SCHOOL VOOR WETENSCHAP EN TECHNIEK
St.-Jozef Schoten
Een warm hart voor technologie
www.sjs.be

OPDRACHT GEÏNTEGREERDE PROEF

BALANCERENDE ROBOT

Wannes Op de Beeck ● n°6

Studierichting TSO ● Elektronica-ICT

Tweede leerjaar ● derde graad

Schooljaar ● 2016 - 2017

Mentor/Leerkracht: Bart Huyskens

Woord vooraf

Ik ben Wannes Op de Beeck en ik zit in het zesde jaar van de richting Elektronica-ICT aan het Sint-Jozefinstituut in Schoten.

Als onderwerp voor mijn geïntegreerde proef heb ik het voorstel gekregen om een zelf-balancerende robot te maken, het interesseerde mij omdat ik al van jongs af aan met robotjes speel en ik vind het ook gewoon interessant om dingen te kunnen laten bewegen. In mijn GIP wil ik bewijzen dat ik de kennis die ik de laatste jaren vergaard heb kan toepassen en er iets mee kan maken. Het was ook mijn bedoeling om er iets leuk, en toch wel handig ,op grootschaliger vlak, van te maken.

Om deze GIP tot stand te kunnen brengen heb ik kunnen rekenen op de hulp en steun van een aantal mensen in mijn omgeving. Eerst en vooral wil ik Bart Huyskens, Dennis Goeyvaerts en mijn stiefbroer (Jelle Vandeweyer) bedanken om mij ideëen en de nodige richtlijnen te geven om stap voor stap deze GIP uit te werken. Ook wil ik mevrouw Jansen bedanken voor de tips en het nalezen van mijn GIP.

Inhoudsopgave

Inleiding

In dit werk maak ik een balancerende robot, vergelijkbaar met een hoverboard of een segway. De bedoeling is om via Bluetooth met een applicatie op uw smart device deze balancerende robot te kunnen besturen. Het spreekt voor zich dat deze robot tevens recht moet blijven staan als je er een duwtje tegen geeft.

Het nut van dit project is om op termijn grootschaliger te kunnen gaan denken en dus deze balancerende robot als vervoersmiddel te kunnen gaan gebruiken. Voordelen hiervan zijn dat het relatief klein kan zijn en dat het op eenvoudige manier bestuurbaar is. Ik heb ook een gerelateerde opdracht gemaakt voor het vak Frans en Engels. Voor Frans moesten we een artikel zoeken in verband met ons onderwerp. Uit dit artikel hebben we een woordenlijst gehaald en vervolgens een presentatie gegeven aan de klas. Voor Engels moesten we foto’s maken/opzoeken die ons voorwerp voorstellen.

