en niet.

3.5. ACTUATOR

Apparaten die elektrische energie omzetten in mechanische energie worden motoren genoemd. De keuze van motor wordt bepaald door de grote van trilling sterkte(amplitude) en energie verbruik. De GonioTrainer moet minstens 8 uur meegaan per gebruik. Vanuit gaande dat een AAA batterij op een nominale spanning van $1.5V$ een capaciteit heeft van $1200mAh$, dan mag de alleen motor maximaal $150mA$ gedurende de 8 uur trekken als hij constant zal trillen. Van uitgaande dat de motor slechts één vierde van de tijd trilt wordt de grens gelegd op $600mA$.Voor de ervaring van de trilling wordt er ook gekeken naar richting waarin de trilling wordt gegeven. Dit kan parallel of loodrecht op de huidoppervlak zijn. Als laatst zal ook naar de tijd gekeken worden die het kost om de motor op maximaal vermogen te laten trillen. De opstart tijd kan bepalend zijn wanneer de haptische feedback gegeven moet worden vanaf de Goniometer naar de motor.

3.5.1. PAGER

Een pager is een gelijkspanning motor met een massa aan een schacht, zie figuur 3.5. Een pager geeft trillingen doordat een massa slechts over een deel van de schaft bevestigt is. Er zal een middelpuntzoekende kracht(F_0) uitgeoefend worden. De grote van deze kracht wordt bepaalt door afstand r tot het middelpunt van massa m op een hoeksnelheid ω. De schaft loopt parallel over de huid. Wanneer de schacht draait zal er een middelpuntzoekende kracht werken op de massa. Dit levert een trilling in de x(blauw) en z(oranje) richting. De z-richting geeft een trilling loodrecht op de huid. Een standaard pager kan trillingen leveren tussen de 0.5 en $6G$ en wordt gevoed op 1.5 of $3V$. De motor trekt tussen de 130 en $250mA$ bij nominale spanning en belasting. De typische opstart tijd van de pager bedraagt $23ms$.

Figure 3.5: Een pager trilt zowel over als tegen de huid

3.5.2. SPREEKSPOELEN

Een spreekspoel of knoopceltrilmotor werkt op dezelfde principe als een pager. Het verschil tussen de twee motoren ligt in de richting waarin de schaft staat gericht. Bij een spreekspoel staat deze loodrecht op de huidoppervlak aangezien je de platte kant tegen de huid drukt. Hierdoor worden er trillingen veroorzaakt in de x(blauw) en z(geel) richting, wat nu leidt tot trillingen **over** de huid. Een ander verschil met een pager is amplitude van een trilling die een knoopceltrilmotor kan leveren. De sterkte bij een pager kan oplopen tot $6G$. Een knoopceltrilmotor kan slechts $2G$ leveren. Standaard hebben zij een DC spanning van 1.5 of $3V$. De stroom loopt tussen de 55 en $80mA$ bij een nominale spanning en belasting. De opstart tijd met lag kan tot $140ms$ duren.

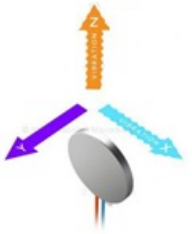


Figure 3.6: Een spreekspoel trilt over de huid

3.5.3. LINEAIRE RESONANTIE ACTUATOR

Een lineaire resonantie actuator(LRA) is een wisselspannings motor, die op soort gelijke manier werkt als een luidspreker. In tegenstelling van een luidspreker werkt een LRA op een klein deel van de frequentie spectrum. Dit komt omdat een LRA aan een magneet vast met een veer ertussen. Wanneer de magneet door een wisselspanning aan het trillen wordt gezet, zal de veer mee bewegen. Afhankelijk van het materiaal van de veer zal hij de trilling van de magneet versterken op zijn resonantiefrequentie. Net zoals de knoopceltrilmotor wordt een LRA met de platte kant tegen de huid bevestigt. De trillingen die een LRA veroorzaak staat in y(paars) richting, dit is gericht **in** huid, zie figuur 3.7 De sterkte van de trilling is wel gelijk aan die van een knoopceltrilmotor zo'n $1.5G$. Een standaard LRA werkt op $2V$ wisselspanning en verbruikt 69 tot $90mA$ bij een nominale spanning en belasting. De opstart met lag tijd kan tot $45ms$ duren.

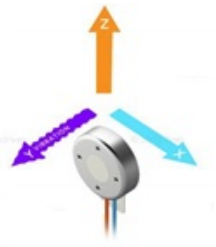


Figure 3.7: Een lineaire resonantie actuator trilt tegen de huid

3.5.4. PIËZO-ELEKTRISCH

Een andere methode om de sporter een trilling te geven als feedback is een piëzo elektrische motor. De motor werkt op basis van het piëzo elektrisch effect: wanneer er een elektrische spanning over een bepaalde stof wordt gezet dan vervormt het. Stoffen die het piëzo elektrisch effect kunnen ondervinden zijn rubber, wol, hout, kwarts en rietsuiker. De kristalvormen van

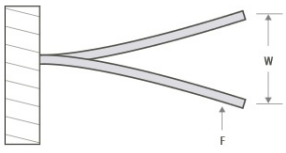


Figure 3.8: Een piëzo elektrische motor trilt tegen de huid

deze stoffen kunnen zich uitzetten tot enkele millimeters van $0.79mm$, maar wel met een amplitude van $35G$ [19]. De grote van gelijkspanning om het piëzo elektrisch effect te laten plaats vinden ligt tussen de 50 tot $200V$. Een standaard piëzo motor die op gelijkspanning van $100V$ werkt trekt <$0.01mA$. Ook kan er een sinusoïde(AC) als ingangsspanning worden aangeboden, maar dan trekt hij $1.3mA$. Als laatst is de piëzo motor ook interessant vanwege zijn afmetingen. De dikte is slechts $0.2mm$ hoog.

3.5.5. BEENMERG GELEIDER

Als laatst wordt er gekeken naar een combinatie van haptische feedback en geluid. Geluid wordt waargenomen door trillingen: geluid komt langs de oorschelp en gehoorgang op de trommelvlies wat gaat trillen. Vloeistoffen in het binnenoor worden door deze geluidstrillingen in beweging gebracht in het slakkenhuis. In het slakkenhuis zitten haarcellen die verbonden zijn met het gehoorzenuw die geluidsignalen doorgeven aan de hersenen en ons laten horen[20]. Beenmerg geleiding geeft geluidstrillingen direct aan het slakkenhuis door. De geleider bestaat onder uit een klein titanium gedeelte dat in de schedel wordt bevestigd en in ongeveer 2 maanden vastgroeit, waarna het stevig vastzit [21].

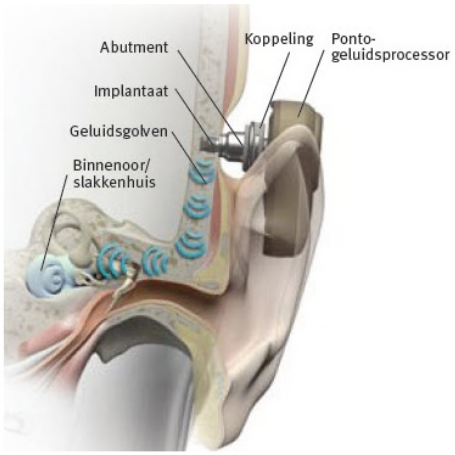


Figure 3.9: Een beenmerg geleider levert trillingen via bot rechtstreeks aan het slakkenhuis

3.5.6. CONCLUSIE

Geen enkele actuator zal meer dan $600mA$ aan stroom trekken wanneer hij één vierde van de tijd staat te trillen in een 8 uur durende training. Voor de implementatie zullen de pager en knoopceltrilmotor het eenvoudigst zijn om te implementeren. Het aansluiten van een gelijkspanning bron is genoeg om deze actuatoren te laten trillen. Een LRA heeft een wisselspanning bron en zal moeten afgesteld worden op de juiste frequentie om een maximale amplitude te krijgen. Een afwijking van bijvoorbeeld $10Hz$ zal de amplitude erg verzwakken of de LRA stil laten staan. Er zijn daarom technieken bedacht om automatisch de juiste resonantie frequentie te zoeken. Texas Instruments's Auto-Resonance is een voorbeeld die uit zich zelf de juiste bandbreedte opzoekt[22]. Een andere methode zal een frequentie controller zijn. Doordat het toepassen van pager eenvoudiger is dan een DC naar AC omvormer met een frequentie controller zal er niet gekozen worden voor de LRA. Een piëzo elektrische motor zal een boost omvormer nodig hebben om een hoogte van $200V$ te halen en een beenmerg geleider zal een operatie nodig om hem aan te sluiten tegen de schedel. De actuator die gebruikt zal gaan worden als real-time feedback module is de pager. De pager is relatief eenvoudig te implementeren en geeft trillingen tegen de huid. Er wordt een aanname gedaan dat de trillingen van een knoopceltrilmotor, die over de huid gaan, niet sterk genoeg zijn om op te merken tijdens het schaatsen.

3.6. MICROCONTROLLER

In de voorgaande paragrafen zijn de verschillende componenten van de GonioTrainer grondig bekeken. De communicatie standaard, de Goniometer, het opslagmedium en de feedback actuator zijn bepaald. Nu moet er nog een microcontroller gekozen worden die al deze componenten aan kan sturen en de benodigde data analyses uit kan voeren. Er zijn twee microcontrollers nodig. Eentje die het feedback element aan zal sturen en een die in de GonioTrainer gebruikt zal worden. De microcontroller in het feedback element moet alleen in staat zijn een Bluetooth verbinding op te kunnen zetten en een trilmotor aan te sturen. Aan het feedback element zullen geen toevoegingen gedaan worden in toekomstige versies, dus deze hoeft alleen gesckikt te zijn voor controle. Het is voor de opdrachtgever echter wel belangrijk dat het prototype van de GonioTrainer doorontwikkeld kan worden. Bij de keuze van de microcontroller in de GonioTrainer moet dus ook rekening gehouden worden met wensen in de toekomst. De microcontroller moet dus naast ondersteuning voor de interfaces die nodig zijn om de huidige componenten aan te sturen ook genoeg overige interfaces hebben om toekomstige sensoren of actuatoren aan te sturen.

In een latere implementatie zal ook een code geïmplementeerd moeten worden die de meetgegevens real-time analyseert en op basis hiervan directe feedback geeft aan de schaatser. Het is mogelijk dat voor een dergelijk algoritme geavanceerde signaalbeweringen nodig zijn, zoals bijvoorbeeld transformaties naar het frequentiedomein. De microcontroller moet dus een processor bevatten die deze bewerken in real-time uit kan voeren.

Ten slotte is het belangrijk dat dit allemaal zo energie zuinig als mogelijk gebeurd. De GonioTrainer is een draagbaar apparaat, wat betekent dat de voeding uit een accu of batterij moet komen. Voor zowel de gebruiksvriendelijkheid als voor het milieu is het noodzaak dat de GonioTrainer lang mee kan op een enkele lading. Het minimum hiervoor is gesteld op 8 uur.

3.6.1. MICROCONTROLLERS

Een microcontroller is een elektronisch component dat gebruikt wordt om andere componenten aan te sturen of te analyseren. Een microcontroller is een Intergrated Circuit waarin een microprocessor is samengevoegd met randapparatuur. In de microprocessor vinden alle berekeningen plaats. Ook stuurt de microprocessor de randapparatuur aan. De randapparatuur is hardware die geoptimaliseerd is voor enkele taken. Zo zijn er in microcontrollers vaak timers te vinden die kunnen werken als stopwatch of wekker. Ook is er randapparatuur die interfaces implementeert, zoals de veelgebruikte SPI, I^2C of UART interfaces. Het feit dat de randapparatuur geïmplementeerd is voor specifieke taken heeft twee voordelen. Als eerste kunnen zij de taken die hebben ze hebben door de geöptimaliseerde implementatie vaak sneller uitvoeren dan de microprocessor zelf. Ten tweede kunnen ze, omdat het allemaal los staande onderdelen zijn, parallel aan elkaar werken. Als er data over een interface verstuurd moet worden kan de microprocessor hier in een paar klokcycli een randapparaat voor configureren en daarna weer verder met andere taken terwijl het randapparaat tegelijkertijd zijn taak uitvoert. Dit bevorderd de efficiëntie van het gehele systeem.

De keuze van een geschikte microcontroller hangt dus af van twee aspecten. De eerste is de microprocessor. Deze moet geschikt zijn om de berekeningen die gedaan dienen te worden binnen een gesteld tijdlimiet uit te voeren. Het andere aspect is dat er voldoende randapparatuur beschikbaar is in de microcontroller om de communicatie met andere componenten effieciënt te laten verlopen.

3.6.2. MICROPROCESSOR

De microprocessor is de kern van de microcontroller. Alles wordt hierdoor aangestuurd en elke taak die niet aan een randapparaat uitbesteed kan worden moet door de microprocessor uitgevoerd worden. Het belangrijkste bij het kiezen van een microprocessor is de instructieset. Een instructie is een berekening die de de microprocessor in één klokslag uit kan voeren. De instructieset is de verzameling van alle instructies die de ondersteund worden door de microprocessor. Over het algemeen geldt de regel dat een uitgebreidere instructieset zorgt voor een snellere uitvoering van berekeningen [23]. Dit is omdat er bij een grotere instructieset complexere instructies zijn die ook in een enkele klokslag uitgevoerd kunnen worden. Er zijn dan dus minder instructies nodig om een bepaalde berekening uit te voeren en dat zorgt voor een sneller systeem. Een grotere instructieset zorgt wel voor een complexere microprocessor en dit kan nadelen opleveren in bijvoorbeeld het energie verbruik, ontwikkelkosten of aanschafkosten. Er zijn veel verschillende instructiesets die

gebruikt worden voor microcontrollers. De meest gebruikte en best ondersteunde instructiesets zijn ARM, AVR en PIC. Deze zullen nader onderzocht worden.

ARM

De ARM instructieset is ontwikkeld door het Britse Acorn Computers Ltd. Het is in 1984 begonnen als instructieset voor de eerste computers, maar de laatste twee decenia is het zich meer gaan richten op toepassing in PDA's en andere draagbare elektronica. Hierdoor is de ontwikkeling voorgericht op energiezuinige microprocessoren. In microcontrollers wordt de ARM Cortex-Mx serie toegepast. Deze bevatten de ARMv6-M, ARMv7-M of ARMv7E-M instructieset. Een overzicht van de verschillen is te zien in tabel 3.3. Het voordeel van ARM is dat het een 32 bits instructieset is, waardoor deze zeer uitgebreid is. Hierdoor zijn onder andere speciale instructies voor digitale signaal bewerkingen beschikbaar welke handig kunnen zijn bij de analyse van meetgegevens.

ARM Cortex-M	**Thumb-2**	**Hardware multiply**	**Hardware divide**	**Saturated math**	**DSP extensions**	**Floating-Point Unit (FPU)**	**ARM architecture**
Cortex-M0	Subset	1 or 32 cycle	No	No	No	No	ARMv6-M
Cortex-M0+	Subset	1 or 32 cycle	No	No	No	No	ARMv6-M
Cortex-M1	Subset	3 or 33 cycle	No	No	No	No	ARMv6-M
Cortex-M3	Entire	1 cycle	Yes	Yes	No	No	ARMv7-M
Cortex-M4	Entire	1 cycle	Yes	Yes	Yes	Optional, SP	ARMv7E-M
Cortex-M7	Entire	1 cycle	Yes	Yes	Yes	Optional, SP, or SP & DP	ARMv7E-M

Table 3.3: Vergelijking Cortex-M microprocessors.

AVR

De AVR instructieset is door twee studenten, A. Bogen en V. Wollan, aan het Norwegion Institute of Technology in 1996 bedacht. In een vroeg stadium is de technologie doorverkocht aan Atmel. Atmel is momenteel nog steeds verantwoordelijk voor de ontwikkeling van AVR. Het is een 8 bits architectuur, dit betekent dat het een erg eenvoudige, maar dus ook gelimiteerde instructieset heeft. Voor veel eenvoudige toepassingen is AVR zeer geschikt. Het grote voordeel van AVR is dat er veel informatie vrij toegankelijk is en dat er een groot aanbod van gratis ontwikkel software aanwezig is.

PIC

De PIC instructiesets zijn gemaakt door Microchip Technology. De instructieset in een PIC varieert van 35 instructies in de simpelste versies tot 80 instructies in de meest geavanceerde PIC's. De instructieset van een PIC is dus erg klein. De mogelijkheden van een PIC microprocessor komen heel erg overeen met de mogelijkheden van een AVR microprocessor. Met beide systemen kan het zelfde bereikt worden met ongeveer de zelfde prestaties. De beste keuze tussen beiden is dan ook zeer applicatie afhankelijk en het verschil in prestatie zal miniem zijn. Over beide instructiesets is veel informatie te vinden. Een keuze tussen AVR en PIC zal dus al snel gebaseerd worden op ervaring met het platform en beschikbaarheid van de componenten.

3.6.3. RANDAPPARATUUR

De microcontroller moet over voldoende extra randapparatuur beschikken zodat het eenvoudig verbinding kan maken met andere onderdelen in het systeem. Hierbij moet er rekening mee gehouden worden dat er in toekomstige versies extra sensoren, actuatoren of andere onderdelen aan de GonioTrainer toegevoegd gaan worden. Het is dus gewenst dat er voldoende ruimte tot uitbreiding is zonder dat daarvoor weer een nieuwe microcontroller nodig is. Er moet dus niet alleen gekeken worden naar wat er op dit moment nodig is, maar ook naar wat veel gebruikte randapparaten zijn.

SERIAL PERIPHERAL INTERFACE

De Serial Peripheral Interface (SPI) is een interface die seriële communicatie tussen onderdelen mogelijk maakt. Er zijn veel sensoren die via SPI aangestuurd en uitgelezen kunnen worden. Ook andere componenten zoals een bijvoorbeeld een Bluetooth module kunnen via SPI aangestuurd worden. Daarnaast worden opslagmedia, zoals flash chips en SD kaarten, ook veelal aangestuurd via SPI.

Via SPI kan een full-duplex communicatie opgesteld worden. Dit houdt in dat het mogelijk is om tegelijk data te verzenden en te ontvangen. SPI is een bus, dus er kunnen een of meerdere componenten aan dezelfde SPI interface gekoppeld worden.

SPI is geen standaard die wordt beheerd door een groep of een bedrijf. Daardoor zijn er verschillende varianten. De klokpolatiteit kan verschillen en de klokfase kan verschillen. Hierdoor zijn er in totaal 4 verschillende versies mogelijk. De meeste microcontrollers bieden de mogelijkheid om de SPI interface te configureren, dus alles kan aangesloten worden op een microcontroller. Het is alleen niet mogelijk om meerdere apparaten op de zelfde bus aan te sluiten als de instellingen voor de polariteit en de fase anders zijn. Het is daarom wenselijk dat er meerdere SPI interfaces op de microcontroller zitten.

INTER INTEGRATED CIRCUIT COMMUNICATIONS

Inter Integrated Circuit Communications(I^2C) is een interface die data verstuurd tussen I^2C apparaten over 2 draden, gezamenlijk aardpunt draad niet meegenomen. Een I^2C interface wordt gebruikt wanneer eenvoud en lage productie kosten belangrijker zijn dan snelheid. I^2C wordt gebruikt in langzame digitaal/analoog naar analoog/digitaal omvormers, volume regeling in speakers, besturen van LCD schermen en aan/uit zetten van stroomvoorzieningen.

I^2C gebruikt twee bi-directional draden wat betekent dat elke apparaat de verbindingen kan beheersen. De communicatie verloopt door middel van het versturen en ontvangen van bevestigingssignalen.

Elke I^2C apparaat word gekenmerkt door een uniek adres. Het is niet mogelijk om dezelfde apparaten op één microcontroller aan te sluiten. Aangezien elke iedereen over de verbindingen kan praten kan een defect apparaat de lijnen bezet houden, waardoor de communicatie vast loopt. Er is een externe reset nodig om dit probleem op te lossen

UNIVERSAL ASYNCHRONOUS RECEIVER AND TRANSMITTER

Een universal asynchronous receiver and transmitter is een seriële interface om data uit te wisselen. Het is een interface die gericht is op communicatie tussen twee apparaten. Het is dus niet mogelijk om een bus te vormen en met meerdere apparaten over dezelfde UART te praten. Met een UART is het wel mogelijk om een full-duplex verbinding op te zetten. Een apparaat dat via UART communiceert kan dus tegelijk zenden en ontvangen. UART wordt over het algemeen alleen gebruikt voor communicatie tussen apparaten die niet op de zelfde PCB zitten. Dit omdat er maar twee draden nodig zijn voor de communicatie. Omdat er altijd maar twee apparaten met UART communiceren is er geen slave of master, maar zijn beide apparaten gelijkwaardig. Wel moeten beide apparaten weten welke doorvoersnelheid (baud) er gebruikt wordt.

3.6.4. KEUZE

De keuze van de juiste microcontroller is afhankelijk van een groot aantal criteria. Zo moeten er zo veel mogelijk randapparaten in zitten, moet die energie zuinig zijn, geavanceerde berekeningen binnen een beperkte tijd kunnen doen, klein in formaat zijn en niet te duur zijn. Voor de GonioTrainer is een microcontroller nodig die over een ARM processor beschikt, omdat er complexe berekeningen op gedaan dienen te worden.

Om een keuze te maken voor een Cortex versie is er gekeken naar de eigenschappen van de verschillende versies. Op de website van ARM staat een overzicht van toepassingen van de verschillende Cortex versies [24]. Daar is te zien dat een Cortex M3 aangeraden wordt voor toepassingen in een "Activity tracker wearable". Aangezien de GonioTrainer goed past in de omschrijving Activity tracker wearable is de M3 waarschijnlijk een goede keuze. Het is handig dat er voor de data analyse de mogelijkheid is om de meetresultaten in het frequentiedomein te analyseren. Daarom is gekeken naar de snelheid waarmee een FFT uitgevoerd kan worden. De resultaten hiervan zijn te zien in tabel 3.4. Hier is te zien dat een M3 dit aanzienlijk sneller kan een M0 of een M0+. Met een kloksnelheid van $32MHz$ kan een 256 bits FFT in iets meer dan een milliseconde uitgevoerd worden. Aangezien de GonioTrainer zal meten met een frequentie van $100Hz$ is er met deze snelheid een goede marge ingebouwd om real-time analyse mogelijk te maken.