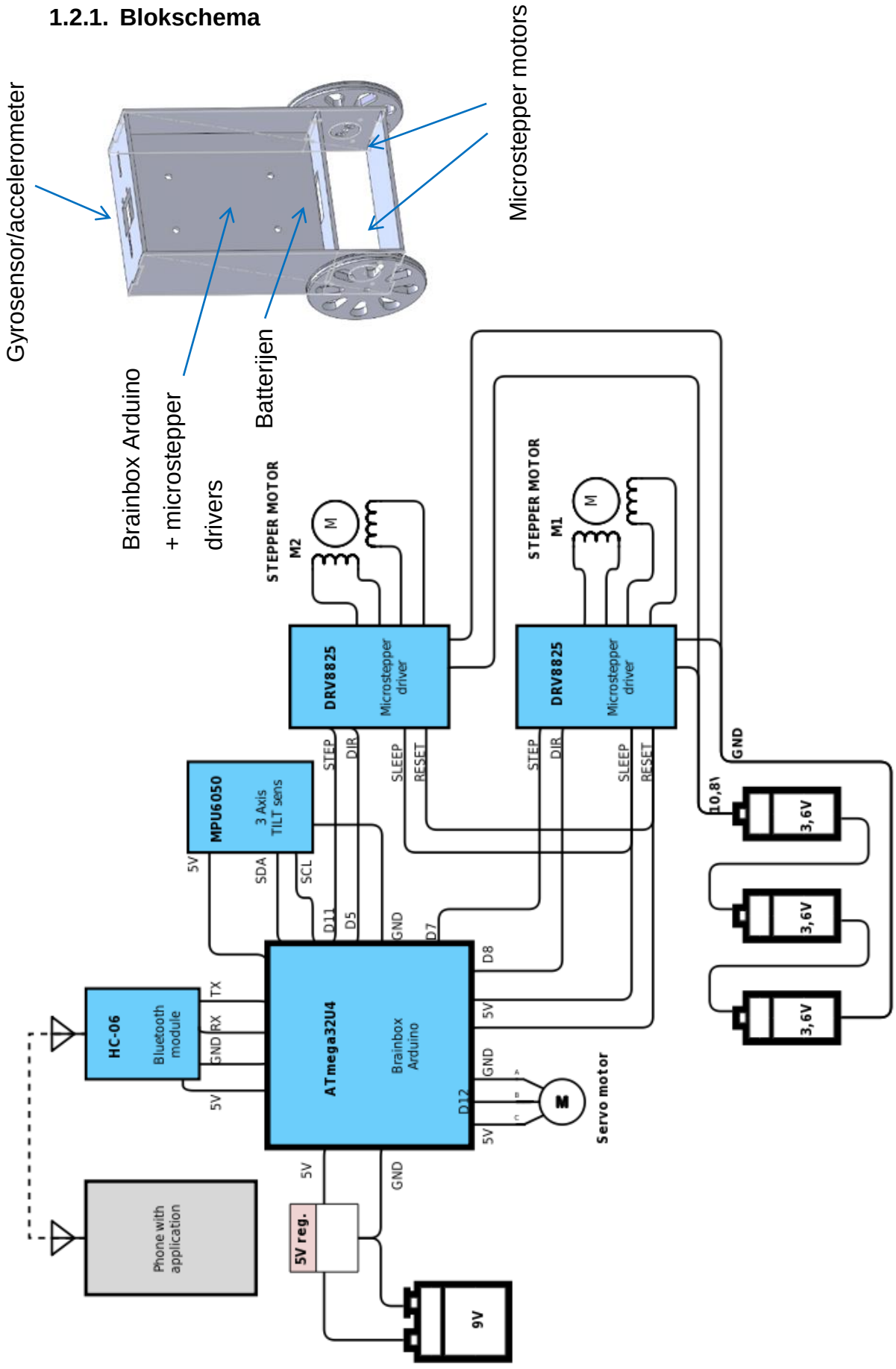
In dit werk vindt u onder andere verschillende schema’s met een beknopte theoretische uitleg, de verschillende benodigde componenten en de code voor de robot en de code van App Inventor voor de applicatie. Ten slotte vind je ook mijn verschillende vak opdrachten terug.

Product	Bestelnummer Farnell	Andere leverancier	Aantal	Prijs/stuk	Prijs
Brainbox Arduino	/	e2cre8.be	1	€ 47,00	€ 47,00
HC-06 Bluetooth module	/	e2cre8.be	1	€ 9,00	€ 9,00
GY-521 (MPU-6050) gyrosensor + accelerometer	/	hobbyelectronica.nl	1	€ 4,95	€ 4,95
Stappenmotor	2507563	/	2	€ 24,55	€ 49,10
Pololu DRV8825 microstepper driver	/	pololu.com	2	€ 8,95	€ 17,90
Servomotor	2075365	/	1	€ 9,63	€ 9,63
5V LDO	2436390	/	1	€ 0,13	€ 0,13
Batterij 9V	2503729	/	1	€ 2,11	€ 2,11
Power batterij 3,6V	/	/	3		
Condensator 10µF	9451153	/	2	€ 0,05	€ 0,10
Condensator 1µF	9451358	/	1	€ 0,05	€ 0,05
Condensator 470µF	2069249	/	4	€ 0,10	€ 0,39
Condensator 100µF	1902882	/	2	€ 0,05	€ 0,09
Weerstand 33K	1128101	/	1	€ 0,05	€ 0,05
Weerstand 22K	1128090	/	1	€ 0,05	€ 0,05
3,6V batterij connector	/	/	3		
9V batterij connector	/	/	1		
2 Polige connector	2396252	/	2	€ 0,27	€ 0,53
Jumper 4 pins	/	/			

Totaal:	30		€ 141,09

1.2. Schema's

1.2.1. Blokschema

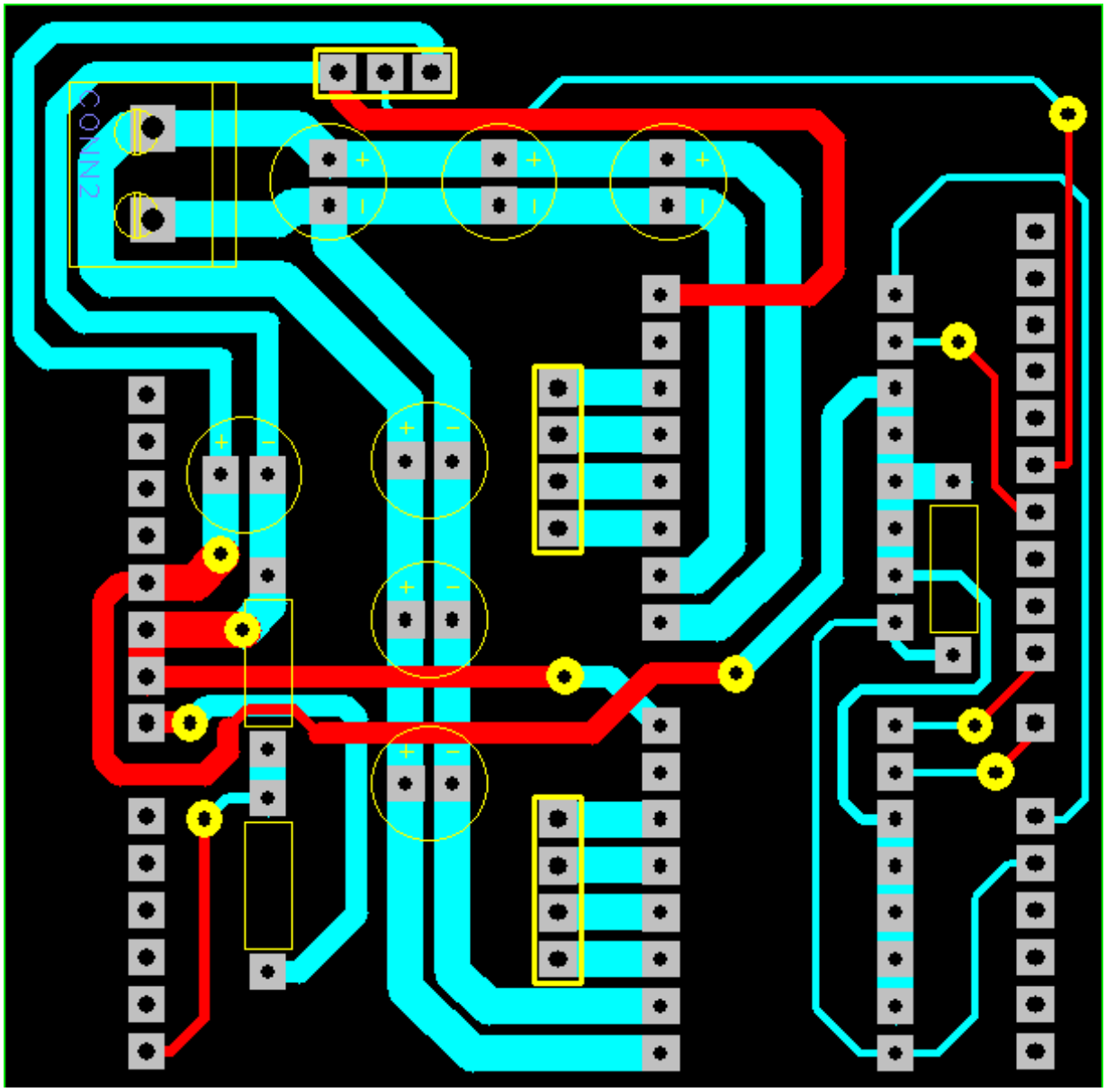