1024-FFT (Complex in Q15 Format)	
Cortex-M0	855 733 cycles
Cortex-M0+	664 531 cycles
Cortex-M3	204 244 cycles
Cortex-M4	89 839 cycles

256-FFT (Complex in Q15 Format)	
Cortex-M0	175 375 cycles
Cortex-M0+	136 296 cycles
Cortex-M3	41 430 cycles
Cortex-M4	18 480 cycles

Table 3.4: Aantal klokcycli dat nodig is voor een FFT.

Er is nu vast gesteld dat de processor van de microcontroller een ARM Cortex M3 moet zijn. Na het aanbod van een aantal fabrikanten bekeken te hebben bleek STMicroelectronics er gunstig te zijn. In figuur 3.10 is een overzicht te zien van ARM Cortex-M gebaseerde microcontrollers die aangeboden worden door STMicroelectronics. Dit is een serie met microcontrollers die allemaal gebaseerd zijn op ARM Cortex-Mx processoren. Daarop is te zien dat de STM32L1 serie een zeer energie zuinige M3 core bevat. Omdat een laag energie verbruik een van de eisen is, is deze serie microcontrollers zeer geschikt. Het blijkt ook dat er voor die microcontrollers ontwikkel borden aanwezig zijn die zeer scherp geprijsd zijn, minder dan € 10. Ook zijn er veel peripherals aanwezig in de microcontrollers. Verder is het mogelijk om via Eclipse een ontwikkel omgeving op te zetten die het mogelijk maakt om software te schrijven die voor ieder doeleinde gebruikt mag worden.

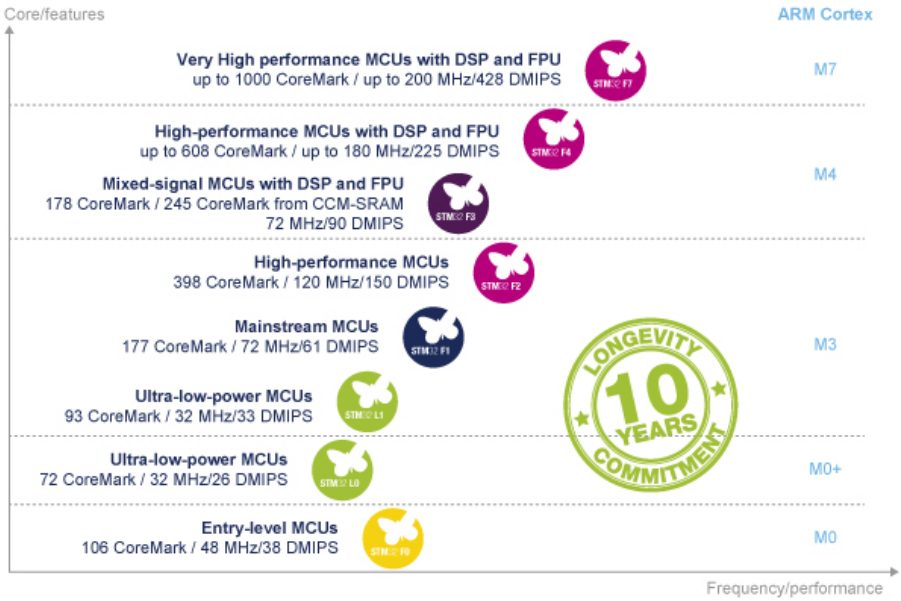


Figure 3.10: Vergelijking van stm32 ARM microcontrollers

3.6.5. CONCLUSIE

In de product keuze zijn veel verschillende controllers met elkaar vergeleken. Hier is gelet op het aantal randapparaten dat de controller beschikt, maar ook naar de aanwezigheid van ontwikkel borden. Uiteindelijk is selectie verkleint tot de STM32L152 serie. Hiervan is eigenlijk de STM32L152xD het meest geschikt. Deze microcontroller bevat namelijk een randapparaat dat speciaal ontwikkeld is voor het aansturen van SD kaarten. Hiermee kunnen SD kaarten veel sneller aangestuurd worden dan via SPI en op die manier blijft er een SPI poort vrij voor andere toepassingen. Helaas was deze niet verkrijgbaar op een ontwikkelbord, dus is gekozen voor de tegenhanger de STM32L152xC. Deze heeft precies de zelfde functionaliteit, alleen heeft deze geen SD interface. Het ontwikkelbord dat gekozen is is de STM32L152RC Discovery.

De trilmotor heeft ook een microcontroller nodig om te kunnen werken. Deze hoeft enkel de Bluetooth verbinding te beheren en de de trilmotor aan te sturen. Het is dus niet nodig dat hier een krachtige ARM processor in zit, maar een eenvoudige, kleine en energie zuinige microcontroller is voldoende. In paragraaf 4.1.5 wordt de Bluetooth module vast gesteld voor de trilmotor. Bij deze module zijn libraries geleverd voor

een Arduino. Arduino Uno is gebaseerd op een AVR microcontroller. Voor het prototype zal een Arduino Uno gebruikt worden, maar dit kan in de toekomst eenvoudig over gezet worden op een andere microcontroller die beter aan de eisen van formaat en energieverbruik voldoet. De attiny serie van Atmel is hier uitermate geschikt voor.

3.7. SYSTEEM OVERZICHT

Met behulp van het concept systeem zijn de volgende keuzes gemaakt over de onderdelen van het systeem. De Goniometer zal een magnetisch encoder zijn. De meetwaarden zullen worden opgeslagen op een SD-kaart van $2GB$. De microcontroller zal een STM32L152RC zijn. De Bluetooth module aan deze microcontroller zal de nRF51822 van Nordic zijn. De actuator zal een PiCoVibe van Precision Drive. Deze zal aan een Arduino Uno een Bluetooth module van Nordic nRF8001.

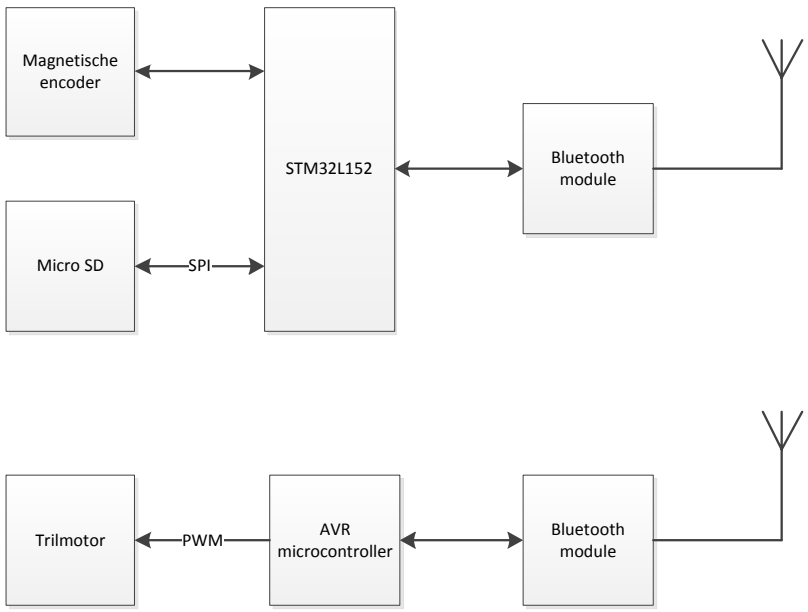


Figure 3.11: Een overzicht van de onderdelen van het concept systeem

4

IMPLEMENTATIE

4.1. COMMUNICATIE

In paragraaf 3.1 is onderzocht dat Bluetooth Low Energy de beste technologie is voor de draadloze verbinding. Met deze technologie kan de actuator aangestuurd worden, kunnen twee GonioTrainers gekoppeld worden en is het ook mogelijk om eenvoudig met een smartphone, computer of tablet te communiceren. Voor de huidige implementatie bestaat de verbinding uit twee delen. De eerste is de implementatie van de verbinding in de GonioTrainer. Dit deel moet in staat zijn om verbinding te leggen met de trilmotor, een andere GonioTrainer en in de toekomst met andere apparaten zoals een tablet. Het tweede deel bestaat uit de trilmotor welke alleen aan een enkele GonioTrainer gekoppeld moet worden. In dit verslag komt alleen de implementatie van de GonioTrainer met de actuator aan bod.

4.1.1. BLUETOOTH LE PROTOCOL STACK

Het Bluetooth LE Protocol is op te delen in verschillende onderdelen die in drie groepen ondergebracht kunnen worden. Dit is te zien in figuur 4.1. Deze verdeling van de onderdelen vertonen veel gelijkenissen met het OSI model [25]. De verdeling van de groepen wordt bepaalt door de plek waar de onderdelen geïmplementeerd worden.

De controller groep bestaat uit de onderdelen die in een Bluetooth module ingebouwd zijn. Een dergelijke module moet aangestuurd worden door een microcontroller die de Host groep implementeert. In de controller groep valt de Physical Layer. Dit onderdeel zorgt voor de fysieke overbrengen van de draadloze signalen en bestaat dus uit de antenne. Ook worden de signalen die binnenkomen via de antenne gedecodeerd tot digitale informatie. Deze informatie wordt vervolgens verwerkt via de Link Layer. Hier worden informatiepakketten voorzien van de benodigde headers. Dit onderdeel voert ook controles uit om te kijken of data juist ontvangen of verstuurd is. Als dit niet het geval blijkt zorgt dit onderdeel er voor dat een verzending over gedaan wordt. Als laatste is er een Direct Test Mode in deze groep geïmplementeerd. Dit onderdeel maakt het mogelijk om verschillende delen van de module te testen en te configureren. Zo kan de antenne versterking hier gecontroleerd worden. Ook zaken zoals de modulatie eigenschappen en de zend frequentie kunnen getest worden. Alle informatie, zowel van de Direct Test Mode als van de Link Layer moet met de microcontroller gedeeld kunnen worden. Ook moet de Bluetooth module commando's van de microcontroller kunnen ontvangen. Hiervoor is de Host Controller Interface. Deze zorgt voor een solide connectie tussen de host (de microcontroller) en de controller (de Bluetooth module). De implementatie hiervan verschilt veel per module. Sommige modules worden aangestuurd via SPI of UART. Bij andere kan het zijn dat de Host en de Controller op een enkele chip geïmplemeteerd zijn en dat deze connectie op dus een heel laag niveau plaats vindt.

De tweede groep is de Host groep. Dit is een groep protocollen dat in de microcontroller die de module aansturen en geïmplementeerd moeten worden. Deze onderdelen zijn dus puur software. Het Logical link control and adaptation protocol (L2CAP) zorgt ervoor dat de link tussen de module en de rest van de software goed verloopt. Dit protocol maakt het mogelijk dat er meerdere Bluetooth connecties op het zelfde apparaat mogelijk zijn. Het is namelijk de taak van het L2CAP om de pakketten afkomstig van verschillende bronnen

te onderscheiden zijn en op de juiste manier aan hogere lagen door te geven. Ook worden grote hoeveelheden data hier opgesplitst in kleinere pakketten of juist weer in elkaar gezet. De Security manager is een optioneel onderdeel. Als de het gewenst is dat de verbinding gecodeerd verloopt omdat er gevoelige informatie verzonden wordt kan de Security Manager geconfigureerd worden om alle informatie versleuteld te versturen. Dit onderdeel zorgt voor het creeëren van sleutels en voor de encryptie en decryptie van pakketten. Naast de security manager loopt het Attribute Protocol. Dit protocol probeert services te vinden in de pakketten die binnen komen via L2CAP. Een service is een vooraf gedefinieerde set regels waarmee apparaten met elkaar kunnen communiceren. Op de zelfde manier heeft USB verschillende klassen waarmee de computer eenvoudig massa opslag apparaten kan onderscheiden van bijvoorbeeld een human input device (een toetsenbord of muis). Met de toevoeging van gestandariseerde services is het mogelijk om apparaten dat twee volledig onafhankelijke apparaten op een eenvoudige manier kunnen communiceren zonder dat er extra drivers nodig zijn. Het Attribute Protocol geeft de data die binnen komt door aan het juiste proces in het Generic Attribute Profile. Hierin worden alle services beheert en worden alle lees en schrijf acties uitgevoerd. Als laatste is er nog het Generic Access Profile. Dit is een protocol dat ervoor zorgt dat twee Bluetooth apparaten aan elkaar gekoppeld konnen worden. Hierin wordt de naam van het apparaat bepaald en ook een uniek adres wat er voor zorgt dat de pakketten bij het juiste apparaat aankomen.

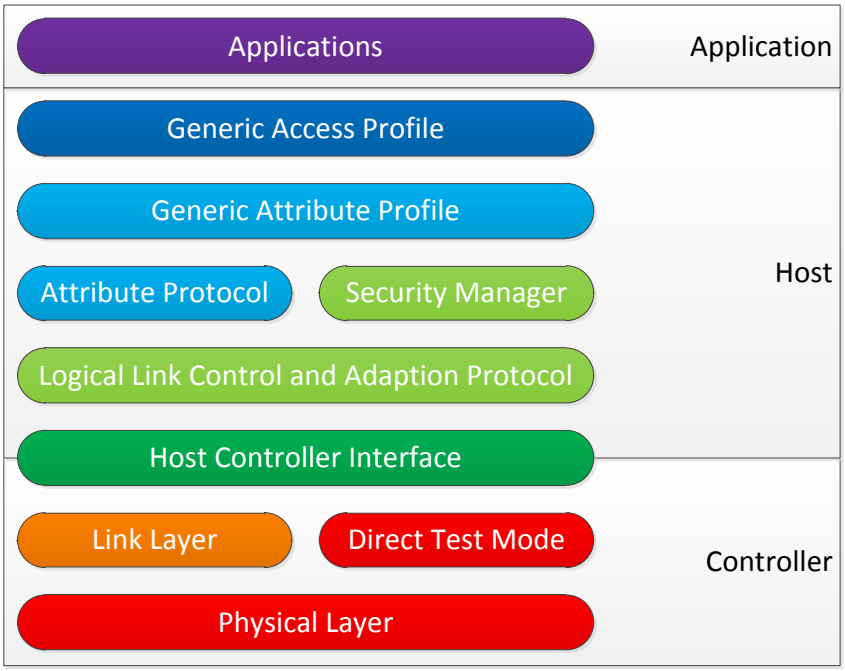


Figure 4.1: Overzicht van de Bluetooth LE protocol stack

4.1.2. HARDWARE

Er zijn verschillende manieren om een Bluetooth LE verbinding te implementeren. De meest gebruikte manier is om een Bluetooth controller te zoeken en deze aan te sturen met een externe microcontroller of, als de controller het toelaat, met interne systemen. Dit zijn zogeheten SoC's (System on a Chip). Hier zit een Bluetooth controller samen met een microcontroller verpakt in een enkele chip. Dan worden dus alle lagen van de Bluetooth Protocol Stack in door een enkel component uitgevoerd. De tegenhangers van de SoC's zijn de Bluetooth modules. Deze hebben alleen de controller protocollen geïmplemeteerd.

Op de markt zijn zeer veel verschillende modules en SoC's te vinden. Als deze nader bekeken worden zijn ze echter allemaal terug te leiden tot twee fabrikanten. De eerste is Texas Instruments, waarvan de CC254x serie Bluetooth controllers zijn. Aan deze chips hoeft enkel een antenne gekoppeld te worden om de Blue-

tooth functionaliteit compleet te maken. Er zijn veel modules te vinden die gebaseerd zijn op de CC254x serie van Texas Instruments. Het nadeel van deze chip is dat er alleen software voor geschreven kan worden met Keil Studio, een zeer dure development omgeving. Dit programma mag gratis gebruikt worden voor firmware die kleiner is dan 32 kB, maar zodra deze grens overschreden wordt moet er een licentie aangeschaft worden die op kan lopen tot enkele duizenden euro's.

De tweede fabrikant die Bluetooth controllers verkoopt is Nordic Semiconductors. Ze bieden zowel SoC's als controllers die een andere microcontroller nodig hebben aan. Veel Bluetooth modules die op de markt te vinden zijn hebben een Bluetooth controller van Nordic Semiconductors [26]. Nordic heeft een eigen Software Development Kit(SDK), die gratis is voor iedereen die een Nordic product aanschaft. Deze SDK kan geïntergreerd worden in Eclipse, welke gedistribueerd wordt onder een GNU General Public License. Met de Bluetooth controllers van Nordic is het dus mogelijk om zonder licentiekosten software te schrijven die voor alle doeleinden gebruikt mag worden. Verder zijn er redelijk goedkope producten te vinden die het ontwikkelen van een product gebasseerd op Nordic Bluetooth Contollers mogelijk maken.

Het is belangrijk dat er van te voren vast staat welke rol een apparaat zal krijgen in het Bluetooth netwerk. Sommige controllers zijn namelijk niet in staat om elke rol te vervullen. De Bluetooth controller die aan de trilmotor gekoppeld zal worden hoeft alleen als slave gebruikt te worden. Deze controller moet maar met een enkele GonioTrainer (de master) kunnen koppelen. Verder hoeft deze controller geen verbindingen te beheren. De nRF8001 is een Bluetooth controller van Nordic Semiconducters welke enkel als slave gebruikt kan worden. In deze controller is de gehele Bluetooth Protocol stack geïmplemeteerd. Dat zorgt voor een zeer gebruiksvriendelijke implementatie. Er is echter wel een externe microcontroller nodig om deze aan te sturen. Er zijn bibliotheken beschikbaar die gebruikt kunnen worden om de externe microcontroller goed samen te laten werken met de nRF8001. De nRF8001 is speciaal ontwikkeld om zeer efficiënt in de slave rol te gebruiken. Dit is dus een goede keuze als Bluetooth controller voor de trilmotor. In het prototype wordt gebruik gemaakt van de MOD-nRF8001 van Olimex. Dit is een kleine PCB waar alle fundamentele hardware, zoals de klok en de antenne, op geïntegreerd zijn. Verder zijn de belangrijke pinnen van de controller makkelijk toegankelijk gemaakt zodat het mogelijk is om dit met eenvoudige gereedschappen te gebruiken.

De GonioTrainer heeft een wat geavanceerdere Bluetooth controller nodig. Deze moet als master kunnen fungeren. Dit betekent dat de Bluetooth controller connecties moet kunnen opzetten en beheren met meerdere apparaten. Hiervoor heeft Nordic de nRF51822 SoC. Dit is een Bluetooth controller met ingebouwde microcontroller. Met deze Bluetooth controller is alles mogelijk wat door de Bluetooth LE technologie ondersteund wordt. Ook is deze SoC goed verkrijgbaar in modules die gebruiksvriendelijk zijn voor het maken van prototypes. In het prototype wordt gebruikt gemaakt van de RBL nRF51822 van Red Bear Labs. Dit is RayTech MBT40 Bluetooth module die op een ontwikkelbord geplaatst is waardoor deze eenvoudig te programmeren is.

4.1.3. SOFTDEVICES

De nRF51822 chip van Nordic werkt met softdevices. Dit zijn stukken firmware die gebruikt worden door de Nordic SDK. De opbouw van een softdevice is te zien in figuur 4.2. Een softdevice implementeert de gehele Bluetooth Protocol Stack. Niet ieder softdevice kan de gehele functionaliteit van de Bluetooth Low Energy benutten. Hieronder zijn de softdevices uiteengezet.

S110

Dit is een softdevice dat alleen als slave kan werken. Als dit Softdevice op de nRF51822 geladen wordt heeft deze dus de zelfde mogelijkheden als en nRF8001.

S120

Dit softdevice implementeert de functionaliteit van een master die in een ster netwerk verbinding kan maken met 8 verschillende slave devices. Voor de GonioTrainer is dit gewenst. Dit geeft namelijk de mogelijkheid om de verbinding met de trilmotor te beheren en om in de toekomst met meerdere apparaten tegelijk te verbinden, zoals extra sensoren of actuatoren en andere GonioTrainers.

S130

Dit softdevice is een combinatie van S110 en S120. Als dit softdevice op de nRF51822 geladen is kan deze dus fingeren als slave of als master. Dit is ideaal voor de GonioTrainer, omdat op deze manier de function-

aliteit om twee GonioTrainers aan elkaar te koppelen makkelijk geïmplementeerd kan worden. Hiervoor dient namelijk een GonioTrainer als slave ingesteld te zijn. Dit is niet mogelijk met enkel een softdevice S120.

CONCLUSIE

De S130 softdevice is het meest geschikt voor de GonioTrainer als de wensen in toekomst perspectief meegenomen worden. Dit softdevice begeeft zich momenteel echter nog in de alpha versie. Dat betekent dat er nog geen definitieve versie van de firmware vrijgegeven is en de alpha versie nog fouten kan bevatten. De S120 is echter al zo ver ontwikkeld dat deze de status Production ready heeft gekregen. Dit softdevice is dus vrij van fouten. Daarom wordt voor de implementatie van de GonioTrainer nu S120 gebruikt. Het programma dat hiervoor geschreven wordt kan in de toekomst direct overgezet worden op softdevice S130.

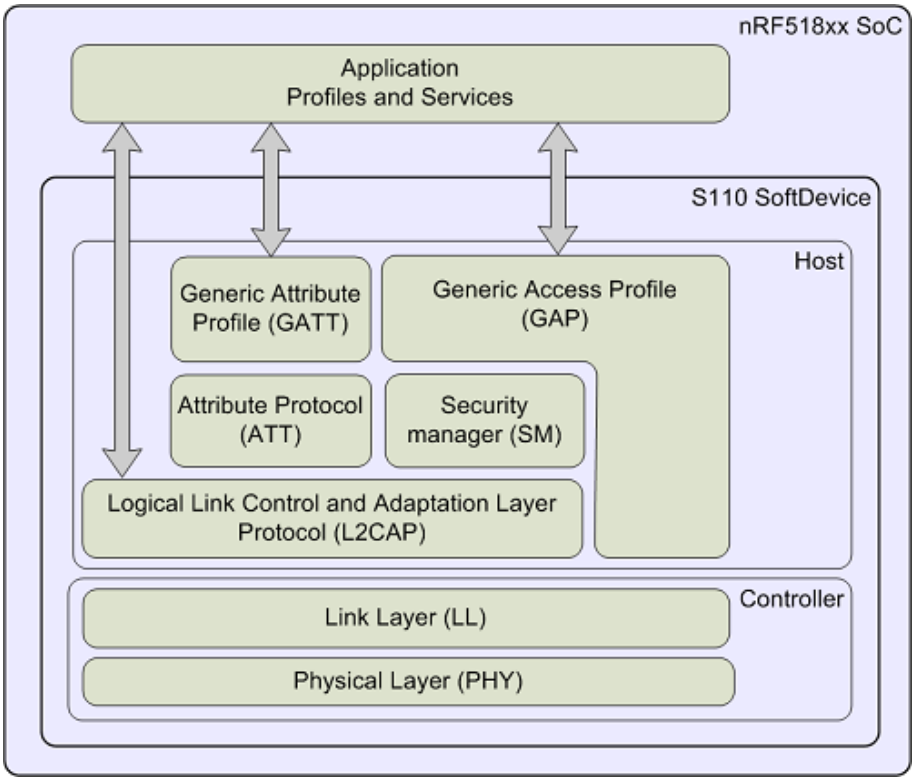


Figure 4.2: De integratie van de Bluetooth stack in de softdevices.

4.1.4. SERVICES

Zoals al eerder is gezegd maakt Bluetooth gebruik van profielen en services. Dit zijn gestandaardiseerde regels waarmee apparaten met elkaar kunnen communiceren. Zo wordt de compatibiliteit tussen de apparaten vergroot. In figuur 4.3 is een overzicht te zien van hoe een profiel is opgebouwd.

Een profiel is niet meer dan een container voor verschillende services. Het kan daarom een of meerdere services bevatten. Als voorbeeld nemen we het heart rate profile van de Bluetooth SIG. Dit profiel is speciaal ontwikkeld voor hartslag sensoren. Het profiel bestaat uit twee services. De eerste is de heart rate service en de tweede is de device information service. Deze services bestaan zelf weer uit verschillende karakteristieken. Zo heeft de heart rate service vier karakteristieken. Eentje voor de hartslag meting, eentje voor de positie van de hartslag sensor en nog twee andere. Op deze manier is de data heel gestuctureerd te benaderen. Een apparaat dat de positie van een Bluetooth hartslag sensor uit wil lezen weet dat hij moet zoeken naar de karakteristiek van de positie in de heart rate service. Elke karakteristiek en service heeft zijn eigen unieke code, de UUID genoemd. Aan de hand van deze code kunnen karakteristieken herkend worden.

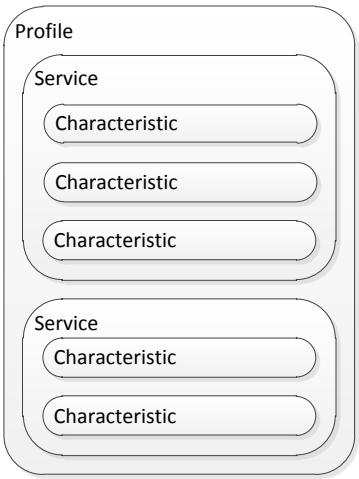


Figure 4.3: Een overzicht van de opbouw van een bluetooth profiel.

Er zijn veel gestandaardiseerde services voor Bluetooth LE. Deze services worden beheerd door de Bluetooth SIG. Hier vallen onder andere services onder die gebruikt kunnen worden voor hartslag sensoren. Helaas is nog geen officiële service beschikbaar die de karakteristieken bevat die aansluiten op de toepassing van de trilmotor in de GonioTrainer. Nordic heeft zelf een onofficiële UART service uitgebracht. Deze service maakt het mogelijk om twee apparaten over Bluetooth the laten communiceren alsof het een UART is. Hiermee is dus communicatie van en naar de trilmotor mogelijk en dat is precies wat er op dit moment nodig is. Het profiel bestaat uit de UART service en de Device information service. In de device information service staat de naam van het apparaat opgeslagen. In de Nordic UART Service zijn een RX characteristic en een TX characteristic aanwezig welke respectievelijk voor de inkomende en voor de uitgaande informatie zorgen. Dit is het punt waarop er goed opgelet moet worden. Bij een normale UART communicatie wordt de RX van de ene UART aangesloten op de TX van de andere UART. Bij de Bluetooth versie is dit iets anders. In de implementatie van de Bluetooth UART wordt gezocht naar de juiste karakteristieken en deze worden een op een aangesloten. De RX karakteristiek van de trilmotor zal dus gekoppeld worden aan de RX karakteristiek van de GonioTrainer. Een karakteristiek kan niet tegelijk een lees en een schrijf functie hebben. Bij een van de twee apparaten zal de RX karakteristiek dus dienen om informatie te verzenden. De conventie die hier is aangehouden is dat er altijd wordt gekeken vanuit de Bluetooth master. Wat voor de master binnenkomende informatie is wordt gestuurd over RX karakteristiek en wat de master terug zegt zal binnen komen over de TX karakteristiek.

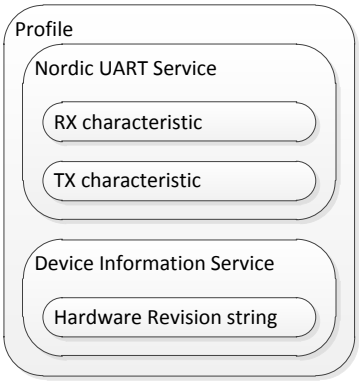


Figure 4.4: Het Bluetooth UART profiel.

4.1.5. IMPLEMENTATIE NRF8001

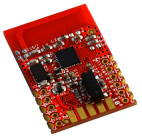


Figure 4.5: De Olimex MOD-nRF8001 Bluetooth module.

De nRF8001 wordt aangestuurd door een Arduino Uno. Hiervoor wordt gebruik gemaakt van de Bluetooth low energy SDK for Arduino. Als het systeem opstart begint deze met de advertising procedure. Dit houdt in dat het om de aantal millisecconde een bericht naar buiten stuurt dat het op zoek is naar een verbinding. Als de nRF51822 dit signaal opvangt zal deze automatisch met de nRF8001 verbinden. Tijdens het tot stand brengen van de verbinding wordt weer informatie uitgewisseld over de karakteristieken en de services. Mocht de verbinding niet succesvol opgezet worden dan begint de advertising opnieuw. Zodra er een succesvolle connectie is opgesteld zal het systeem in low power modus gaan zo lang er geen berichten klaar staan. Op het moment dat er wel een bericht klaar staat wordt er een callback aangeroepen. Deze functie bekijkt de waarde van het binnengekomen bericht en stuurt deze waarde via een PWM signaal naar de trilmotor. Na een halve seconde stopt de trilling weer en stuurt de nRF8001 het de zelfde waarde terug naar de nRF51822. De code die is gebruikt voor deze implementatie is terug te vinden in Appendix B.3. Deze code is als finite state machine te zien in figuur 4.6.

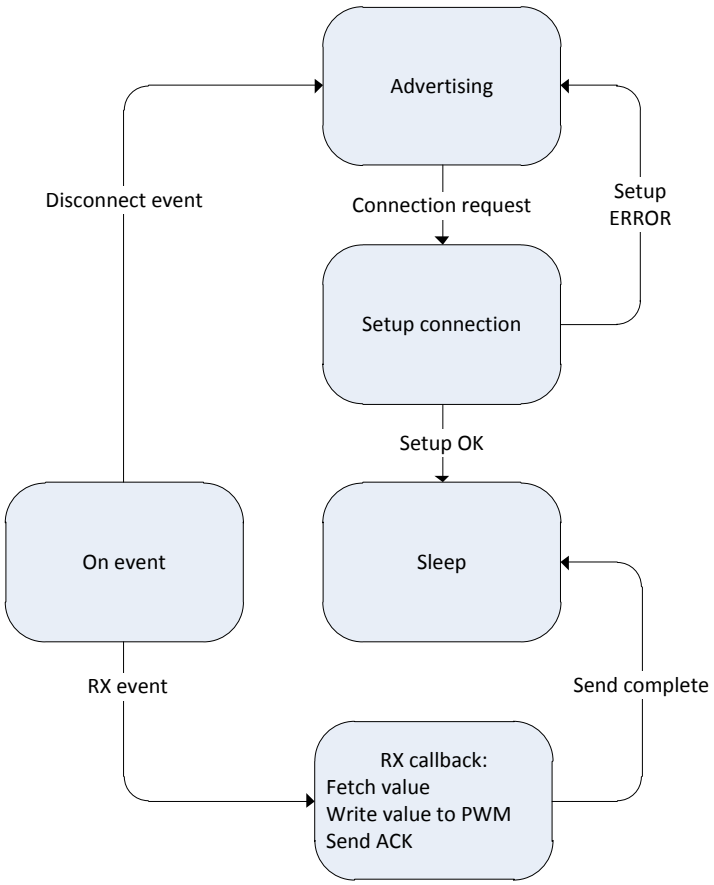


Figure 4.6: FSM van de code van de nRF8001.

4.1.6. IMPLEMENTATIE NRF51822

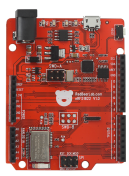


Figure 4.7: De RBL nRF518522 Bluetooth module.

De nRF51822 zal zodra deze opgestart is gaan zoeken of er apparaten in de buurt zijn waarmee deze kan verbinden. Dit wordt scannen genoemd. Tijdens het scannen checkt hij van alle gevonden apparaten het UUID van het profiel. Als dit overeenkomt met de UUID van het UART profiel dan wordt er een connectie gemaakt. nadat de connectie opgezet is gaat de nRF51822 in slaap stand. Op het moment dat er een trilling gegeven dient te worden krijgt de nRF51822 een signaal binnen van de

STM32L152RC. Dit gebeurd via een interrupt. Via de interrupt wordt dan een functie aangeroepen die een byte met daarin de trilsterkte van de motor naar de nRF8001 stuurt. Deze ontvangt dit signaal, stuurt de trilmotor aan en stuurt de zelfde byte terug als conformatie. De nRF51822 geeft dit weer door aan de STM32L152RC en gaat weer in slaap stand. De code die is gebruikt voor deze implementatie is terug te vinden in Appendix B.1. Deze code is als finite state machine te zien in figuur 4.8.

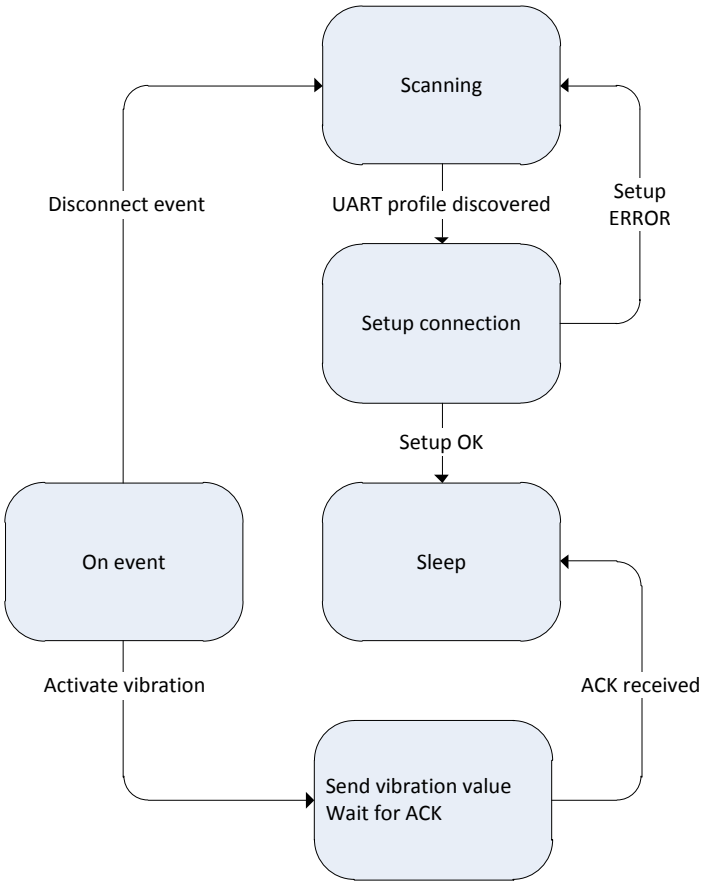


Figure 4.8: FSM van de code van de nRF51822.

4.1.7. CONCLUSIE

De Bluetooth verbinding is succesvol geïmplementeerd. Het is mogelijk om beide modules zelfstandig verbinding te laten maken en om data over en weer te sturen. Dit is nu gedaan met de onofficiële UART Service. Het is in de toekomst netter om een service op te stellen die speciaal gemaakt is voor de trilmotor. Ook zal in de toekomst over gestapt moeten worden op softdevice S130, zodat er connectie gemaakt kan worden tussen twee GonioTrainers. Voor deze connecties dienen dan ook aparte services opgezet te worden. De eis voor de verbinding was dat deze binnen $50ms$ de trilmotor aan kan sturen. In hoofdstuk 5 wordt een experiment gedaan waar aangettond wordt dat aan deze eis voldaan wordt. Ook wordt het stroomverbruik van de verbinding gemeten. Hier komt uit dat de nRF8001 $1mA$ gebruikt, wanneer de verbinding op zijn snelst is ingesteld. Het stroomverbruik van de nRF51822 is helaas niet te meten omdat deze chip geintergreed is op een ontwikkelbord en het enkel mogelijk is om het stroom verbruik van het gehele bord te meten. Dit bedraagt op dit moment gemiddeld 22 mA. De verwachting is dat het energieverbruik van de bluetooth module veel lager ligt omdat deze het energieverbruik voornamelijk door het verbindingsinterval bepaald wordt, en deze het zelfde verbindingsinterval heeft als de nRF8001.

4.2. GONIOMETER

In paragraaf 3.2 is onderzocht dat een magnetische encoder de beste sensor zal zijn voor de toepassing in de GonioTrainer. Er zijn twee magnetische encoders getest om te dienen als Goniometer. De gekozen encoders zijn de AS5055A en AS5600 van AMS. Het belangrijkste verschil tussen deze twee encoders ligt in de interface. De AS5055A maakt gebruik van SPI terwijl de AS5600 I^2C gebruikt om met de microcontroller te communiceren, nadere uitleg over deze twee interfaces is te lezen in paragraaf 3.6.3.

4.2.1. IMPLEMENTATIE AS5055A

De AS5055A is een magnetische positie sensor met een geïntegreerde Hall sensor, een 12 bits analoog naar digitaal omzetter en een slim vermogens management controller. De sensor heeft een afmeting van $4mm$ x $4mm$ x $0.85mm$, heeft een DC voedingsspanning tussen de 3.0 en $3.6V$ nodig en verbruikt maximaal $85mA$ onder normale werkings mode. De AS5055A kan alleen in slave mode werken met een leessnelheid van maximaal $500\mu s$.De meting van een hoek wordt door middel van vier draden SPI interface naar de STM32L152RC microcontroller verstuurt. Om de stabiliteit te vergroten van de communicatie over de SPI word er gebruikt gemaakt van een parity bit. Een parity of check bit is een methode om te controleren op fouten. De AS5055A encoder maakt gebruik van een even parity bit. Een data frame bestaat uit 16 bits: 14 bits data, 1 bit command frame error en 1 parity bit. Over de eerste 15 bits wordt er gekeken hoeveel bits een waarde 1 hebben. Bij een even aantal enen wordt de zestiende bit laag gezet wat betekent dat er geen fout is ontdekt. Bij een oneven aantal enen wordt de parity bit hoog gezet, wat betekent dat er een fout is ontstaan tijdens het lezen of versturen. Om een waarde te meten moet er een leescommando verstuurt worden naar de registe. Dit gebeurd met behulp van de MOSI pin (Master output Slave Input). De slave zal 1 CS periode(Slave Select) later via de MISO pin(Master Input Slave Output) de hoek waarde versturen. De hoekwaarde wordt uitgedrukt in 12 bits. Naast de 2 bits partity en command frame error worden er 2 bits gespendeerd aan alarm flag voor een te laag of groot magnetisch veld.

4.2.2. IMPLEMENTATIE AS5600

De AS5600 is een programmeerbare magnetische positie sensor met een 12 bit analoog of PWM uitgang. Het PWM signaal kan op vier snelheden worden gegeven: 115, 230, 460 en $920Hz$ Het is ook mogelijk om de hoek waarde uit een register te lezen met I^2C, dit duurt maximaal $150\mu s$. Met behulp van I^2C kan de default bereik van 0 naar 360°worden verkleint tot minimaal 18°. Wanneer er over een bereik van 0 tot 180°gemeten wordt, dan verdubbelt de resolutie zich. Over 360°zal de resolutie 0.087°zijn, terwijl over 180°een resolutie van 0.044°. De afmetingen van de sensor zijn $4.9mm$ x $3.9mm$ x $1.75mm$ en kan op een 5 of $3.3V$ spanning aangesloten worden. Hierbij zijn de grenzen voor de $3.3V$ voedingsspanning 3.0 tot $3.6V$. Hierbij verbruikt de AS5600 maximaal $6.5mA$. Voor de I^2C zijn er pull-up weerstanden nodig. De communicatie tussen sensor en microcontroller heeft geen fout detectie. I^2C maakt gebruik van bevestigingssignalen(ACK) om data te ontvangen en versturen.

4.2.3. TOEPASSEN

De keuze welke magnetische positie sensoren te implementeren is gevallen op de AS5055A. Volgens vele forums is een SPI interface eenvoudigere op te stellen. Wanneer dit gelukt zal zijn zou ook de SD-kaart kunnen worden geïmplementeerd. Helaas is het vertellen over de SPI eenvoudiger dan het toe passen. De eerste problemen kwamen bij het besturen van de CS(Slave Select). De SPI interface zou moeten beginnen wanneer de CS een dalende flank heeft. De SPI functies van de microcontroller leken niet te werken. De CS bleef constant hoog. Oplossing was de pinnen zelf te veranderen. Vervolgens zal er een byte door de MOSI verstuurt moeten worden van de microcontroller naar de slaaf. Dit signaal heeft nooit bestaan. Implementatie van de SPI interface op de STM32L152RC microcontroller is niet gelukt. De fout zal hoogst waarschijnlijk liggen in het initialiseren van de pinnen of in de bedrading en connecties.

Aangezien de problemen met de SPI niet opgelost konden worden is er tijdens het project besloten om de AS5600 met I^2C te implementeren. Er werden pull-up weerstanden en ontkoppel condensatoren gebruikt in het circuit zie figuur 4.9. Een pull-up weerstand is nodig om geen zwevende verbindingen te hebben. Een pull- up of down weerstand zorgt ervoor dat er geen kortsluiting tussen een spanningsbron en aardpunt ontstaat. Dit kan de sensor en microcontroller beschadigen en zelf kapot maken[27]. Ontkoppel condensatoren verminderen de ruis in de chip's stroomvoorziening. Hierbij is het van belang; hoe dichter bij de condensatoren bij de chip, hoe sneller condensatoren kunnen reageren tegen ruis fluctuaties op de stroomvoorzien-

ing van bijvoorbeeld piekstromen. Ook kunnen de capaciteit minder groot te zijn wanneer ze dichtbij geplaatst worden doordat er geen invloed is van draden[28].

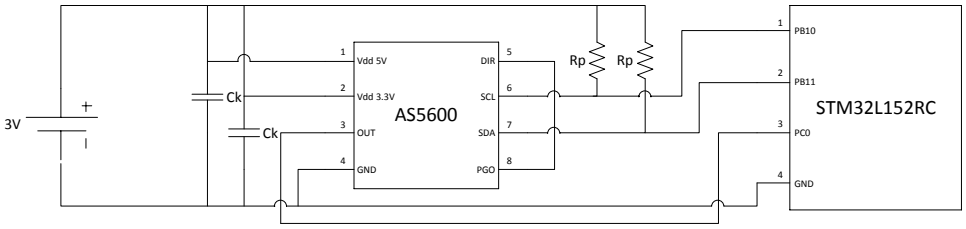


Figure 4.9: AS5600 sensor met de microcontroller, 100nF ontkoppel condensatoren en 4.7$k\Omega$ pull-up weerstanden

Om de hoek waarde te lezen moet de I^2C interface een START signaal vanuit de master generen. Vervolgens zal de 7 bits slave adres verstuurd worden met als achtste bit het read command. De adres met read command zal de volgende byte opleveren: 0110 1101. De slave zal een bevestiging van ontvangst moeten versturen op de negende klok periode. De bevestiging zal te zien zijn doordat de slave de SDA lijn omlaag trekt. Helaas liep het hierop vast, zie figuur 4.10. De slave stuurt geen bevestigingssignaal.