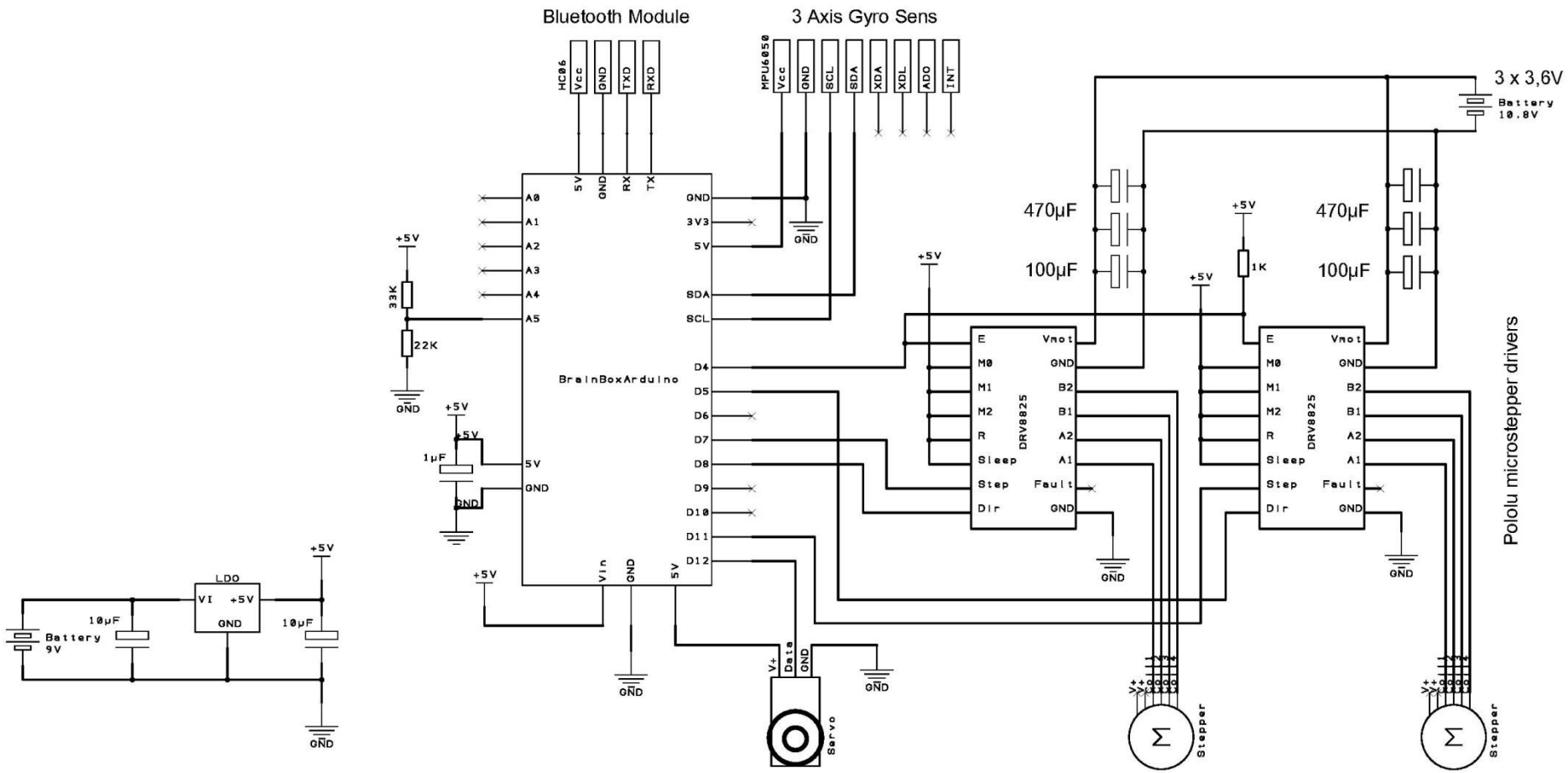
1.2.2. Elektronisch schema

Zie bijgevoegd blad.