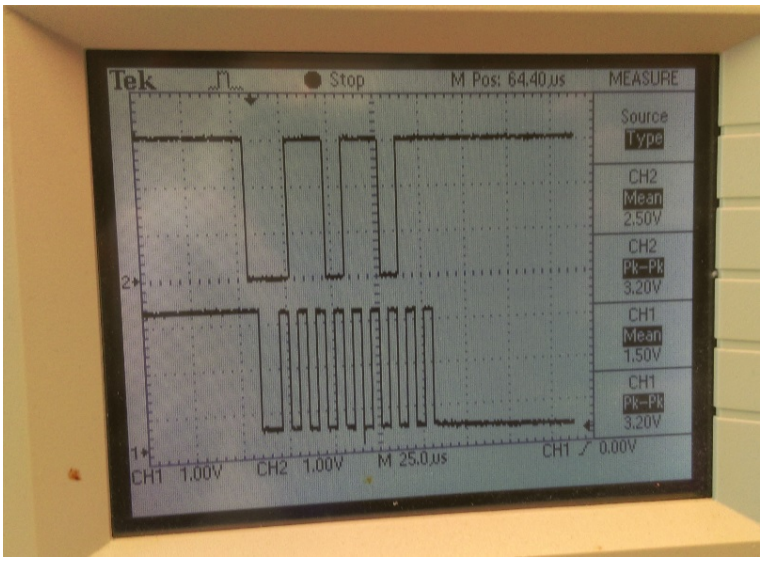


Figure 4.10: SDA en SCL lijnen. Het START signaal, 7bit adres en read bit worden door de meester verstuurd. De slave trekt de SDA niet omlaag

Een mogelijk probleem werd gevonden en opgelost. De STM32L152RC microcontroller kan op zijn pinnen en theoretische uitgangsspanningen van 3.0V geven. Dit is de minimale benodigde voedingsspanning voor de encoders! Metingen gaven waarden van 2.8 tot 2.9V. De magnetische encoders worden niet gevoed met hun minimale spanning. Een externe voedingsspanning van 3.3V werd er als ingangsspanning aangeboden. Helaas gaf de slave geen bevestiging. Er werd namelijk een nieuw probleem gecreeërd met de externe voeding. De encoder en de microcontroller hadden geen gezamenlijk aardpunt. Een extra verbinding tussen de aardpunten van de encode en microcontroller was nodig. Helaas bleek dit ook niet de oplossing van alle problemen te zijn. De hoek waarden kon niet uitgelezen worden met behulp van I^2C.

Het uitlezen van de hoek kon niet met SPI of I^2C, maar figuur 4.9 heeft een interessante pin genaamd OUT. In default mode kan een analoog signaal worden gelezen wanneer een magneet boven de encoders wordt gehouden. Door pin 8(Digital input) aan een aardpunt te bevestigen kan de spanning oplopen wanneer de magneet met de richting van de klok draait. Als hij aan een voedingsspanning wordt bevestigt zal de spanning oplopen bij verdraaiing in tegen gestelde richting van de klok. Het analoog signaal kan met een omvormer

van de STM32L152RC microcontroller omgezet worden om de hoek in graden uit te drukken, zie Appendix B.2 voor de C code. De tijd om een analoog naar digitaal om te vormen duurt maximaal 24.75µs en wordt vertaalt in 12 bits. Hiervoor heeft de microcontroller maximaal 100mA nodig. Afhankelijk van voedings- en referentie spanning van de microcontroller worden er typische fouten gemaakt van 1 bit met een gemiddelde waarde van $\pm$ 2 σ. De meet nauwkeurigheid zal dan een 0.5 bit zijn wat staat voor 0.044°.

4.2.4. CONCLUSIE

De meetfrequentie van 100Hz wordt gehaald door elke methode. De hoek uitlezen mag maximaal 10ms duren. De AS5055A duurt 500µs om met SPI de hoek versturen. Helaas is het toepassen van zowel de SPI als I^2C niet gelukt, maar er kan wel een hoek gemeten worden vanuit de analoge pin van de AS5600. Het stroomverbruik van de AS5600 en de microcontroller is 6.5mA en 100mA. De maximale stroomverbruik is 106.5mA.

Een ander gevolg van het mislukken van de implementatie van SPI is, dat de SD-kaart niet in het systeem wordt toegevoegd. De eis van leessnelheid wordt wel gehaald. Ook de resolutie van 0.1°wordt behaald. AS5055A en AS5600 hebben 12 bist voor 360°wat een resolutie van 0.087°geeft. De AS5600 bereik van 360°kan verkleint worden tot maximaal 18°, waardoor de resolutie verbetert kan worden. De omvormer van de microcontroller bestaat uit een 12 bits wat de zelfde resolutie geeft als de AS5055A en AS5600. Voor het analoog uitlezen is er een maximale meet nauwkeurigheid van 0.044°. De AS5600A kan in 150µs de hoek uitlezen en de microcontroller kan dit signaal omzetten in een digitaal signaal in maximaal 24.75µs. Het duurt maximaal 174.75µs om een analoog hoek te lezen.

4.3. ACTUATOR

In paragraaf 3.5 is geconcludeerd dat een pager het meest geschikt is om te dienen als actuator voor de GonioTrainer. Volgens paragraaf 3.4 kan een trilling door iedere persoon anders worden opgemerkt. Er zal een controller geïmplementeerd worden die de frequentie van de trilling kan afstellen.

4.3.1. IMPLEMENTATIE PAGER

De pager die gekozen is komt van Precision Microdrives en heeft als naam PicoVibe. De PicoVibe is een cilinder met een diameter van 8.8mm en een lengte van 24.9mm, zie Appendix A.2. Hij levert trillingen met een snelheid van 10.000 tot 16.000RPM en start binnen 23ms op. De PicoVibe heeft een nominale spanning van 3V, trekt gemiddeld 130mA en heeft een amplitude van 6G op zijn nominale spanning. Om de pager te laten trillen op een lagere sterkte zal de snelheid waarmee hij trilt moeten verlagen.

Om de frequentie aan te passen zal de motor sneller of langzamer aan/uit gezet moeten worden. Dit kan door middel van pulsbreedte modulatie(PWM). Door de duty cycle aan te passen kan je de frequentie eenvoudig beheersen. In paragraaf 4.1.5 wordt het gebruik van de Arduino vertelt. Een Arduino kan een PWM signaal geven, maar echter gelimiteerd. Pinnen van een Arduino Uno kunnen bij een 3.3V spanning maximaal 50mA aan. Volgens de datasheet van de PicoVibe zal deze grens worden overschreden bij een frequentie van 80Hz op 1.8V.In plaats van de PWM direct aan de PicoVibe aan te bieden kan hij ook aan een schakelaar worden aangeboden. Deze schakelaar zal wel tegen stromen van 200mA moeten kunnen.

Zo'n schakelaar kan een MOSFET(Metal Oxide Semiconductor Field-Effect Transistor) zijn. De PicoVibe moet dan wel extern worden gevoed, zie figuur 4.11. De transistor kan bij een V_{gs} van 4.5V en T van 25°C een I_d van 2.5A aan en heeft een opstart tijd van maximaal 35ns Een MOSFET zal wel de stroom kunnen trekken. Om de transistor te bedienen als schakelaar moet hij afhankelijk van het PWM signaal hoog(verrijking) of laag(verarming) gezet worden, zie Appendix B.3 voor de C code. Een n-type MOSFET zal aanstaan wanneer de $V_{gs} > V_{th}$ hier zal de I_d maximaal zijn. Hij zal uitstaan bij $V_{gs} < V_{th}$ er zal nu geen stroom door de drain kunnen stromen. Doordat de spanning over de transistor en actuator door het PWM signaal snel hoog en laag wordt zijn er diodes nodig over deze componenten. De actuator heeft een inductieve load, waardoor hij kwetsbaar is tegen spikes(spanningspieken).

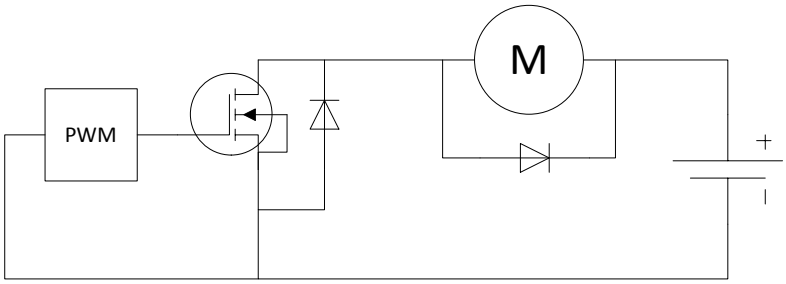


Figure 4.11: PWM controller die n-mosfet schakelt voor een actuator die gevoed wordt door een batterij

4.3.2. CONCLUSIE

Om een pager aan te passen in sterkte moet de frequentie verstelbaar zijn. Dit kan door de duty cycle van een PWM signaal. De microcontroller kan PWM signalen generen, maar geen stroomverbruik van meer dan 50mA aan. Door een n-mosfet te gebruiken als schakelaar kan de duty cycle worden toegepast om de trilmotor af te stellen. Het stroomverbruik is gemiddeld 130mA en heeft een opstartijd van 23ms. De maximale opstart tijd van de transistor bedraagt 35ns en kan ten opzichte van de pager tijd verwaarloost worden.

5

MEETRESULTATEN

5.1. COMMUNICATIE

Zoals al eerder is vermeld is het belangrijk dat de vertraging in de draadloze verbinding erg laag is. Op die manier is het eenvoudiger om een feedback algoritme te implementeren. De eis is dat communicatie maximaal 50ms bedraagt. Om te testen of aan deze eis voldaan wordt zijn er metingen gedaan aan de vertraging.

5.1.1. OPSTELLING

Voor de meting zijn de twee Bluetooth modules die met elkaar gaan communiceren 50cm van elkaar gescheiden. Deze afstand is gekozen omdat dit een realistische afstand is die de modules op het lichaam van elkaar gescheiden zullen zijn. De opstelling is te zien in figuur 5.1. Zodra beide modules AAN gaan maken ze een verbinding met elkaar te maken. Als deze verbinding tot stand gebracht is begint het meten van de tijd.

De nRF51822 is de master. Deze verzend een byte naar de nRF8001(de slave) op precies de zelfde manier zoals deze dit zal doen als er het nodig is om de trilmotor te laten trillen. Vlak vóór dat de functie aangeroepen wordt die byte zal verzenden wordt er een digitale output pin van laag naar hoog geschakeld. Deze pin is via een kabel verbonden met de Arduino. Op het moment dat er een overgang van laag naar hoog op de digitale pin gedetecteerd wordt zal er een interrupt procedure worden aangeroepen. In deze interrupt wordt het aantal milliseconde sinds het opstarten van de Arduino opgeslagen in een variabele. Er wordt verondersteld dat de vertraging hiervan t' verwaarlozen valt. Op het moment dat de byte daadwerkelijk ontvangen wordt door de nRF8001 wordt eerst het PWM signaal naar de trilmotor gestuurd. Nadar dit gebeurd is wordt er weer gekeken hoeveel milliseconde het geleden is dat de Arduino gestart is. Hier wordt de eerder verkregen waarde vanaf gehaald en dat levert het aantal milliseconde op dat het duurt om de trilmotor aan te zetten via een Bluetooth verbinding. Dit stuurt de nRF8001 vervolgens terug naar de nRF51822 en die geeft de waarde door aan de computer via een USB verbinding. vervolgens wordt er een seconde gewacht en dan wordt de volgende meting gedaan.

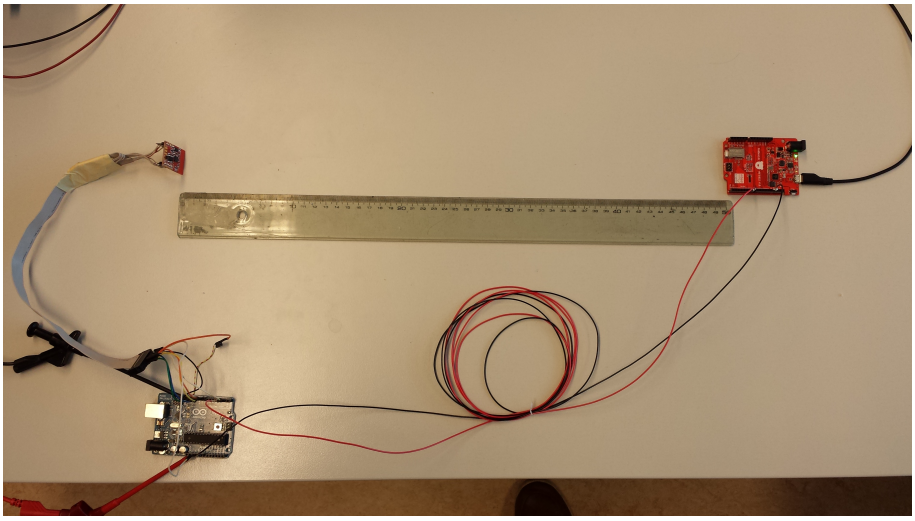


Figure 5.1: De meetopstelling voor het meten van de vertraging.

5.1.2. RESULTATEN

In het General Access Profile (GAP) kunnen instellingen voor de connectie geconfigureerd worden. Hier kan onder andere het minimale connectie interval(min) en het maximale connectie interval(max) ingesteld worden. Deze instellingen bepalen hoe lang het duurt voordat aan elkaar gekoppelde apparaten controleren of er een nieuw bericht is. Des te langer dit interval duurt, des te energie zuiniger de verbinding is. In de figuren 5.2, 5.3 en 5.4 zijn de resultaten te zien van verschillende instellingen.

Gemiddelde vertraging: 22,29 ms

Vertraging(ms)	*Frequentie*
0	0
4	0
8	0
12	70
16	16
20	0
24	86
28	0
32	78
36	7
40	0
44	0
48	0
52	1
Meer	2

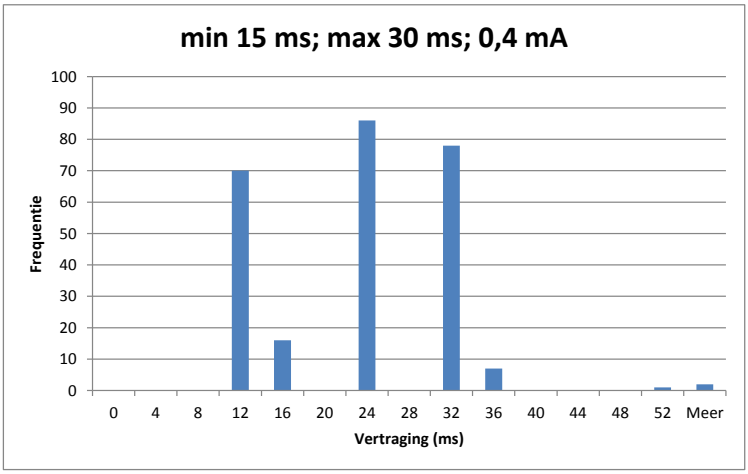


Figure 5.2: Meetresultaten voor de instellingen: min 15; max 30;

Gemiddelde vertraging: 14,69 ms

Vertraging	*Frequentie*
0	0
4	56
8	118
12	161
16	130
20	140
24	141
28	37
32	4
36	3
40	3
44	2
48	0
52	0
Meer	0

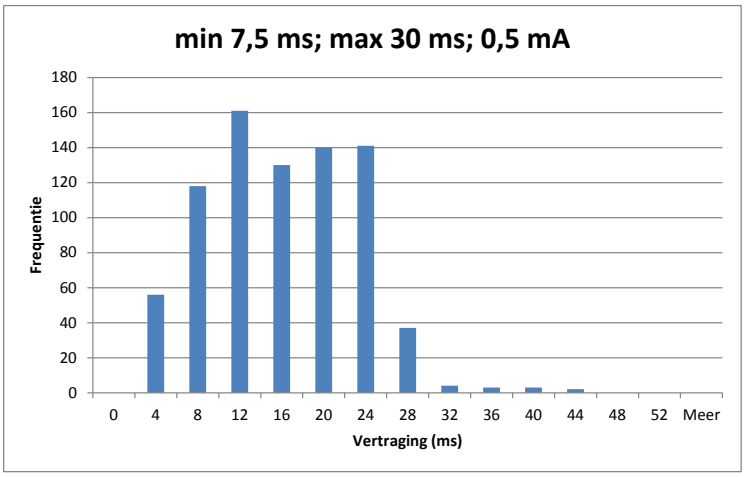


Figure 5.3: Meetresultaten voor de instellingen: min 7,5; max 30;

Gemiddelde vertraging: 11,94 ms

Vertraging	*Frequentie*
0	0
4	0
8	0
12	1558
16	310
20	1
24	16
28	0
32	0
36	0
40	0
44	0
48	0
52	0
Meer	0

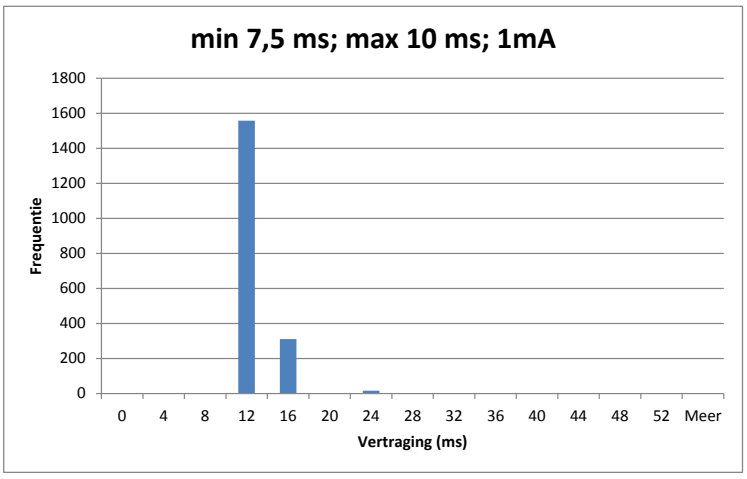


Figure 5.4: Meetresultaten voor de instellingen: min 7,5; max 10;

5.1.3. DISCUSSIE

Uit de resultaten wordt duidelijk dat er een afweging gedaan dient te worden. De connectie kan sneller gemaakt worden, maar dit betekent wel dat er meer energie verbruikt wordt. De eis die gesteld is aan de verbinding is dat de trilmotor binnen $50ms$ aangestuurd moet worden. De trilmotor zelf doet er $23ms$ over om op te starten. Het signaal moet dus binnen $27ms$ overgebracht zijn. Gemiddeld gezien liggen alle resultaten onder de $30ms$. Bij de energie zuinigste variant, waarvan de resultaten in figuur 5.2 te zien zijn, haalt 34% van de pakketten deze grens niet. De kans dat een signaal hier te laat komt is dus 34%. Bij de tweede variant, die $1mA$ meer aan stroom verbruikt is er echter maar 1,6% van de pakketten die de gestelde norm niet haalt. Dit is een enorme vooruitgang voor maar een beetje extra vermogen. In de laatste configuratie haalden 100% van de pakketten de norm. Er is in het programma van eisen niet gesproken over een percentage van pakketten dat later aan mag komen dan de eis van $50ms$. Daarom is voor nu de laatste configuratie de beste. Hoewel deze twee keer meer stroom verbruikt is het nog steeds een zeer zuinig apparaat. Een gemiddelde CR2032 knoopcel heeft een capaciteit van $200mA$h. Dit betekent dat de verbinding 200 uur mee kan op een

knoopcel wat ver onder de eis van 8 uur valt. Voor de huidige toepassing zal dit dus de beste keuze zijn.

5.1.4. CONCLUSIE

Het is gelukt om de vertraging van de Bluetooth verbinden binnen de $50ms$ te houden. Afhankelijk van het gewenste stroomverbruik is een vertraging van $11ms$ heel goed haalbaar. Bij deze instellingen verbruikt de Bluetooth module ongeveer $1mA$ bij $3V$. Dit betekent dat de microcontroller gemiddeld $24mA$ mag gebruiken om een minimale levensduur van 8 uur op een knoopcel te realiseren. Aangezien de gehele Arduino dit stroomverbruik had moet dit haalbaar zijn.

6

AANBEVELINGEN

In dit project is een concept gemaakt voor een systeem dat de GonioTrainer in zijn geheel implementeert. Daarnaast is de draadloze verbinding met de trilmotor als prototype uitgewerkt. Uit dit proces en de reflectie hierop zijn een paar aanbevelingen voortgekomen.

Op dit moment wordt voor de verbinding gebruik gemaakt van de Nordic UART Service. Dit is een onofficiële Bluetooth service die gedefineerd is door het ontwikkel team van Nordic Semiconductors. Het is netter om een service te ontwikkelen speciaal voor de trilmotor van de GonioTrainer. Nu is het namelijk zo dat de GonioTrainer automatisch verbinding maakt met apparaten die het UART profiel dragen. Dit hoeft niet per se een trilmotor te zijn. Als er een speciaal trilmotor profiel is wordt dit voorkomen en is er zekerheid dat een apparaat waarmee de GonioTrainer verbindt altijd juist reageert op de instructies die toegezonden worden.

De nRF8001 module wordt momenteel aangestuurd door een Arduino Uno. Dit is gedaan om de ontwikkeltijd voor het prototype haalbaar te maken. Het is uit energie en ruimte opzicht beter om deze microcontroller te vervangen door een microcontroller uit de Atmel AVR attiny serie. Deze serie microcontrollers is geoptimaliseerd voor een laag energie verbruik en een een klein formaat. Omdat het ook een AVR microcontroller is, is het eenvoudig om de code die gebruikt is voor de Arduino over te zetten op de attiny.

De nRF51822 module wordt nu aangestuurd met een enkele pin. Dit omdat er nog maar één commando is. Zodra er meer functionaliteit in de bluetooth verbinding komt zijn er meer commando's nodig. Daarom zal de koppeling tussen de STM32L152RC anders moeten. Als een UART of SPI interface gebruikt wordt dan kunnen er veel meer commando's gestuurd worden tussen de twee apparaten.

Momenteel wordt gebruik gemaakt van de STM32L152RC, omdat deze eenvoudig en goedkoop verkrijgbaar was op een ontwikkel bord. Als de GonioTrainer verder ontwikkeld wordt is het aan te raden om over te stappen op een STM32L152RD microcontroller. Deze heeft een speciale interface voor het aansturen van SD-kaarten. Dan kan de verbinding met de SD kaart sneller en blijft er een SPI interface vrij voor verbindingen met eventueel andere componenten.