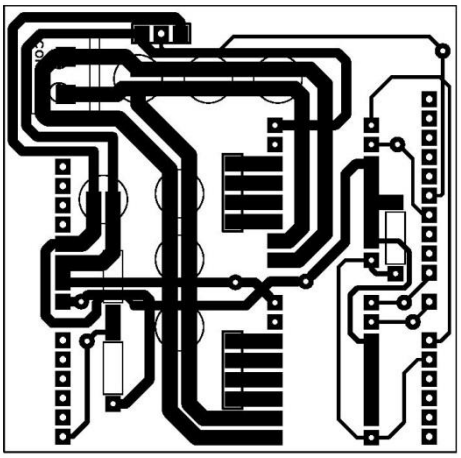
1.2.3. PCB

Zie bijgevoegd blad voor correcte afmetingen.



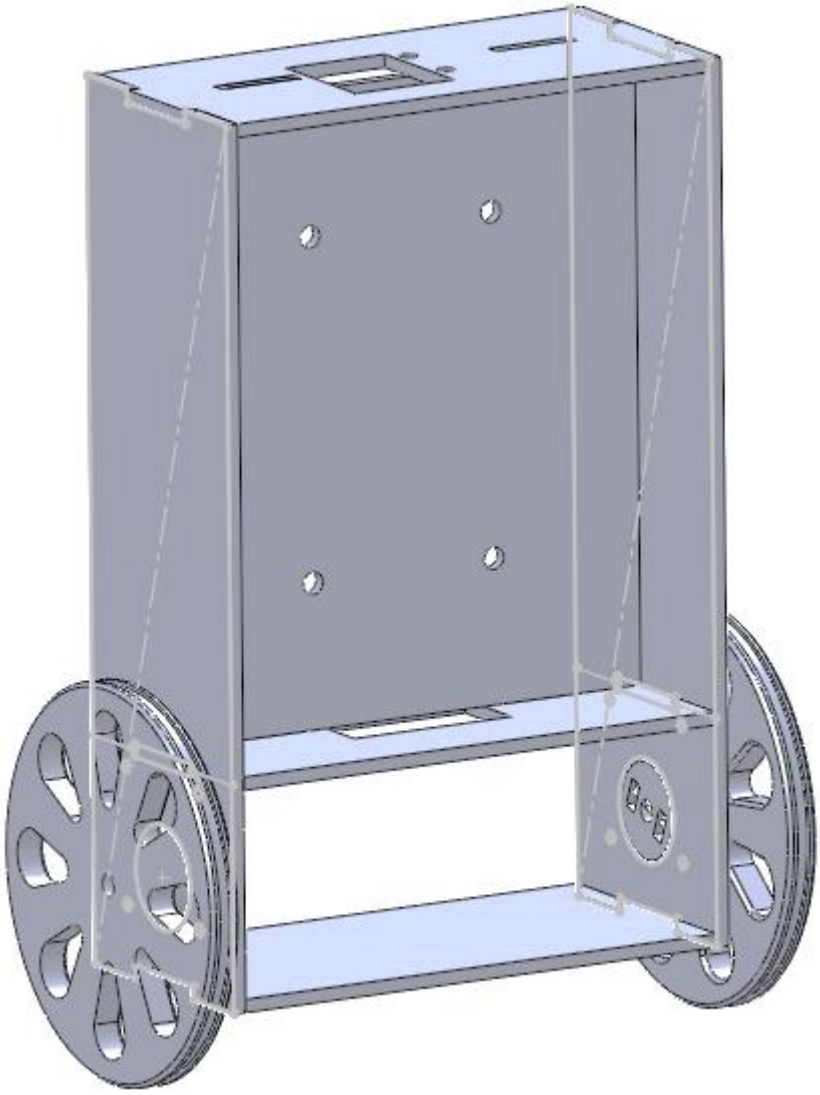


Sint-Jozef Instituut - Schoten - Jozef Hendrickxstraat 153 - 2900 Schoten - Afdeling ELEKTRONICA						Zelfevaluatie: elektrisch correct? ja tekst en bestelnummers? ordelijk? ja
Titel + korte omschrijving project: Elektrisch schema balancerende robot						
Naam: Wannes Op de Beeck	Klas: 6EA-ICT	NR: 6	Datum: 18/05/2017		Behaalde punten: /10	Eigen punten: /10