De PicoVibe trilmotor heeft aan het eind van het project een opvolger gekregen. Deze pager heeft een $1mm$ kleinere diameter, kan $300RPM$ sneller trillen, gebruikt bij nominale spanning en belasting $30mA$ minder en geeft ook $1G$ grotere amplitude. De vermindering van het stroomverbruik maakt de opvolger vooral interessant voor de GonioTrainer. De GonioTrainer zal langer mee kunnen gaan ten opzichte van de gebruikte PicoVibe.

7

ETHISCHE ASPECTEN

Het gebruik van de GonioTrainer kan op basis van ethische aspecten worden overwogen. Vanuit de aspectenleer kan de GonioTrainer systematisch worden beoordeeld te beginnen met het economische aspect. De GonioTrainer dient om je sport prestaties te verbeteren. Er zullen sporters zijn die niet in staat zijn om de GonioTrainer te kopen vanwege de hoogte van de prijs. Er zal oneerlijke concurrentie ontstaan tussen de sporters die de GonioTrainer wel en niet kopen. Is het daarom eerlijk om de GonioTrainer te verbieden of gratis weg te geven, zodat iedereen er gebruik van kan maken?

Het juridische aspect gaat over regels en wetten leggen over het gebruik van sport verbeterende middelen. Zo is het innemen van bepaalde stoffen die je sport prestaties verhogen(doping) verboden. Ook sommige zwempakken zijn verboden om te gebruiken tijdens wedstrijden. Op dit moment zijn er geen regels bekend omtrent het dragen van elektronica aan het lichaam tijdens schaats wedstrijden. De vraag blijft of er regels nodig zijn voor apparaten als de GonioTrainer. Een schaatser gaat door de GonioTrainer niet direct sneller schaatsen, maar laat alleen weten wanneer hij of niet diep genoeg door de knieën buigt. Echter door deze kennis kan de schaatser anticiperen en weer dieper zakken wat wel weer leidt naar een snellere rondetijd. Dit kan een argument zijn om aan te tonen dat het apparaat je sport prestaties oneerlijk verbetert. Echter de informatie over de houding kan ook geschreeuwd worden door een coach die langs de baan staat. Moet de coach tijdens wedstrijden ook worden verboden?

Als laatst kan er vanuit het formatieve aspect gekeken worden naar de GonioTrainer. De techniek laat mensen denken dat de werkelijkheid bewerkt moet worden. Wanneer je niet goed kan schaatsen, dan moet techniek gebruikt worden om de schaats prestaties te verbeteren. Bij het schaatsen is al eerder gekeken naar verbeteringen door middel van techniek. Als je klapschaatsen gebruikt kan je sneller. Het gevolg is dat nu iedereen deze schaatsen gebruikt. De GonioTrainer kan de volgende technische uitvinding zijn waardoor schaatsers nog sneller kunnen schaatsen. Moet je techniek steeds gebruiken om sneller te schaatsen?

Naar onze mening kan je het verbeteren van de werkelijkheid, in dit geval schaats prestaties, niet stoppen. Het bewerken van de werkelijkheid om ons welzijn te verbeteren is iets menselijk. Het blijft wel goed om na te denken of sommige verbetering echt nuttig zijn.

8

CONCLUSIE

De opdracht was om het bestaande ontwerp van de GonioTrainer te vernieuwen een draadloze feedback module toe te voegen. Hierbij zijn eisen gesteld die terug te vinden zijn in hoofdstuk 2.

Het meten van de kniehoek is toegepast in de GonioTrainer. Bij verdraaiing van de hoek wordt er een analoog signaal vanuit de magnetische encoder AS5600 omgezet naar een bruikbaar digitaal signaal door de microcontroller STM32L152RC. De GonioTrainer kan binnen maximaal 174.75µs een hoek uitlezen en digitaal aanbieden aan de communicatie. De resolutie van 0.1°is behaalt met een waarde van 0.087°. Het stroomverbruik is maximaal 106.5mA. De draadloze aansturing van de trilmotor is succesvol geïmplementeerd. Afhankelijk van het gewenste stroomverbruik is een vertraging van 11ms haalbaar. Bij deze instellingen verbruikt de Bluetooth module ongeveer 1mA. De trilmotor heeft een opstarttijd van 23ms en verbruikt 130mA

Het is niet gelukt om aan de eis te voldoen dat alle huidige functionaliteit behouden blijft. De SPI verbinding is niet werkend gekregen, waardoor de SD kaart niet aangestuurd kon worden. Het is met het huidige model dus niet mogelijk om metingen uit te voeren die achteraf geanalyseerd kunnen worden.
De onderdelen van de GonioTrainer is niet gezamenlijk getest op energie verbruik, maar de onderdelen zijn los van elkaar onderzocht. Het uitlezen van de kniehoek, versturen door de communicatie en het opstarten van de trilmotor zal maximaal 0.17+23+11= 34.175ms duren. De reactietijd van 50ms is gehaald. Het stroomverbruik zal 106.5+1+130= 237.5mA zijn voor de componenten bij elkaar opgeteld. Berekeningen met eventuele bronnen zijn niet behandeld in dit project.

De maximale afmetingen en gewicht kunnen pas bepaalt worden als de onderdelen van de prototype gebruikt worden op een enkele chip. De prototype bestaat uit componenten die eenvoudig te gebruiken waren om de functionaliteit te testen.

De onderdelen van de GonioTrainer zijn gekozen op basis van open licenties. Hierdoor hoefde er alleen betaalt te worden voor gebruikte componenten en niet voor software. De prototype bestaat uit ontwikkelbordjes waar de benodigde componenten op zitten. De kosten van de prototype zal niet gelijk zijn aan die van de GonioTrainer wanneer alleen het benodigde word gebruikt.

A

Appendices

A.1. DATASHEET RESOLVER

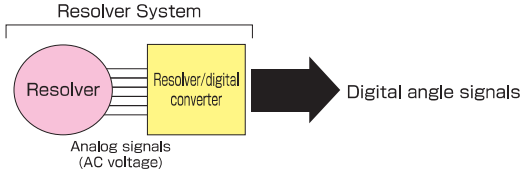
RESOLVERS

A resolver is an angle sensor that outputs rotational angles as two-phase AC voltages (analog signals). It features structurally high environmental resistance compared to other sensors due to its simple design comprised of only an iron core and a coil.
An AC output voltage is induced in the winding on the output side through excitation of the excitation winding using an AC voltage.
As this output voltage will vary depending on the angle of rotation, the angle of rotation can be calculated using the voltage reading.

◎ Features

- Wide operating temperature range
- Excellent environmental resistance
- Rotatable at high speeds
- High reliability
- Absolute value position detection
- Long-distance transmission available (Highly resistant to noise)
- Compact assembly

■Resolver System Structure

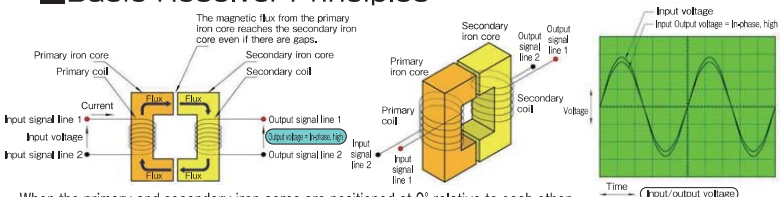


Comparison of Angle Sensors

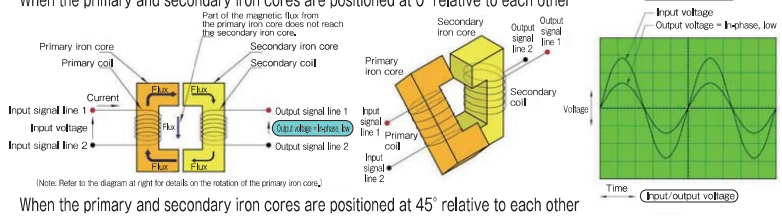
Item \ Type of sensor	Singlsyn® (VR series resolver)	Smartsyn® (Blushless resolver)	MRsensor	Hall effect sensor	Optical encoder	Potentiometer
Accuracy (Sensor alone)	△ 0.3° to 1°	○ 0.05° to 0.2°	△ 0.3° to 0.5°	△ 0.5° to 1°	◎ 0.02° to 0.1°	○ (Around 0.5% to 1%)
Resolution (Single-turn)	10-bit to 12-bit (In combination with RD) X (2, 3, 4-↑) (Up to 10000 pulses/r)	10-bit to 16-bit (In combination with RD) (Up to 65536 pulses/r)	— (Up to 4000 pulses/r)	— (Up to 500 pulses/r)	10-bit to 14-bit (Up to 5000 pulses/r)	(Equivalent to up to 14-bit)
Output signals	Absolute angle output A, B, Z UVW	Absolute angle output A, B, Z UVW	Incremental A, B	Incremental A, B	Incremental (Absolute angle) A, B, Z (UVW)	Absolute angle output Analog output
Rotational tracking speed	30000 min⁻¹ or above ◎	20000 min⁻¹ or above ○	30000 min⁻¹ or above ◎	30000 min⁻¹ or above ◎	10000 min⁻¹ or above ○	300 min⁻¹ or above ×
Shape/dimensions	◎	○	◎	◎	△	◎
Heat resistance	-55°C (-67°F) to +155°C (+311°F)	-55°C (-67°F) to +155°C (+311°F)	-10°C (+14°F) to 85°C (+185°F)	-10°C (+14°F) to 110°C (+230°F)	-10°C (+14°F) to 85°C (+185°F)	-10°C (+14°F) to 50°C (+122°F)
Anti-vibration	◎	◎	○	○	△	×
Reliability	◎	◎	○	△	○	×
Anti-noise	◎	◎	△	△	○	○
Price	◎	○	◎	◎	△	◎
Features	Absolute position detection High reliability Excellent environmental resistance High-speed rotation	Absolute position detection High reliability Excellent environmental resistance	Low resolution	Low resolution	High resolution but poor reliability	Contact type, finite angle

◎ : Excellent ○ : Good △ : Poor

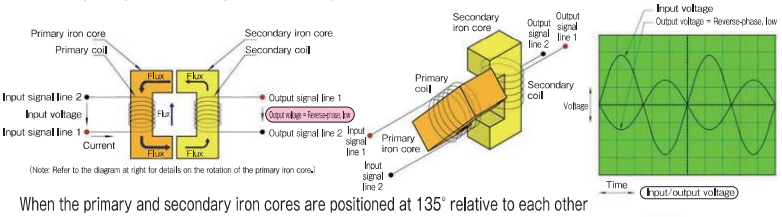
■Basic Resolver Principles



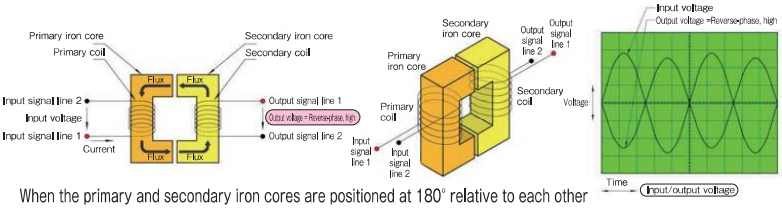
When the primary and secondary iron cores are positioned at 0° relative to each other



When the primary and secondary iron cores are positioned at 45° relative to each other



When the primary and secondary iron cores are positioned at 135° relative to each other

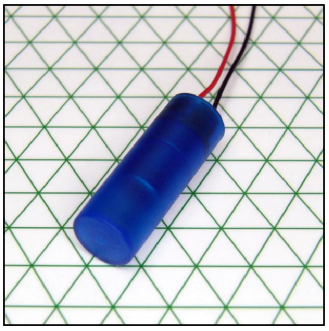


When the primary and secondary iron cores are positioned at 180° relative to each other

Figure A.1: Datasheet en werking van standaard resolvers

A.2. DATASHEET PICOVICE

307-100



9mm Vibration Motor - 25mm Type
Shown on 6mm Isometric Grid

PRECISION MICRODRIVES™

Product Data Sheet
Pico Vibe™
9mm Vibration Motor - 25mm Type

Model: 307-100

Ordering Information

The model number 307-100 fully defines the model, variant and additional features of the product. Please quote this number when ordering.
For stocked types, testing and evaluation samples can be ordered directly through our online store.

Datasheet Versions

It is our intention to provide our customers with the best information available to ensure the successful integration between our products and your application. Therefore, our publications will be updated and enhanced as improvements to the data and product updates are introduced.
To obtain the most up-to-date version of this datasheet, please visit our website at:
www.precisionmicrodrives.com
The version number of this datasheet can be found on the bottom left hand corner of any page of the datasheet and is referenced with an ascending R-number (e.g. R002 is newer than R001). Please contact us if you require a copy of the engineering change notice between revisions.

If you have any questions, suggestions or comments regarding this publication or need technical assistance, please contact us via email at:
enquiries@precisionmicrodrives.com or call us on +44 (0) 1932 252 482

Key Features

Body Diameter:	8.8 mm [+/- 0.2]
Body Length:	24.9 mm [+/- 0.3]
Rated Operating Voltage:	3 V
Rated Vibration Speed:	13,500 rpm [+/- 2,700]
Typical Rated Operating Current:	130 mA
Typical Normalised Amplitude:	6 G

Typical Vibration Motor Performance Characteristics

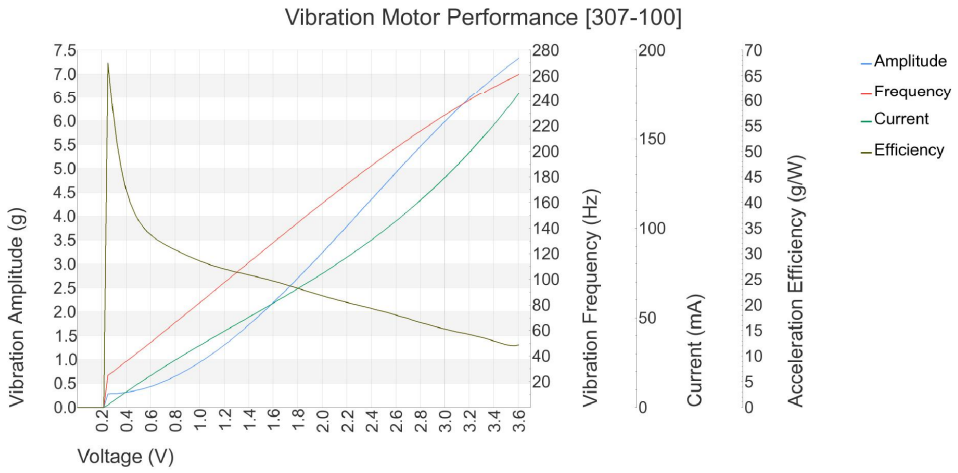


Figure A.2: Datasheet PicoVibe

B

C Code

B.1. Bluetooth

```
/* C code main voor de Bluetooth communicatie
* door Gert-Jan van Raamsdonk en Erwin Visser
* Januari 2015, EE3842 Bachelor Afstudeerproject */

#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include "nordic_common.h"
#include "nrf_sdm.h"
#include "ble.h"
#include "ble_db_discovery.h"
#include "softdevice_handler.h"
#include "app_util.h"
#include "app_error.h"
#include "ble_advdata_parser.h"
#include "boards.h"
#include "nrf_gpio.h"
#include "nrf_delay.h"
#include "pstorage.h"
#include "device_manager.h"
#include "app_trace.h"
#include "ble_uart_c.h"
#include "app_util.h"
#include "simple_uart.h"
#include "app_timer.h"

#define SERIAL_DEBUG        /* Enable serial debugging. */
#define MEASURE_DELAY       /* Enable dealy measurement. */

#define timing_interrupt      17    /* Pin for sending an interrupt to the peer when MEASURE_DELAY is enal
#define START_SEND_PIN        16    /* Pin which is sensed for an interrupt to send a byte to the feedbacl
#define CONNECTED_LED_PIN_NO    15    /* Is on when device has connected. */

#define SEC_PARAM_BOND        0                /* Perform bonding. */
#define SEC_PARAM_MITM        1                /* Man In The Middle protection not required. */
#define SEC_PARAM_IO_CAPABILITIES BLE_GAP_IO_CAPS_NONE        /* No I/O capabilities. */
#define SEC_PARAM_OOB       0               /* Out Of Band data not available. */
#define SEC_PARAM_MIN_KEY_SIZE    7             /* Minimum encryption key size. */
#define SEC_PARAM_MAX_KEY_SIZE    16              /* Maximum encryption key size. */

#define SCAN_INTERVAL       0x00A0            /* Determines scan interval in units of 0.625 millisecor
#define SCAN_WINDOW         0x0050            /* Determines scan window in units of 0.625 millisecond

#define MIN_CONNECTION_INTERVAL   MSEC_TO_UNITS(7.5, UNIT_1_25_MS)  /* Determines maximum connection inte
#define MAX_CONNECTION_INTERVAL   MSEC_TO_UNITS(15, UNIT_1_25_MS)   /* Determines maximum connection inte
#define SLAVE_LATENCY       0               /* Determines slave latency in counts of connection events.
#define SUPERVISION_TIMEOUT     MSEC_TO_UNITS(4000, UNIT_10_MS)   /* Determines supervision time-out in u
```

```
#define MAX_PEER_COUNT        DEVICE_MANAGER_MAX_CONNECTIONS    /* Maximum number of peer's applicat
#define UUID16_SIZE         2                 /* Size of 16 bit UUID */
#define APP_TIMER_PRESCALER     0                 /* Value of the RTC1 PRESCALER register. */
#define APP_TIMER_MAX_TIMERS    4                 /* Maximum number of simultaneously created timers
#define APP_TIMER_OP_QUEUE_SIZE   5                 /* Size of timer operation queues. */

#define UART_SEND_INTERVAL  APP_TIMER_TICKS(1000, APP_TIMER_PRESCALER) /* Timer to send a dummy byte

/* Variable length data encapsulation in terms of length and pointer to data */
typedef struct
{
    uint8_t *   p_data;   /* Pointer to data. */
    uint16_t  data_len; /* Length of data. */
}data_t;

typedef enum
{
    BLE_NO_SCAN,      /* No advertising running. */
    BLE_WHITELIST_SCAN,   /* Advertising with whitelist. */
    BLE_FAST_SCAN,      /* Fast advertising running. */
} ble_advertising_mode_t;

static ble_db_discovery_t           m_ble_db_discovery;           /* Structure used to identify the
static ble_uart_c_t                 m_ble_uart_c;             /* Structure used to identify the UART
static ble_gap_scan_params_t        m_scan_param;             /* Scan parameters requested for scann
static dm_application_instance_t    m_dm_app_id;              /* Application identifier. */
static dm_handle_t                  m_dm_device_handle;           /* Device Identifier identifier. *
static uint8_t                      m_peer_count = 0;           /* Number of peer's connected. */
static uint8_t                      m_scan_mode;              /* Scan mode used by application. */
static bool                         m_memory_access_in_progress = false;  /* Flag to keep track of
static app_timer_id_t               m_uart_send_timer_id;                 /* Dummy timer. */
static char             str[128];               /* String buffer used for debugging through serial p

uint8_t uart_service_uuid[16] = {0x9E, 0xCA, 0xDC, 0x24, 0x0E, 0xE5, 0xA9, 0xE0,
                                 0x93, 0xF3, 0xA3, 0xB5, 0x01, 0x00, 0x40, 0x6E};
/*
 * Connection parameters requested for connection.
 */
static const ble_gap_conn_params_t m_connection_param =
{
    (uint16_t)MIN_CONNECTION_INTERVAL,   // Minimum connection
    (uint16_t)MAX_CONNECTION_INTERVAL,   // Maximum connection
    0,                                   // Slave latency
    (uint16_t)SUPERVISION_TIMEOUT        // Supervision time-out
};

static void scan_start(void);

/* Function for error handling, which is called when an error has occurred.
 *
 * Param[in] error_code  Error code supplied to the handler.
 * Param[in] line_num    Line number where the handler is called.
 * Param[in] p_file_name Pointer to the file name.
 */
void app_error_handler(uint32_t error_code, uint32_t line_num, const uint8_t * p_file_name)
{
  #ifdef SERIAL_DEBUG
    simple_uart_putstring((const uint8_t *)"ASSERT␣ERROR.\r\n");
  #endif //SERIAL_DEBUG

    // On assert, the system can only recover with a reset.
    NVIC_SystemReset();
}

/* Function for asserts in the SoftDevice.
 *
 * Param[in] line_num     Line number of the failing ASSERT call.
 * Param[in] p_file_name  File name of the failing ASSERT call.
 */
void assert_nrf_callback(uint16_t line_num, const uint8_t * p_file_name)
```

```
{
    app_error_handler(0xDEADBEEF, line_num, p_file_name);
}

/* Callback handling device manager events.
 *
 * Details: This function is called to notify the application of device manager events.
 *
 * Param[in]   p_handle      Device Manager Handle. For link related events, this parameter
 *                            identifies the peer.
 * Param[in]   p_event       Pointer to the device manager event.
 * Param[in]   event_status  Status of the event.
 */
static api_result_t device_manager_event_handler(const dm_handle_t     * p_handle,
                                                 const dm_event_t      * p_event,
                                                 const api_result_t      event_result)
{
    uint32_t err_code;

    switch(p_event->event_id)
    {
        case DM_EVT_CONNECTION:
        {
      #ifdef SERIAL_DEBUG
            simple_uart_putstring((const uint8_t *)">> DM_EVT_CONNECTION.\r\n");
      #endif //SERIAL_DEBUG

            nrf_gpio_pin_set(CONNECTED_LED_PIN_NO);
            m_dm_device_handle = (*p_handle);

            // Discover peer's services.
            err_code = ble_db_discovery_start(&m_ble_db_discovery,
                                              p_event->event_param.p_gap_param->conn_handle);
            APP_ERROR_CHECK(err_code);

            m_peer_count++;
            if (m_peer_count < MAX_PEER_COUNT)
            {
                scan_start();
            }
      #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)"<< DM_EVT_CONNECTION.\r\n");
      #endif //SERIAL_DEBUG
            break;
        }

        case DM_EVT_DISCONNECTION:
        {
      #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)">> DM_EVT_DISCONNECTION.\r\n");
      #endif //SERIAL_DEBUG
            memset(&m_ble_db_discovery, 0 , sizeof (m_ble_db_discovery));

            nrf_gpio_pin_clear(CONNECTED_LED_PIN_NO);
            if (m_peer_count == MAX_PEER_COUNT)
            {
                scan_start();
            }
            m_peer_count--;
      #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)"<< DM_EVT_DISCONNECTION.\r\n");
      #endif //SERIAL_DEBUG
            break;
        }

        case DM_EVT_SECURITY_SETUP:
        {
      #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)">> DM_EVT_SECURITY_SETUP.\r\n");
      #endif //SERIAL_DEBUG
```

```
            // Slave security request received from peer, if from a non bonded device,
            // initiate security setup, else, wait for encryption to complete.
            err_code = dm_security_setup_req(&m_dm_device_handle);
            APP_ERROR_CHECK(err_code);

      #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)"<< DM_EVT_SECURITY_SETUP.\r\n");
      #endif //SERIAL_DEBUG
            break;
        }
        case DM_EVT_SECURITY_SETUP_COMPLETE:
        {
      #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)">> DM_EVT_SECURITY_SETUP_COMPLETE.\r\n");
      #endif //SERIAL_DEBUG
            // UART service discovered. Enable notification of RX events.
            err_code = ble_uart_c_notif_enable(&m_ble_uart_c);
            APP_ERROR_CHECK(err_code);
      #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)"<< DM_EVT_SECURITY_SETUP_COMPLETE.\r\n");
      #endif //SERIAL_DEBUG
            break;
        }

        case DM_EVT_LINK_SECURED:
      #ifdef SERIAL_DEBUG
        simple_uart_putstring((const uint8_t *)">> DM_LINK_SECURED_IND.\r\n");
      #endif //SERIAL_DEBUG
      #ifdef SERIAL_DEBUG
        simple_uart_putstring((const uint8_t *)"<< DM_LINK_SECURED_IND.\r\n");
      #endif //SERIAL_DEBUG
            break;

        case DM_EVT_DEVICE_CONTEXT_LOADED:
      #ifdef SERIAL_DEBUG
        simple_uart_putstring((const uint8_t *)">> DM_EVT_LINK_SECURED.\r\n");
      #endif //SERIAL_DEBUG
            APP_ERROR_CHECK(event_result);
      #ifdef SERIAL_DEBUG
        simple_uart_putstring((const uint8_t *)"<< DM_EVT_DEVICE_CONTEXT_LOADED.\r\n");
      #endif //SERIAL_DEBUG
            break;

        case DM_EVT_DEVICE_CONTEXT_STORED:
      #ifdef SERIAL_DEBUG
        simple_uart_putstring((const uint8_t *)">> DM_EVT_DEVICE_CONTEXT_STORED.\r\n");
      #endif //SERIAL_DEBUG
            APP_ERROR_CHECK(event_result);
      #ifdef SERIAL_DEBUG
        simple_uart_putstring((const uint8_t *)"<< DM_EVT_DEVICE_CONTEXT_STORED.\r\n");
      #endif //SERIAL_DEBUG
            break;

        case DM_EVT_DEVICE_CONTEXT_DELETED:
      #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)">> DM_EVT_DEVICE_CONTEXT_DELETED.\r\n");
      #endif //SERIAL_DEBUG
            APP_ERROR_CHECK(event_result);
      #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)"<< DM_EVT_DEVICE_CONTEXT_DELETED.\r\n");
      #endif //SERIAL_DEBUG
            break;

        default:
            break;
    }

    return NRF_SUCCESS;
}

/* Funtion for parsing advertisement data, providing length and location of the field in case
```

```
 * matching data is found.
 *
 * Param[in]  Type of data to be looked for in advertisement data.
 * Param[in]  Advertisement report length and pointer to report.
 * Param[out] If data type requested is found in the data report, type data length and
 *            pointer to data will be populated here.
 *
 * Retval NRF_SUCCESS if the data type is found in the report.
 * Retval NRF_ERROR_NOT_FOUND if the data type could not be found.
 */
static uint32_t adv_report_parse(uint8_t type, data_t * p_advdata, data_t * p_typedata)
{
    uint32_t index = 0;
    uint8_t * p_data;

    p_data = p_advdata->p_data;

    while (index < p_advdata->data_len)
    {
        uint8_t field_length = p_data[index];
        uint8_t field_type = p_data[index+1];

        if (field_type == type)
        {
            p_typedata->p_data = &p_data[index+2];
            p_typedata->data_len = field_length-1;
            return NRF_SUCCESS;
        }
        index += field_length+1;
    }
    return NRF_ERROR_NOT_FOUND;
}

/* Function for handling the Application's BLE Stack events.
 *
 * Param[in]   p_ble_evt   Bluetooth stack event.
 */
static void on_ble_evt(ble_evt_t * p_ble_evt)
{
    uint32_t                err_code;
    const ble_gap_evt_t   * p_gap_evt = &p_ble_evt->evt.gap_evt;

    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_ADV_REPORT:
        {
            data_t adv_data;
            data_t type_data;
      #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)"Caught␣an␣advertising␣packet,␣checking␣for␣UUID␣mat
      #endif //SERIAL_DEBUG

            // Initialize advertisement report for parsing.
            adv_data.p_data = (uint8_t *)p_gap_evt->params.adv_report.data;
            adv_data.data_len = p_gap_evt->params.adv_report.dlen;

            err_code = adv_report_parse(BLE_GAP_AD_TYPE_128BIT_SERVICE_UUID_MORE_AVAILABLE,
                                        &adv_data,
                                        &type_data);
            if (err_code != NRF_SUCCESS)
            {
                // Compare 128 UUID.
                err_code = adv_report_parse(BLE_GAP_AD_TYPE_128BIT_SERVICE_UUID_COMPLETE,
                                            &adv_data,
                                            &type_data);
            }
            // Verify if short or complete name matches target.
            if (err_code == NRF_SUCCESS)
            {
        if(!memcmp(uart_service_uuid,type_data.p_data,16))
        {
```

```
            #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)"\tUUID␣matched\r\n");
            #endif //SERIAL_DEBUG
            // Stop scanning.
            err_code = sd_ble_gap_scan_stop();
            if (err_code != NRF_SUCCESS)
            {
              #ifdef SERIAL_DEBUG
                simple_uart_putstring((const uint8_t *)"Scan␣stop␣failed\r\n");
              #endif //SERIAL_DEBUG
            }
            #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)"Scanning␣finished\r\n");
            #endif //SERIAL_DEBUG

            m_scan_param.selective = 0;

            // Initiate connection.
            err_code = sd_ble_gap_connect(&p_gap_evt->params.adv_report.\
                            peer_addr,
                            &m_scan_param,
                            &m_connection_param);

            if (err_code != NRF_SUCCESS)
            {
              #ifdef SERIAL_DEBUG
                simple_uart_putstring((const uint8_t *)"Connection␣Request␣Failed\r\n");
              #endif //SERIAL_DEBUG
            }
            break;
          }
              }
              break;
          }
          case BLE_GAP_EVT_TIMEOUT:
              if(p_gap_evt->params.timeout.src == BLE_GAP_TIMEOUT_SRC_SCAN)
              {
          #ifdef SERIAL_DEBUG
                    simple_uart_putstring((const uint8_t *)"Scan␣timed␣out.\r\n");
          #endif //SERIAL_DEBUG
                   if (m_scan_mode ==  BLE_WHITELIST_SCAN)
                   {
                       m_scan_mode = BLE_FAST_SCAN;

                       // Start non selective scanning.
                       scan_start();
                   }
              }
              else if (p_gap_evt->params.timeout.src == BLE_GAP_TIMEOUT_SRC_CONN)
              {
          #ifdef SERIAL_DEBUG
                    simple_uart_putstring((const uint8_t *)"Connection␣Request␣timed␣out.\r\n");
          #endif //SERIAL_DEBUG
              }
              break;
          case BLE_GAP_EVT_CONN_PARAM_UPDATE_REQUEST:
              // Accepting parameters requested by peer.
              err_code = sd_ble_gap_conn_param_update(p_gap_evt->conn_handle,
                                                      &p_gap_evt->params.conn_param_update_request.cor
              APP_ERROR_CHECK(err_code);
              break;
          default:
              break;
      }
}

/* Function for handling the Application's system events.
 *
 * Param[in]   sys_evt   system event.
 */
static void on_sys_evt(uint32_t sys_evt)
```

```
{
    switch(sys_evt)
    {
        case NRF_EVT_FLASH_OPERATION_SUCCESS:
        case NRF_EVT_FLASH_OPERATION_ERROR:
            if (m_memory_access_in_progress)
            {
                m_memory_access_in_progress = false;
                scan_start();
            }
            break;
        default:
            // No implementation needed.
            break;
    }
}

/* Function for dispatching a BLE stack event to all modules with a BLE stack event handler.
 *
 * Details: This function is called from the scheduler in the main loop after a BLE stack event has
 *      been received.
 *
 * Param[in]   p_ble_evt   Bluetooth stack event.
 */
static void ble_evt_dispatch(ble_evt_t * p_ble_evt)
{
    dm_ble_evt_handler(p_ble_evt);
    ble_db_discovery_on_ble_evt(&m_ble_db_discovery, p_ble_evt);
    ble_uart_c_on_ble_evt(&m_ble_uart_c, p_ble_evt);
    on_ble_evt(p_ble_evt);
}

/* Function for dispatching a system event to interested modules.
 *
 * Details: This function is called from the System event interrupt handler after a system
 *          event has been received.
 *
 * Param[in]   sys_evt   System stack event.
 */
static void sys_evt_dispatch(uint32_t sys_evt)
{
    pstorage_sys_event_handler(sys_evt);
    on_sys_evt(sys_evt);
}

/* Function for initializing the BLE stack.
 *
 * Details: Initializes the SoftDevice and the BLE event interrupt.
 */
static void ble_stack_init(void)
{
    uint32_t err_code;

    // Initialize the SoftDevice handler module.
    SOFTDEVICE_HANDLER_INIT(NRF_CLOCK_LFCLKSRC_XTAL_20_PPM, false);

    // Register with the SoftDevice handler module for BLE events.
    err_code = softdevice_ble_evt_handler_set(ble_evt_dispatch);
    APP_ERROR_CHECK(err_code);

    // Register with the SoftDevice handler module for System events.
    err_code = softdevice_sys_evt_handler_set(sys_evt_dispatch);
    APP_ERROR_CHECK(err_code);
}

/*
 * Function for initializing the Device Manager.
 */
static void device_manager_init(void)
{
    dm_application_param_t param;
```

```
    dm_init_param_t         init_param;

    uint32_t               err_code;

    err_code = pstorage_init();
    APP_ERROR_CHECK(err_code);

    err_code = dm_init(&init_param);
    APP_ERROR_CHECK(err_code);

    memset(&param.sec_param, 0, sizeof (ble_gap_sec_params_t));

    // Event handler to be registered with the module.
    param.evt_handler            = device_manager_event_handler;

    // Service or protocol context for device manager to load, store and apply on behalf of applicat
    // Here set to client as application is a GATT client.
    param.service_type           = DM_PROTOCOL_CNTXT_GATT_CLI_ID;

    // Securirty parameters to be used for security procedures.
    param.sec_param.bond         = SEC_PARAM_BOND;
    param.sec_param.mitm         = SEC_PARAM_MITM;
    param.sec_param.io_caps      = SEC_PARAM_IO_CAPABILITIES;
    param.sec_param.oob          = SEC_PARAM_OOB;
    param.sec_param.min_key_size = SEC_PARAM_MIN_KEY_SIZE;
    param.sec_param.max_key_size = SEC_PARAM_MAX_KEY_SIZE;
    param.sec_param.kdist_periph.enc = 1;
    param.sec_param.kdist_periph.id  = 1;

    err_code = dm_register(&m_dm_app_id,&param);
    APP_ERROR_CHECK(err_code);
}

/*
 * Function for the LEDs initialization.
 */
static void leds_init(void)
{
    nrf_gpio_cfg_output(CONNECTED_LED_PIN_NO);
  #ifdef MEASURE_DELAY
    nrf_gpio_cfg_output(timing_interrupt);
  #endif //MEAURE_DELAY
}

/*
 * Function for the Power manager.
 */
static void power_manage(void)
{
    uint32_t err_code = sd_app_evt_wait();
    APP_ERROR_CHECK(err_code);
}

/*
 * UART Client Handler.
 */
static void uart_c_evt_handler(ble_uart_c_t * p_uart_c, ble_uart_c_evt_t * p_uart_c_evt)
{
    uint32_t err_code;

    switch (p_uart_c_evt->evt_type)
    {
        case BLE_UART_C_EVT_DISCOVERY_COMPLETE:
            // Initiate bonding.
            err_code = dm_security_setup_req(&m_dm_device_handle);
            APP_ERROR_CHECK(err_code);

            // UART service discovered. Enable notification of RX events.
            err_code = ble_uart_c_notif_enable(p_uart_c);
            APP_ERROR_CHECK(err_code);
      write_dummy();
```

```
        err_code = app_timer_start(m_uart_send_timer_id, UART_SEND_INTERVAL, NULL);
        APP_ERROR_CHECK(err_code);

        #ifdef SERIAL_DEBUG
          simple_uart_putstring((const uint8_t *)"Discovery␣complete.\r\n");
        #endif //SERIAL_DEBUG
              break;

          case BLE_UART_C_EVT_RX:
          {
        #ifdef SERIAL_DEBUG
              simple_uart_putstring((const uint8_t *)"RX␣event␣occured.\r\n");
        #endif //SERIAL_DEBUG
        p_uart_c_evt->params.uart.rx_data[p_uart_c_evt->params.uart.len]=0;

        #ifdef SERIAL_DEBUG
          sprintf(str,"RX␣received:␣%s\r\n", p_uart_c_evt->params.uart.rx_data);
          simple_uart_printf((char *)str);
        #endif //SERIAL_DEBUG
        #ifdef MEASURE_DELAY
          sprintf(str,"%d\r\n", p_uart_c_evt->params.uart.rx_data[0]);
          simple_uart_printf((char *)str);
          nrf_gpio_pin_clear(timing_interrupt);
        #endif

              break;
          }
          default:
              break;
      }
}

/*
 * UART service initialization.
 */
static void uart_c_init(void)
{
    ble_uart_c_init_t uart_c_init_obj;

    uart_c_init_obj.evt_handler = uart_c_evt_handler;

    uint32_t err_code = ble_uart_c_init(&m_ble_uart_c, &uart_c_init_obj);
    APP_ERROR_CHECK(err_code);
}

/*
 * Database discovery collector initialization.
 */
static void db_discovery_init(void)
{
    uint32_t err_code = ble_db_discovery_init();
    APP_ERROR_CHECK(err_code);
}

/*
 * Function to start scanning.
 */
static void scan_start(void)
{
    ble_gap_whitelist_t   whitelist;
    ble_gap_addr_t        * p_whitelist_addr[BLE_GAP_WHITELIST_ADDR_MAX_COUNT];
    ble_gap_irk_t         * p_whitelist_irk[BLE_GAP_WHITELIST_IRK_MAX_COUNT];
    uint32_t              err_code;
    uint32_t              count;

    // Verify if there is any flash access pending, if yes delay starting scanning until
    // it's complete.
    err_code = pstorage_access_status_get(&count);
    APP_ERROR_CHECK(err_code);

    if (count != 0)
```

```
    {
        m_memory_access_in_progress = true;
        return;
    }

    // Initialize whitelist parameters.
    whitelist.addr_count = BLE_GAP_WHITELIST_ADDR_MAX_COUNT;
    whitelist.irk_count  = 0;
    whitelist.pp_addrs   = p_whitelist_addr;
    whitelist.pp_irks    = p_whitelist_irk;

    // Request creating of whitelist.
    err_code = dm_whitelist_create(&m_dm_app_id,&whitelist);
    APP_ERROR_CHECK(err_code);

    if (((whitelist.addr_count == 0) && (whitelist.irk_count == 0)) ||
         (m_scan_mode != BLE_WHITELIST_SCAN))
    {
        // No devices in whitelist, hence non selective performed.
        m_scan_param.active       = 1;            // Active scanning set.
        m_scan_param.selective    = 0;            // Selective scanning not set.
        m_scan_param.interval     = SCAN_INTERVAL;// Scan interval.
        m_scan_param.window       = SCAN_WINDOW;  // Scan window.
        m_scan_param.p_whitelist  = NULL;         // No whitelist provided.
        m_scan_param.timeout      = 0x0000;       // No timeout.
    }
    else
    {
        // Selective scanning based on whitelist first.
        m_scan_param.active       = 1;            // Active scanning set.
        m_scan_param.selective    = 1;            // Selective scanning not set.
        m_scan_param.interval     = SCAN_INTERVAL;// Scan interval.
        m_scan_param.window       = SCAN_WINDOW;  // Scan window.
        m_scan_param.p_whitelist  = &whitelist;   // Provide whitelist.
        m_scan_param.timeout      = 0x001E;       // 30 seconds timeout.

        // Set whitelist scanning state.
        m_scan_mode = BLE_WHITELIST_SCAN;
    }

    err_code = sd_ble_gap_scan_start(&m_scan_param);
    APP_ERROR_CHECK(err_code);

  #ifdef SERIAL_DEBUG
    simple_uart_putstring((const uint8_t *)"Scanning started\r\n");
  #endif //SERIAL_DEBUG
}

#ifdef MEASURE_DELAY
/* Timer handler to send dummy data to peer every one second. */
static void uart_send_timeout_handler(void * p_context)
{
  write_dummy();
}

static void timers_init(void)
{
    uint32_t err_code;

    // Initialize timer module.
    APP_TIMER_INIT(APP_TIMER_PRESCALER, 1, 2, false);
  err_code = app_timer_create(&m_uart_send_timer_id,
                APP_TIMER_MODE_REPEATED,
                uart_send_timeout_handler);
    APP_ERROR_CHECK(err_code);
    // Create timers.

}
#endif //MEASURE_DELAY

/* Function for initialising the UART for serial debugging
```

```
 */
void debug_uart_init(void)
{
  simple_uart_config(8, 9, 10, 11, true);
}

/* Function for configuring: pin 0 for input, pin 8 for output,
 * and configures GPIOTE to give an interrupt on pin change.
 */
static void gpio_init(void)
{
    nrf_gpio_cfg_input(START_SEND_PIN, NRF_GPIO_PIN_NOPULL);

    // Enable interrupt:
    NVIC_EnableIRQ(GPIOTE_IRQn);
    NRF_GPIOTE->CONFIG[START_SEND_PIN] =  (GPIOTE_CONFIG_POLARITY_Toggle << GPIOTE_CONFIG_POLARITY_Pos)
                          | (START_SEND_PIN << GPIOTE_CONFIG_PSEL_Pos)
                          | (GPIOTE_CONFIG_MODE_Event << GPIOTE_CONFIG_MODE_Pos);
    NRF_GPIOTE->INTENSET  = GPIOTE_INTENSET_IN0_Set << GPIOTE_INTENSET_IN0_Pos;
}


/* Function for handling the GPIOTE interrupt which is triggered on pin 0 change.
 */
void GPIOTE_IRQHandler(void)
{
    // Event causing the interrupt must be cleared.
    if ((NRF_GPIOTE->EVENTS_IN[START_SEND_PIN] == 1) &&
        (NRF_GPIOTE->INTENSET & GPIOTE_INTENSET_IN0_Msk))
    {
        NRF_GPIOTE->EVENTS_IN[START_SEND_PIN] = 0;
    }
  write_byte(128,1);
}

int main(void)
{
    // Initialization of various modules.
    debug_uart_init();
  nrf_delay_ms(5000);
  #ifdef SERIAL_DEBUG
    simple_uart_putstring((const uint8_t *)"Init␣succesfull!\r\n");
  #endif //SERIAL_DEBUG
    leds_init();
  #ifdef MEASURE_DELAY
    timers_init();
  #endif //MEASURE_DELAY
    ble_stack_init();
    device_manager_init();
    db_discovery_init();
    uart_c_init();

    // Start scanning for peripherals and initiate connection
    // with devices that advertise UART UUID.
    scan_start();

    for (;;)
    {
        power_manage();
    }
}
```