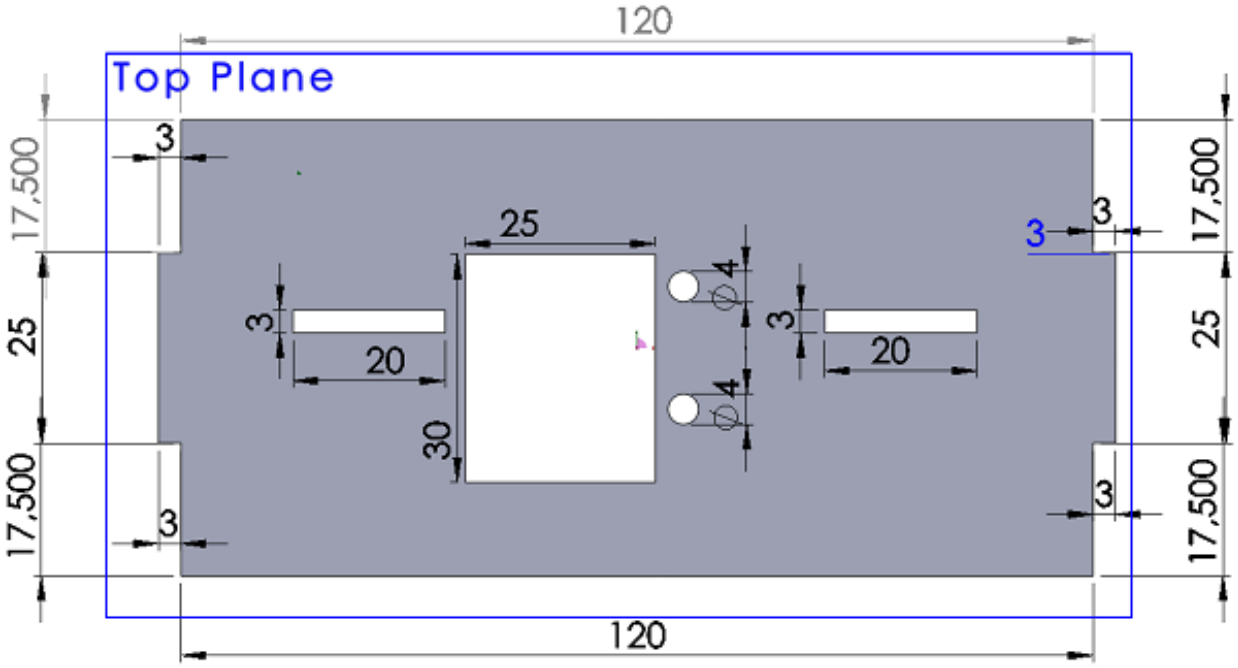


1.2.4. Solidworks

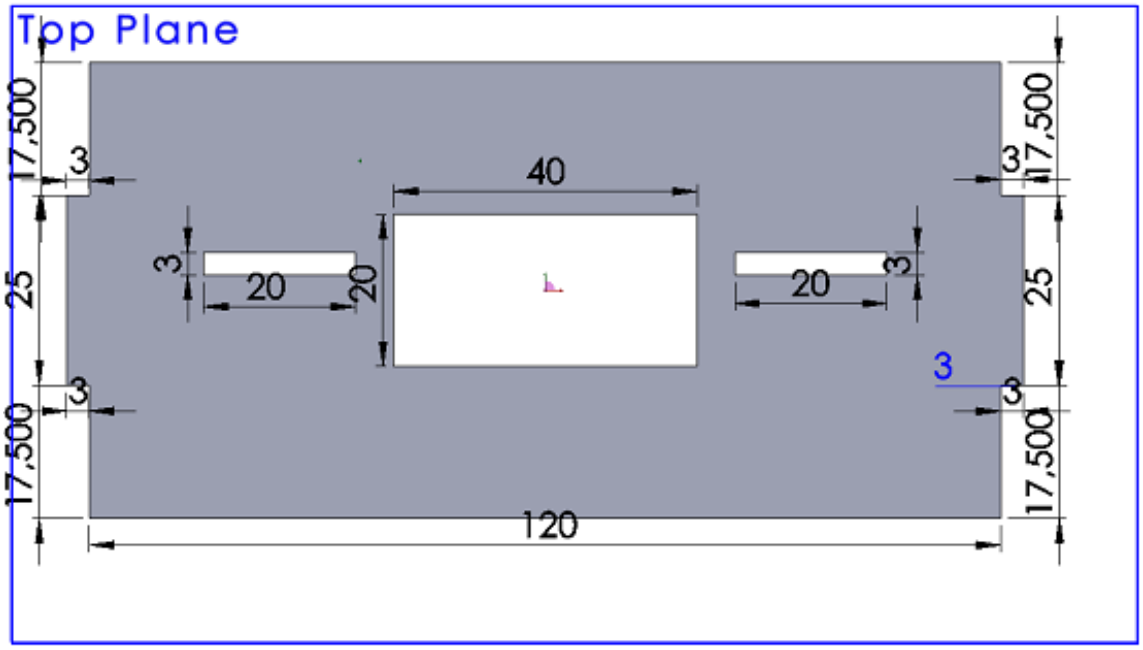