Code/main.c

```
/* Header file voor de realisatie van een BLE UART service
* door Erwin Visser
* Januari 2015, EE3842 Bachelor Afstudeerproject */

#ifndef BLE_uart_C_H__
#define BLE_uart_C_H__
#define BLE_BASE_UUID_UART_SERVICE    {{0x9E, 0xCA, 0xDC, 0x24, 0x0E, 0xE5, 0xA9, 0xE0,\
```

```
                      0x93, 0xF3, 0xA3, 0xB5, 0x00, 0x00, 0x40, 0x6E}}  /**< The BASE UUID of the Nordic UART Service. */
#define BLE_UUID_UART_SERVICE           0x0001                                      /**< The UUID of the Nordic UART Service. */
#define BLE_UUID_UART_TX_CHARACTERISTIC 0x0002                                      /**< The UUID of the TX Characteristic. */
#define BLE_UUID_UART_RX_CHARACTERISTIC 0x0003                                      /**< The UUID of the RX Characteristic. */
#define WRITE_MESSAGE_LENGTH        20                              /**< Length of the write message for the TX characteristic. */
#define timing_interrupt        17  // DIGITAL 7

#include <stdint.h>
#include "ble.h"

/* Uart Client event type. */
typedef enum
{
  BLE_UART_C_EVT_DISCOVERY_COMPLETE = 1,  /* Event indicating that the UART Service has been discovered at the peer. */
  BLE_UART_C_EVT_RX           /* Event indicating that a notification of the RX characteristic has been received from the peer. */
} ble_uart_c_evt_type_t;

/* Structure containing the RX message received from the peer. */
typedef struct
{
  uint8_t rx_data[20];  /* RX Value. */
  uint8_t len;
} ble_uart_t;

/* UART Event structure. */
typedef struct
{
  ble_uart_c_evt_type_t evt_type;  /* Type of the event. */
  union
  {
    ble_uart_t  uart;  /* UART measurement received. This will be filled if the evt_type is @ref BLE_UART_C_EVT_RX. */
  } params;
} ble_uart_c_evt_t;

/* Forward declaration of the ble_bas_t type. */
typedef struct ble_uart_c_s ble_uart_c_t;

/* Event handler type.
 *
 * Details: This is the type of the event handler that should be provided by the application
 *          of this module in order to receive events.
 */
typedef void (* ble_uart_c_evt_handler_t) (ble_uart_c_t * p_ble_uart_c, ble_uart_c_evt_t * p_evt);

/* UART Client structure.
 */
typedef struct ble_uart_c_s
{
  uint16_t          conn_handle;  /* Connection handle as provided by the SoftDevice. */
  uint16_t          RX_cccd_handle; /* Handle of the CCCD of the RX characteristic. */
  uint16_t          RX_handle;    /* Handle of the RX characteristic as provided by the SoftDevice. */
  uint16_t          TX_handle;    /* Handle of the TX characteristic as provided by the SoftDevice. */
  ble_uart_c_evt_handler_t  evt_handler;  /* Application event handler to be called when there is an event related to the UART. */
} ble_uart_c_t;

/* UART Client initialization structure.
 */
typedef struct
{
    ble_uart_c_evt_handler_t  evt_handler;  /* Event handler to be called by the UART Client module when there is an event related to the UART. */
} ble_uart_c_init_t;

typedef enum
{
    READ_REQ,  /* Type identifying that this tx_message is a read request. */
    WRITE_REQ  /* Type identifying that this tx_message is a write request. */
} tx_request_t;

/* Structure for writing a message to the peer, i.e. CCCD.
 */
typedef struct
```

```
{
    uint8_t                  gattc_value[WRITE_MESSAGE_LENGTH];  /* The message to write. */
    ble_gattc_write_params_t gattc_params;                       /* GATTC parameters for this message. */
} write_params_t;

/* Structure for holding data to be transmitted to the connected central.
 */
typedef struct
{
    uint16_t     conn_handle;  /* Connection handle to be used when transmitting this message. */
    tx_request_t type;         /* Type of this message, i.e. read or write message. */
    union
    {
        uint16_t       read_handle;  /* Read request message. */
        write_params_t write_req;    /* Write request message. */
    } req;
} tx_message_t;

void write_dummy(void);

void write_byte(uint8_t byte, uint8_t len);

uint32_t ble_uart_c_init(ble_uart_c_t * p_ble_uart_c, ble_uart_c_init_t * p_ble_uart_c_init);

void ble_uart_c_on_ble_evt(ble_uart_c_t * p_ble_uart_c, const ble_evt_t * p_ble_evt);

uint32_t ble_uart_c_notif_enable(ble_uart_c_t * p_ble_uart_c);

#endif // BLE_uart_C_H__
```

Code/ble_uart_c.h

```
/* Code voor de realisatie van een BLE UART service
* door Erwin Visser
* Januari 2015, EE3842 Bachelor Afstudeerproject */

#include <stdint.h>
#include <string.h>
#include "ble_uart_c.h"
#include "ble_db_discovery.h"
#include "ble_types.h"
#include "ble_srv_common.h"
#include "nordic_common.h"
#include "nrf_error.h"
#include "nrf_gpio.h"
#include "ble_gattc.h"
#include "app_util.h"
#include "app_trace.h"
#include "simple_uart.h"

#define SERIAL_DEBUG  /* Enable serial debugging. Comment out to disable debugging. */
#define MEASURE_DELAY /* Enable dealy measurement.  Comment out to disable delay measurement. */

#define TX_BUFFER_MASK  0x07  /* TX Buffer mask, must be a mask of continuous zeroes, followed by continu
#define TX_BUFFER_SIZE  (TX_BUFFER_MASK + 1) /* Size of send buffer, which is 1 higher than the mask. */

static ble_uart_c_t *   mp_ble_uart_c;  /* Pointer to the current instance of the uart Client module. The
static tx_message_t   m_tx_buffer[TX_BUFFER_SIZE];  /* Transmit buffer for messages to be transmitted to
static uint32_t       m_tx_insert_index = 0;  /* Current index in the transmit buffer where the next mess
static uint32_t       m_tx_index = 0; /* Current index in the transmit buffer from where the next message
static ble_uuid_t   uart_uuid;
static char       str[128]; /* Buffer for the debug output string. */

/*
 * Function for passing any pending request from the buffer to the stack.
 */
static void tx_buffer_process(void)
{
    if (m_tx_index != m_tx_insert_index)
    {
        uint32_t err_code;
```

```
        if (m_tx_buffer[m_tx_index].type == READ_REQ)
        {
            err_code = sd_ble_gattc_read(m_tx_buffer[m_tx_index].conn_handle,
                                         m_tx_buffer[m_tx_index].req.read_handle,
                                         0);
        }
        else
        {
            err_code = sd_ble_gattc_write(m_tx_buffer[m_tx_index].conn_handle,
                                          &m_tx_buffer[m_tx_index].req.write_req.gattc_params);
        }

        if (err_code == NRF_SUCCESS)
        {
      #ifdef SERIAL_DEBUG
            simple_uart_putstring((const uint8_t *)"SD Read/Write API returns Success.\r\n");
      #endif //SERIAL_DEBUG
            m_tx_index++;
            m_tx_index &= TX_BUFFER_MASK;
        }
        else
        {
      #ifdef SERIAL_DEBUG
            simple_uart_putstring((const uint8_t *)"SD Read/Write API returns error. This message se
      #endif //SERIAL_DEBUG
        }
    }
}


/* Function for handling write response events.
 *
 * Param[in] p_ble_uart_c Pointer to the UART Client structure.
 * Param[in] p_ble_evt    Pointer to the BLE event received.
 */
static void on_write_rsp(ble_uart_c_t * p_ble_uart_c, const ble_evt_t * p_ble_evt)
{
    // Check if there is any message to be sent across to the peer and send it.
    tx_buffer_process();
}


/* Function for handling Handle Value Notification received from the SoftDevice.
 *
 * Details:   This function will uses the Handle Value Notification received from the SoftDevice
 *            and checks if it is a notification of the RX characteristic from the peer. If
 *            it is, this function will send it to the event handler in the main application.
 *
 * Param[in]  p_ble_uart_c Pointer to the UART Client structure.
 * Param[in]  p_ble_evt    Pointer to the BLE event received.
 */
static void on_hvx(ble_uart_c_t * p_ble_uart_c, const ble_evt_t * p_ble_evt)
{
    // Check if this is a UART RX notification.
    if (p_ble_evt->evt.gattc_evt.params.hvx.handle == p_ble_uart_c->RX_handle)
    {
        ble_uart_c_evt_t ble_uart_c_evt;

        ble_uart_c_evt.evt_type = BLE_UART_C_EVT_RX;
    memcpy(ble_uart_c_evt.params.uart.rx_data,
         p_ble_evt->evt.gattc_evt.params.hvx.data,
         p_ble_evt->evt.gattc_evt.params.hvx.len);
    ble_uart_c_evt.params.uart.len = p_ble_evt->evt.gattc_evt.params.hvx.len;

    /* Event created. Is processed by the event handler in main.c */
        p_ble_uart_c->evt_handler(p_ble_uart_c, &ble_uart_c_evt);
    }
}
```

```
/* Function for handling events from the database discovery module.
 *
 * Details:   This function will handle an event from the database discovery module, and determine
 *            if it relates to the discovery of UART service at the peer. If so, it will
 *            call the application's event handler indicating that the UART service has been
 *            discovered at the peer. It also populates the event with the service related
 *            information before providing it to the application.
 *
 * Param[in]  p_evt Pointer to the event received from the database discovery module.
 */
static void db_discover_evt_handler(ble_db_discovery_evt_t * p_evt)
{
    // Check if the UART Service was discovered.
    if (p_evt->evt_type == BLE_DB_DISCOVERY_COMPLETE &&
        p_evt->params.discovered_db.srv_uuid.uuid == BLE_UUID_UART_SERVICE &&
        p_evt->params.discovered_db.srv_uuid.type == uart_uuid.type)
    {
        mp_ble_uart_c->conn_handle = p_evt->conn_handle;

        // Find the CCCD Handle of the characteristics.
        uint32_t i;

        // Find the CCCD Handle of the RX characteristic.
        for (i = 0; i < p_evt->params.discovered_db.char_count; i++)
        {
            if ((p_evt->params.discovered_db.charateristics[i].characteristic.uuid.uuid == BLE_UUID_UART_R
              &&(p_evt->params.discovered_db.charateristics[i].characteristic.uuid.type==uart_uuid.type))

            {
                // Found RX characteristic. Store CCCD handle .
                mp_ble_uart_c->RX_cccd_handle =
                    p_evt->params.discovered_db.charateristics[i].cccd_handle;
                mp_ble_uart_c->RX_handle      =
                    p_evt->params.discovered_db.charateristics[i].characteristic.handle_value;
        #ifdef SERIAL_DEBUG
                simple_uart_putstring((const uint8_t *)"RX handles discovered and stored.\r\n");
        #endif //SERIAL_DEBUG

            }
        // Find the CCCD Handle of the TX characteristic.
    if ((p_evt->params.discovered_db.charateristics[i].characteristic.uuid.uuid == BLE_UUID_UART_TX_CHARAC
      &&(p_evt->params.discovered_db.charateristics[i].characteristic.uuid.type==uart_uuid.type))
            {
                // Found TX characteristic. Store CCCD handle .
                mp_ble_uart_c->TX_handle      =
                    p_evt->params.discovered_db.charateristics[i].characteristic.handle_value;
        #ifdef SERIAL_DEBUG
                simple_uart_putstring((const uint8_t *)"TX handles discovered and stored.\r\n");
        #endif //SERIAL_DEBUG

            }

        }

    #ifdef SERIAL_DEBUG
      simple_uart_putstring((const uint8_t *)"UART Service discovered.\r\n");
    #endif //SERIAL_DEBUG

        ble_uart_c_evt_t evt;

        evt.evt_type = BLE_UART_C_EVT_DISCOVERY_COMPLETE;

        mp_ble_uart_c->evt_handler(mp_ble_uart_c, &evt);
    }
}


uint32_t ble_uart_c_init(ble_uart_c_t * p_ble_uart_c, ble_uart_c_init_t * p_ble_uart_c_init)
{
  ble_uuid128_t   uart_base_uuid = BLE_UART_SERVICE_UUID;
  uint32_t        err_code;
```

```
    if ((p_ble_uart_c == NULL) || (p_ble_uart_c_init == NULL))
    {
        return NRF_ERROR_NULL;
    }

  err_code = sd_ble_uuid_vs_add(&uart_base_uuid, &uart_uuid.type);
  //NOTE: after this, uart_uuid.type will hold the index of the UART 128bit base UUID in the UUID da
  //Store and use this to distinguise between characteristics have different 128bit base UUIDs.

    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    uart_uuid.uuid = BLE_UUID_UART_SERVICE;

    mp_ble_uart_c = p_ble_uart_c;

    mp_ble_uart_c->evt_handler    = p_ble_uart_c_init->evt_handler;
    mp_ble_uart_c->conn_handle    = BLE_CONN_HANDLE_INVALID;
    mp_ble_uart_c->RX_cccd_handle = BLE_GATT_HANDLE_INVALID;

    return ble_db_discovery_evt_register(&uart_uuid,
                                         db_discover_evt_handler);
}


void ble_uart_c_on_ble_evt(ble_uart_c_t * p_ble_uart_c, const ble_evt_t * p_ble_evt)
{
    if ((p_ble_uart_c == NULL) || (p_ble_evt == NULL))
    {
        return;
    }

    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:
            p_ble_uart_c->conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
            break;

        case BLE_GATTC_EVT_HVX:
            on_hvx(p_ble_uart_c, p_ble_evt);
            break;

        case BLE_GATTC_EVT_WRITE_RSP:
            on_write_rsp(p_ble_uart_c, p_ble_evt);
            break;

        default:
            break;
    }
}


/* Function for creating a message for writing to the CCCD.
 */
static uint32_t cccd_configure(uint16_t conn_handle, uint16_t handle_cccd, bool enable)
{
  #ifdef SERIAL_DEBUG
    sprintf(str, "Configuring␣CCCD.␣CCCD␣Handle␣=␣%d,␣Connection␣Handle␣=␣%d\r\n.",  handle_cccd, co
    simple_uart_printf((char *)str);
  #endif //SERIAL_DEBUG

    tx_message_t * p_msg;
    uint16_t       cccd_val = enable ? BLE_GATT_HVX_NOTIFICATION : 0;

    p_msg              = &m_tx_buffer[m_tx_insert_index++];
    m_tx_insert_index &= TX_BUFFER_MASK;

    p_msg->req.write_req.gattc_params.handle   = handle_cccd;
```

```
    p_msg->req.write_req.gattc_params.len      = 2;//WRITE_MESSAGE_LENGTH;
    p_msg->req.write_req.gattc_params.p_value  = p_msg->req.write_req.gattc_value;
    p_msg->req.write_req.gattc_params.offset   = 0;
    p_msg->req.write_req.gattc_params.write_op = BLE_GATT_OP_WRITE_REQ;
    p_msg->req.write_req.gattc_value[0]        = LSB(cccd_val);
    p_msg->req.write_req.gattc_value[1]        = MSB(cccd_val);
    p_msg->conn_handle                         = conn_handle;
    p_msg->type                                = WRITE_REQ;

    tx_buffer_process();
    return NRF_SUCCESS;
}


/* Function for creating a message for writing to the CCCD.
 */
static uint32_t write_char(uint16_t conn_handle, uint16_t char_handle, uint8_t *data, uint8_t len)
{
  #ifdef SERIAL_DEBUG
    sprintf(str, "Writing to characteristic Handle = %d, Connection Handle = %d.\r\n",char_handle,conn_han
    simple_uart_printf((char *)str);
  #endif //SERIAL_DEBUG

    tx_message_t * p_msg;

    p_msg              = &m_tx_buffer[m_tx_insert_index++];
    m_tx_insert_index &= TX_BUFFER_MASK;

    p_msg->req.write_req.gattc_params.handle   = char_handle;
    p_msg->req.write_req.gattc_params.len      = WRITE_MESSAGE_LENGTH;
    p_msg->req.write_req.gattc_params.p_value  = p_msg->req.write_req.gattc_value;
    p_msg->req.write_req.gattc_params.offset   = 0;
    p_msg->req.write_req.gattc_params.write_op = BLE_GATT_OP_WRITE_CMD;
    memcpy(p_msg->req.write_req.gattc_value,data,len);

    p_msg->conn_handle                         = conn_handle;
    p_msg->type                                = WRITE_REQ;

    tx_buffer_process();
    return NRF_SUCCESS;
}

#ifdef MEASURE_DELAY
void write_dummy(void)
{
   //Need to check for state, there is a chance that this is called before connection establish
   //it will be an invalid connection handle if it's not connected
  nrf_gpio_pin_set(timing_interrupt);
  write_char(mp_ble_uart_c->conn_handle, mp_ble_uart_c->TX_handle, (uint8_t *) "dummy", 5);
}
#endif //MEASURE_DELAY

void write_byte(uint8_t byte, uint8_t len)
{
  write_char(mp_ble_uart_c->conn_handle, mp_ble_uart_c->TX_handle, &byte, len);
}

uint32_t ble_uart_c_notif_enable(ble_uart_c_t * p_ble_uart_c)
{
    if (p_ble_uart_c == NULL)
    {
        return NRF_ERROR_NULL;
    }

  #ifdef SERIAL_DEBUG
    simple_uart_putstring((const uint8_t *)"Sending notify enable.\r\n");
  #endif //SERIAL_DEBUG
    return cccd_configure(p_ble_uart_c->conn_handle, p_ble_uart_c->RX_cccd_handle, true);
}
```

Code/ble_uart_c.c

B.2. ENCODER

```
/* Uitlezen  analoge signaal en omzetten in een hoek waarde.
* Door middel van het branden van LED4 kan verschil gemaakt tussen de grote van de hoek
* door Gert-Jan van Raamsdonk
* Januari 2015, EE3842 Bachelor Afstudeerproject */

#include "stm32l1xx_conf.h"
#include "stm32l1xx.h"
#include <stdio.h>
#include "stm32l152_eval.h"
#include "AS5600.h"
#include "stm32l1xx_gpio.h"
#include "stm32l1xx_rcc.h"
#include "stm32l1xx_i2c.h"

int main(){
  int result;

  ENCODER_DeInit();

  STM_EVAL_LEDInit(LED4);
  STM_EVAL_LEDInit(LED3);
  RCC_Confign();
  ENCODER_Init();
  ADC_Configuration();


  while(1){

  result = readADC1();

  if (result <= 180){
    STM_EVAL_LEDOff(LED4);
    }
  else if (result > 180){
    STM_EVAL_LEDOn(LED4);
    }
}
```

Code/main_2_.c

```
/* C header beschrijving van de AS5600 encoder
* door Gert-Jan van Raamsdonk
* Januari 2015, EE3842 Bachelor Afstudeerproject */

#ifndef AS5600_H_
#define AS5600_H_

#include "stm32l152_eval.h"
#include "stm32l1xx_i2c.h"
#include "stm32l1xx_rcc.h"
#include "stm32l1xx_gpio.h"

/*Pin assigments */
#define ENCODER_I2C                          I2C2
#define ENCODER_I2C_CLK                      RCC_APB1Periph_I2C2
#define ENCODER_I2C_GPIO_PORT              GPIOB                        // GPIOB
#define ENCODER_ANALOOG_PIN          GPIO_Pin_0

#define ENCODER_I2C_SCL_PIN                  GPIO_Pin_10                // PB.13
#define ENCODER_I2C_SCL_GPIO_PORT            GPIOB                      // GPIOB
#define ENCODER_I2C_SCL_GPIO_CLK             RCC_AHBPeriph_GPIOB
#define ENCODER_I2C_SCL_SOURCE               GPIO_PinSource10
#define ENCODER_I2C_SCL_AF                    GPIO_AF_I2C2

#define ENCODER_I2C_SDA_PIN                  GPIO_Pin_11                //* PB.13
#define ENCODER_I2C_SDA_GPIO_PORT            GPIOB                      // GPIOB
#define ENCODER_I2C_SDA_GPIO_CLK             RCC_AHBPeriph_GPIOB
#define ENCODER_I2C_SDA_SOURCE               GPIO_PinSource11
#define ENCODER_I2C_SDA_AF                    GPIO_AF_I2C2
```

```
/* Registers  */
#define ENCODER_REG_RAW_ANGLE       0x0D
#define ENCODER_REG_ANGLE         0x0F
#define ENCODER_REG_ZPOS      0x02  //start position register
#define ENCODER_REG_MPOS      0x04  //stop position register
#define ENCODER_REG_MANG      0x06  //Maximal Angle register

#define I2C_TIMEOUT             ((uint32_t)0x3FFFF)
#define ENCODER_ADDR              0x36
#define ENCODER_I2C_SPEED         100000 //I2C Speed


#define ENCODER_FLAG_TIMEOUT         ((uint32_t)0x1000)
#define ENCODER_LONG_TIMEOUT         ((uint32_t)(10 * ENCODER_FLAG_TIMEOUT))


/*Functions*/
void ENCODER_DeInit(void);
void ENCODER_Init(void);
void RCC_Confign(void);
void ADC_Configuration(void);

ErrorStatus ENCODER_GetStatus(void);

void delay(unsigned int time);
void blinky1(unsigned int times);
void blinky2(unsigned int times);
void blinky3(unsigned int times);

uint8_t LeesRegister(uint16_t RegName);
uint8_t readADC1(void);

#ifdef USE_TIMEOUT_USER_CALLBACK
 uint8_t ENCODER_TIMEOUT_UserCallback(void);
#else
 #define ENCODER_TIMEOUT_UserCallback() ENCODER_FAIL
#endif /* USE_TIMEOUT_USER_CALLBACK */


#endif /* AS5600_H_ */
```

Code/AS5600.h

```
/* C code beschrijving van de AS5600 encoder
* door Gert-Jan van Raamsdonk
* Januari 2015, EE3842 Bachelor Afstudeerproject */

#include "AS5600.h"
 __IO uint32_t  ENCODER_Timeout = ENCODER_LONG_TIMEOUT;

/*Encoder deInit*/

/* DeInitializes the ENCODER_I2C.*/
void ENCODER_LowLevel_DeInit(void){
  GPIO_InitTypeDef  GPIO_InitStructure;

    RCC_APB1PeriphClockCmd(ENCODER_I2C_CLK, DISABLE);/*!< ENCODER_I2C Periph clock disable */

  /*!< Configure ENCODER_I2C pins: SCL */
  GPIO_InitStructure.GPIO_Pin = ENCODER_I2C_SCL_PIN;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(ENCODER_I2C_SCL_GPIO_PORT, &GPIO_InitStructure);

  /*!< Configure ENCODER_I2C pins: SDA */
  GPIO_InitStructure.GPIO_Pin = ENCODER_I2C_SDA_PIN;
  GPIO_Init(ENCODER_I2C_SDA_GPIO_PORT, &GPIO_InitStructure);
}
```

```
void ENCODER_DeInit(void){
  ENCODER_LowLevel_DeInit();}

/* Encoder I2C Init*/
void ENCODER_LowLevel_Init(void)
{
  GPIO_InitTypeDef  GPIO_InitStructure;

  /*GPIO_CLK Periph clock enable */
  RCC_AHBPeriphClockCmd(ENCODER_I2C_SCL_GPIO_CLK | ENCODER_I2C_SDA_GPIO_CLK, ENABLE);
  /*!< ENCODER_I2C Periph clock enable */
  RCC_APB1PeriphClockCmd(ENCODER_I2C_CLK, ENABLE);


 /* Connect PXx to I2C_SCL */
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource10,GPIO_AF_I2C2) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource11,GPIO_AF_I2C2) ;

  /* Configure ENCODER_I2C pins: SCL */
  GPIO_InitStructure.GPIO_Pin = ENCODER_I2C_SCL_PIN;    //PB_10
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;        //alternated function
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;     //reactietijd pinnen
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;      //Push Pull
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;       // pull-up resistor intern
  GPIO_Init(ENCODER_I2C_SCL_GPIO_PORT, &GPIO_InitStructure);

  /* Configure ENCODER_I2C pins: SDA */
  GPIO_InitStructure.GPIO_Pin = ENCODER_I2C_SDA_PIN;    //PB_11
  GPIO_Init(ENCODER_I2C_SDA_GPIO_PORT, &GPIO_InitStructure);

  /*Configure ADC pins:  analog input */
    GPIO_InitStructure.GPIO_Pin = ENCODER_ANALOOG_PIN;    //PC_0
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;      //analoog
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Speed =  GPIO_Speed_40MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);


}
void ENCODER_Init(void)
{
  I2C_InitTypeDef I2C_InitStructure;
  ENCODER_LowLevel_Init();

  /* Init I2C */
  I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
  I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
  I2C_InitStructure.I2C_OwnAddress1 =  ENCODER_ADDR;            //  Slaaf adres 0110110
  I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
  I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
  I2C_InitStructure.I2C_ClockSpeed = 100000;             //I2C snelheid 100kHz
  I2C_Init(ENCODER_I2C,&I2C_InitStructure);
  I2C_Cmd(ENCODER_I2C, ENABLE);
 }
/*ANAloog Int*/
void ADC_Configuration(void)
{
  ADC_InitTypeDef ADC_InitStructure;

  /*Configure ADC1 - Channel 10*/
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_8b;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;              // 1 kanaal
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 1;
  ADC_Init(ADC1, &ADC_InitStructure);

  ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_384Cycles); // ADC rank en snelhe

  ADC_Cmd(ADC1, ENABLE); //Enable ADC1
```

```
  /* Wait until the ADC1 is ready */
  while(ADC_GetFlagStatus(ADC1, ADC_FLAG_ADONS) == RESET);
}

void RCC_Confign(void){
  RCC_DeInit();

  /* Enable the HSI */
  RCC_HSICmd(ENABLE);

  /* Wait until HSI oscillator is ready */
  while(RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET);

  RCC_HCLKConfig(RCC_SYSCLK_Div1);
  RCC_PCLK1Config(RCC_HCLK_Div1);
  RCC_PCLK2Config(RCC_HCLK_Div1);
  RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);

  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA | RCC_AHBPeriph_GPIOB | RCC_AHBPeriph_GPIOC, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
}
/*Direct Memory Acces*/
static void ENCODER_DMA_Config(ENCODER_DMADirection_TypeDef Direction, uint8_t* buffer, uint8_t NumData){
  DMA_InitTypeDef DMA_InitStructure;

  RCC_AHBPeriphClockCmd(ENCODER_DMA_CLK, ENABLE);

  /* Initialize the DMA_PeripheralBaseAddr member */
  DMA_InitStructure.DMA_PeripheralBaseAddr = ENCODER_I2C_DR;
  /* Initialize the DMA_MemoryBaseAddr member */
  DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)buffer;
   /* Initialize the DMA_PeripheralInc member */
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  /* Initialize the DMA_MemoryInc member */
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
  /* Initialize the DMA_PeripheralDataSize member */
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
  /* Initialize the DMA_MemoryDataSize member */
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
  /* Initialize the DMA_Mode member */
  DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
  /* Initialize the DMA_Priority member */
  DMA_InitStructure.DMA_Priority = DMA_Priority_VeryHigh;
  /* Initialize the DMA_M2M member */
  DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

  /* If using DMA for Reception */
  if (Direction == ENCODER_DMA_RX)
  {
    /* Initialize the DMA_DIR member */
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;

    /* Initialize the DMA_BufferSize member */
    DMA_InitStructure.DMA_BufferSize = NumData;

    DMA_DeInit(ENCODER_DMA_RX_CHANNEL);

    DMA_Init(ENCODER_DMA_RX_CHANNEL, &DMA_InitStructure);
  }
   /* If using DMA for Transmission */
  else if (Direction == ENCODER_DMA_TX)
  {
    /* Initialize the DMA_DIR member */
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;

    /* Initialize the DMA_BufferSize member */
    DMA_InitStructure.DMA_BufferSize = NumData;

    DMA_DeInit(ENCODER_DMA_TX_CHANNEL);
```

```
    DMA_Init(ENCODER_DMA_TX_CHANNEL, &DMA_InitStructure);
  }
}


/*Checks the ENCODER status, (ERROR or SUCCESS)  */
ErrorStatus ENCODER_GetStatus(void){
  uint32_t I2C_TimeOut = I2C_TIMEOUT;

  /*!< Clear the ENCODER_I2C AF flag */
  I2C_ClearFlag(ENCODER_I2C, I2C_FLAG_AF);

  /*!< Enable ENCODER_I2C acknowledgement*/
  I2C_AcknowledgeConfig(ENCODER_I2C, ENABLE);

  /*!< Send ENCODER_I2C START condition */
  I2C_GenerateSTART(ENCODER_I2C, ENABLE);

  /*!< Test on ENCODER_I2C EV5 and clear it */
  while ((!I2C_GetFlagStatus(ENCODER_I2C,I2C_FLAG_SB)) && I2C_TimeOut){  /*!< EV5 */
      I2C_TimeOut--;}

  if (I2C_TimeOut == 0)  {
   return ERROR;}

  I2C_TimeOut = I2C_TIMEOUT;
  /*!< Send STENCODER slave address for write */
  I2C_Send7bitAddress(ENCODER_I2C, ENCODER_ADDR, I2C_Direction_Transmitter);

  while ((!I2C_CheckEvent(ENCODER_I2C, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED)) && I2C_TimeOut){
    I2C_TimeOut--; }

  if ((I2C_GetFlagStatus(ENCODER_I2C, I2C_FLAG_AF) != 0x00) || (I2C_TimeOut == 0))
  {   return ERROR; }
    else  {
    return SUCCESS;}
}

/* Programming zero and maximum angle  through the I2C interface */
void Encoder_startup(void){
  uint16_t  start, stop;

  start = ENCODER_ReadReg(ENCODER_REG_RAW_ANGLE);
  ENCODER_WriteReg(ENCODER_REG_ZPOS, start);
  blinky2(1);                  //start positie ingelezen, beweeg ten minsten 18 graden
  delay(1000);                 //wacht ten minsten least 1ms
  blinky2(1);                  //stop positie wordt zometeen ingelezen
  stop = ENCODER_ReadReg(ENCODER_REG_RAW_ANGLE);
  ENCODER_WriteReg(ENCODER_REG_MPOS, stop);
  delay(1000);
  blinky3(2);                  //bereik encoder ingesteld
}

/* Read analog signal and covert to 8-bit */
uint8_t readADC1(void){
  // Start the conversion
  ADC_SoftwareStartConv(ADC1);

  // Wait until conversion completion
  while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);

  // Get the 8-bit conversion value
  return((uint8_t)ADC_GetConversionValue(ADC1));
}

/* Read a register - ENCODER_REG_ANGLE:     scaled and filtered
  *                 - ENCODER_REG_RAW_Angle: unscaled and unmodified
  *                 - ENCODER_REG_ZPOS:      start position
  *                 - ENCODER_REG_MPOS:      stop position
  *                 - ENCODER_REG_MANG:      Maximal Angle  */
uint8_t ENCODER_ReadReg(uint8_t RegName)
```

```
{
  uint8_t ENCODER_BufferRX
  uint8_t tmp = 0;

   /* Test on BUSY Flag */
  ENCODER_Timeout = ENCODER_LONG_TIMEOUT;
  while (I2C_GetFlagStatus(ENCODER_I2C,I2C_FLAG_BUSY == RESET))  {
    if((ENCODER_Timeout--) == 0) return ENCODER_TIMEOUT_UserCallback();  }

  /* Configure DMA Peripheral */
  ENCODER_DMA_Config(ENCODER_DMA_RX, (uint8_t*)ENCODER_BufferRX, 8);

  /* Enable DMA NACK automatic generation */
 I2C_DMALastTransferCmd(ENCODER_I2C, ENABLE);

  /* Enable the I2C peripheral */
  I2C_GenerateSTART(ENCODER_I2C, ENABLE);

  /* Test on SB Flag */
  ENCODER_Timeout = ENCODER_FLAG_TIMEOUT;
  while (!I2C_GetFlagStatus(ENCODER_I2C,I2C_FLAG_SB))  {
    if((ENCODER_Timeout--) == 0)  blinky1(1); }

  /* Send device address for write */
  I2C_Send7bitAddress(ENCODER_I2C, ENCODER_ADDR, I2C_Direction_Transmitter);     // R/W =0

  /* Test on ADDR Flag */
  ENCODER_Timeout = ENCODER_FLAG_TIMEOUT;
  while (!I2C_CheckEvent(ENCODER_I2C, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED))  {
    if((ENCODER_Timeout--) == 0) blinky1(2);  }

  /* Send the device's internal address to write to */
  I2C_SendData(ENCODER_I2C, RegName);

  /* Test on TXE FLag (data sent) */
  ENCODER_Timeout = ENCODER_FLAG_TIMEOUT;
  while ((!I2C_GetFlagStatus(ENCODER_I2C,I2C_FLAG_TXE)) && (!I2C_GetFlagStatus(ENCODER_I2C,I2C_FLAG_BTF)))
{
    if((ENCODER_Timeout--) == 0) blinky1(4);  }

  /* Send START condition a second time */
  I2C_GenerateSTART(ENCODER_I2C, ENABLE);

  /* Test on SB Flag */
  ENCODER_Timeout = ENCODER_FLAG_TIMEOUT;
  while (!I2C_GetFlagStatus(ENCODER_I2C,I2C_FLAG_SB))  {
    if((ENCODER_Timeout--) == 0)  blinky1(5);  }

  /* Send ENCODER address for read */
  I2C_Send7bitAddress(ENCODER_I2C, ENCODER_ADDR, I2C_Direction_Receiver);

  /* Test on ADDR Flag */
  ENCODER_Timeout = ENCODER_FLAG_TIMEOUT;
  while (!I2C_CheckEvent(ENCODER_I2C, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED))  {
    if((ENCODER_Timeout--) == 0) blinky1(5);  }

  /* Enable I2C DMA request */
  I2C_DMACmd(ENCODER_I2C,ENABLE);

  /* Enable DMA RX Channel */
  DMA_Cmd(ENCODER_DMA_RX_CHANNEL, ENABLE);

  /* Wait until DMA Transfer Complete */
  ENCODER_Timeout = ENCODER_LONG_TIMEOUT;
  while (!DMA_GetFlagStatus(ENCODER_DMA_RX_TCFLAG))  {
  if((ENCODER_Timeout--) == 0) return ENCODER_TIMEOUT_UserCallback();  }

  /* Send STOP Condition */
  I2C_GenerateSTOP(ENCODER_I2C, ENABLE);

  /* Disable DMA RX Channel */
```

```
  DMA_Cmd(ENCODER_DMA_RX_CHANNEL, DISABLE);

  /* Disable I2C DMA request */
  I2C_DMACmd(ENCODER_I2C,DISABLE);

  /* Clear DMA RX Transfer Complete Flag */
  DMA_ClearFlag(ENCODER_DMA_RX_TCFLAG);

  /*!< Store ENCODER_I2C received data */
  tmp = (uint16_t)(ENCODER_BufferRX[0] << 8);
  tmp |= ENCODER_BufferRX[1];

  /* return a Reg value */
  return (uint8_t)tmp;
}

/* Write a value to a register */
uint8_t ENCODER_WriteReg(uint8_t RegName, uint16_t RegValue)
{
  uint8_t ENCODER_BufferTX[2] ={0,0};
  ENCODER_BufferTX[0] = (uint8_t)(RegValue >> 8);
  ENCODER_BufferTX[1] = (uint8_t)(RegValue);

  /* Test on BUSY Flag */
  ENCODER_Timeout = ENCODER_LONG_TIMEOUT;
  while (I2C_GetFlagStatus(ENCODER_I2C,I2C_FLAG_BUSY))  {
    if((ENCODER_Timeout--) == 0) return ENCODER_TIMEOUT_UserCallback();  }

  /* Configure DMA Peripheral */
  ENCODER_DMA_Config(ENCODER_DMA_TX, (uint8_t*)ENCODER_BufferTX, 2);

  /* Enable the I2C peripheral */
  I2C_GenerateSTART(ENCODER_I2C, ENABLE);

  /* Test on SB Flag */
  ENCODER_Timeout = ENCODER_FLAG_TIMEOUT;
  while (I2C_GetFlagStatus(ENCODER_I2C,I2C_FLAG_SB) == RESET)  {
    if((ENCODER_Timeout--) == 0) return ENCODER_TIMEOUT_UserCallback();  }

  /* Transmit the slave address and enable writing operation */
  I2C_Send7bitAddress(ENCODER_I2C, ENCODER_ADDR, I2C_Direction_Transmitter);

  /* Test on ADDR Flag */
  ENCODER_Timeout = ENCODER_FLAG_TIMEOUT;
  while (!I2C_CheckEvent(ENCODER_I2C, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED))  {
    if((ENCODER_Timeout--) == 0) return ENCODER_TIMEOUT_UserCallback();  }

  /* Transmit the first address for r/w operations */
  I2C_SendData(ENCODER_I2C, RegName);

  /* Test on TXE FLag (data sent) */
  ENCODER_Timeout = ENCODER_FLAG_TIMEOUT;
  while ((!I2C_GetFlagStatus(ENCODER_I2C,I2C_FLAG_TXE)) && (!I2C_GetFlagStatus(ENCODER_I2C,I2C_FLAG_
{
    if((ENCODER_Timeout--) == 0) return ENCODER_TIMEOUT_UserCallback();  }

  /* Enable I2C DMA request */
  I2C_DMACmd(ENCODER_I2C,ENABLE);

  /* Enable DMA TX Channel */
  DMA_Cmd(ENCODER_DMA_TX_CHANNEL, ENABLE);

  /* Wait until DMA Transfer Complete */
  ENCODER_Timeout = ENCODER_LONG_TIMEOUT;
  while (!DMA_GetFlagStatus(ENCODER_DMA_TX_TCFLAG))  {
    if((ENCODER_Timeout--) == 0) return ENCODER_TIMEOUT_UserCallback();  }

  /* Wait until BTF Flag is set before generating STOP */
  ENCODER_Timeout = ENCODER_LONG_TIMEOUT;
  while (I2C_GetFlagStatus(ENCODER_I2C,I2C_FLAG_BTF))  {
    if((ENCODER_Timeout--) == 0) return ENCODER_TIMEOUT_UserCallback();  }
```

```
  /* Send STOP Condition */
  I2C_GenerateSTOP(ENCODER_I2C, ENABLE);

  /* Disable DMA TX Channel */
  DMA_Cmd(ENCODER_DMA_TX_CHANNEL, DISABLE);

  /* Disable I2C DMA request */
  I2C_DMACmd(ENCODER_I2C,DISABLE);

  /* Clear DMA TX Transfer Complete Flag */
  DMA_ClearFlag(ENCODER_DMA_TX_TCFLAG);

  return ENCODER_OK;
}
```

Code/AS5600.c

B.3. TRILMOTOR

```
/* Code voor de realisatie van de Bluetooth en PWM controlle
* door Erwin Visser
* Januari 2015, EE3842 Bachelor Afstudeerproject */

#include <SPI.h>
#include "BLE_UART.h"

#define BLE_REQ 10
#define BLE_RDY 2
#define BLE_RST 9
#define MEASURE_DELAY

int picovibePin = 6;
uint8_t echoVal = 0;
#ifdef MEASURE_DELAY
  unsigned long time = 0;
  int interruptPin = 6;
#endif //MEASURE_DELAY

BLE_UART uart = BLE_UART(BLE_REQ, BLE_RDY, BLE_RST);


/**************************************************************************/
/*!
    This function is called as an interrupt by hardware interrupt 1
*/
/**************************************************************************/
#ifdef MEASURE_DELAY
void startCount() {
  time = millis();
}
#endif //MEASURE_DELAY
/**************************************************************************/
/*!
    This function is called whenever select ACI events happen
*/
/**************************************************************************/
void aciCallback(aci_evt_opcode_t event)
{
  switch(event)
  {
    case ACI_EVT_DEVICE_STARTED:
      break;
    case ACI_EVT_CONNECTED:
      break;
    case ACI_EVT_DISCONNECTED:
      break;
    default:
      break;
  }
}
```

```
/**************************************************************************/
/*!
    This function is called whenever data arrives on the RX channel
*/
/**************************************************************************/
void rxCallback(uint8_t *buffer, uint8_t len)
{
  #ifdef MEASURE_DELAY
    unsigned long result, current;

    echoVal = *buffer;
    analogWrite(picovibePin, echoVal);  // Write the value as dutycycle to the PWM pin

    uint8_t result_write = 0;
    current = millis();
    result = millis() - time;
    result_write = (uint8_t)result;
    time = 0;

    uart.write(&result_write, 1);
  #else
    echoVal = *buffer;
    analogWrite(picovibePin, echoVal);  // Write the value as dutycycle to the PWM pin
    uart.write(&echoVal, 1);
  #endif //MEASURE_DELAY
}

/**************************************************************************/
/*!
    Configure the Arduino and start advertising with the radio
*/
/**************************************************************************/
void setup(void)
{
  #ifdef MEASURE_DELAY
  pinMode(interruptPin, INPUT);        //Interrupt pin for timing
  attachInterrupt(1,startCount,RISING); //Attatch interrupt procedure
  #endif //MEASURE_DELAY

  pinMode(picovibePin, OUTPUT);   // sets the pin as output
  analogWrite(picovibePin, 0);  // configure the pin as low output

  uart.setRXcallback(rxCallback);
  uart.setACIcallback(aciCallback);
  uart.setDeviceName("PICOVIB"); /* 7 characters max! */
  uart.begin();
}

/**************************************************************************/
/*!
    Constantly checks for new events on the nRF8001
*/
/**************************************************************************/
void loop()
{
  uart.pollACI();
}
```

Code/FeedbackCode.c

BIBLIOGRAPHY

[1] Duosport, *Techniek - duosport,* http://www.duosport.nl/techniek/.

[2] P. E. Di Prampero, G. Cortili, P. Mognoni, and F. Saibene, *Energy cost of speed skating and efficiency of work against air resistance,* Journal of Applied Physiology **40**, 584 (1976).

[3] D. I. Inc, *This is ant - ant+ in mobile,* (), http://www.thisisant.com/business/opportunities/mobile/.

[4] P. Smith, *Comparing low-power wireless technologies,* Techzone (2011).

[5] M. N. D. Chen and A. Mok, *WirelessHART: Real-Time Mesh Network for Industrial Automation* (Springer, 2010).

[6] D. I. Inc, *Sending data through an 802.15.4 network latency timing. - digi international,* (), http://www.digi.com/support/kbase/kbaseresultdetl?id=3065.

[7] K. Sidhu, *Understanding linear position sensing technologies,* http://www.sensorsmag.com/position-presence-proximity/understanding-linear-position-sensing-technologies-10139.

[8] i. C. iC Haus, *Absolute encoder design: Magnetic or optical?* http://www.ichaus.de.

[9] J. C. McCallum, *Flash memory prices (2003-2014),* http://www.jcmit.com/flashprice.htm.

[10] IEEE, *Ieee long island section, ascii code table,* (), http://www.ieee.li/computer/ascii.htm.

[11] Toshiba, *What is the difference between nand and nor flash?* EDN Network (2006).

[12] A. Tal, *Two flash technologies compared: Nor vs nand,* M-Systems Inc. (2002).

[13] IEEE, *Ieee long island section, ascii code table,* (), http://www.usb.org.

[14] P. B. Shull, W. Jirattigalachote, M. A. Hunt, M. R. Cutkosky, and S. L. Delp, *Quantified self and human movement: A review on the clinical impact of wearable sensing and feedback for gait analysis and intervention,* Gait and Posture **40**, 11 (2014).

[15] J. Whitaker, *The Electronics Handbook*, Vol. 2 (CRC Press, 2005).

[16] G. T., *Electrical Power Transmission System Engineering* (CRC Press, 2014-14-5) Chap. 2, pp. 77–78.

[17] T. Miwa, *Evaluation methods for vibration effect*, The electrical greatness of the pulses, Vol. 7 (Ind Health, 1968) pp. 143–164.

[18] K. N. R., *Effect of frequency and amplitude of vibration,* http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3146107/.

[19] Mide, *Hek piezoelectric driver board,* http://www.mide.com.

[20] L. Spilsbury and R. Spilsbury, *Light and Sound* (Raintree, 2014) pp. 13–16.

[21] Medicinfo, *Gehoorapparaten met beengeleiding,* http://www.medicinfo.nl/%7B28d0490f-26eb-4aad-baf2-7045a2349541%7D.

[22] T. Instruments, *Haptics solutions for erm and lra actuators,* http://www.ti.com/lit/ml/sszb151/sszb151.pdf.

[23] J. L. H. David A. Patterson, *Computer organisation and design* (Morgan Kaufmann, 2014) pp. 28–40.

[24] ARM, *Cortex-m series - arm,* http://www.arm.com/products/processors/cortex-m/.

[25] V. M. P., *Data Communications Networking* (Purdue University Press, 2006-1-11) Chap. 1, pp. 15–16.

[26] N. Semiconductors, *3rd party module partners - nordic semiconductor,* http://www.nordicsemi.com/Products/3rd-Party-Bluetooth-Smart-Modules.

[27] M. Mando, *Pull-up resistors,* https://www.sparkfun.com/tutorials/pull-up-resistors.

[28] J. R. P. M. M. A. V. K. S. F. E.G., *PowerDistribution Networks with On-Chip Decoupling Capacitors* (Springer Science and Business Media, 2010-23-11) pp. 93–108.