Assembly



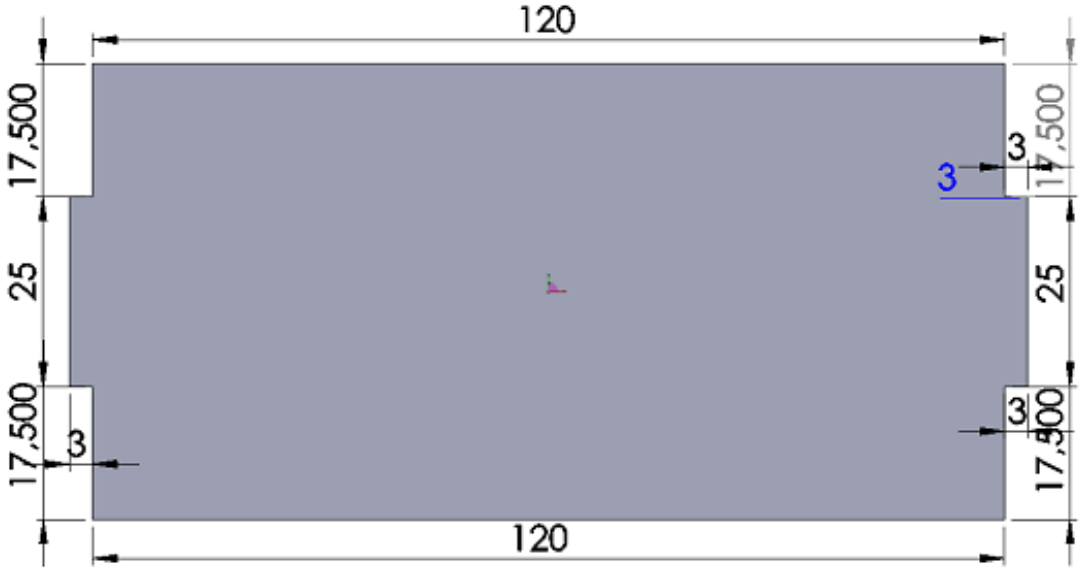
Top plate



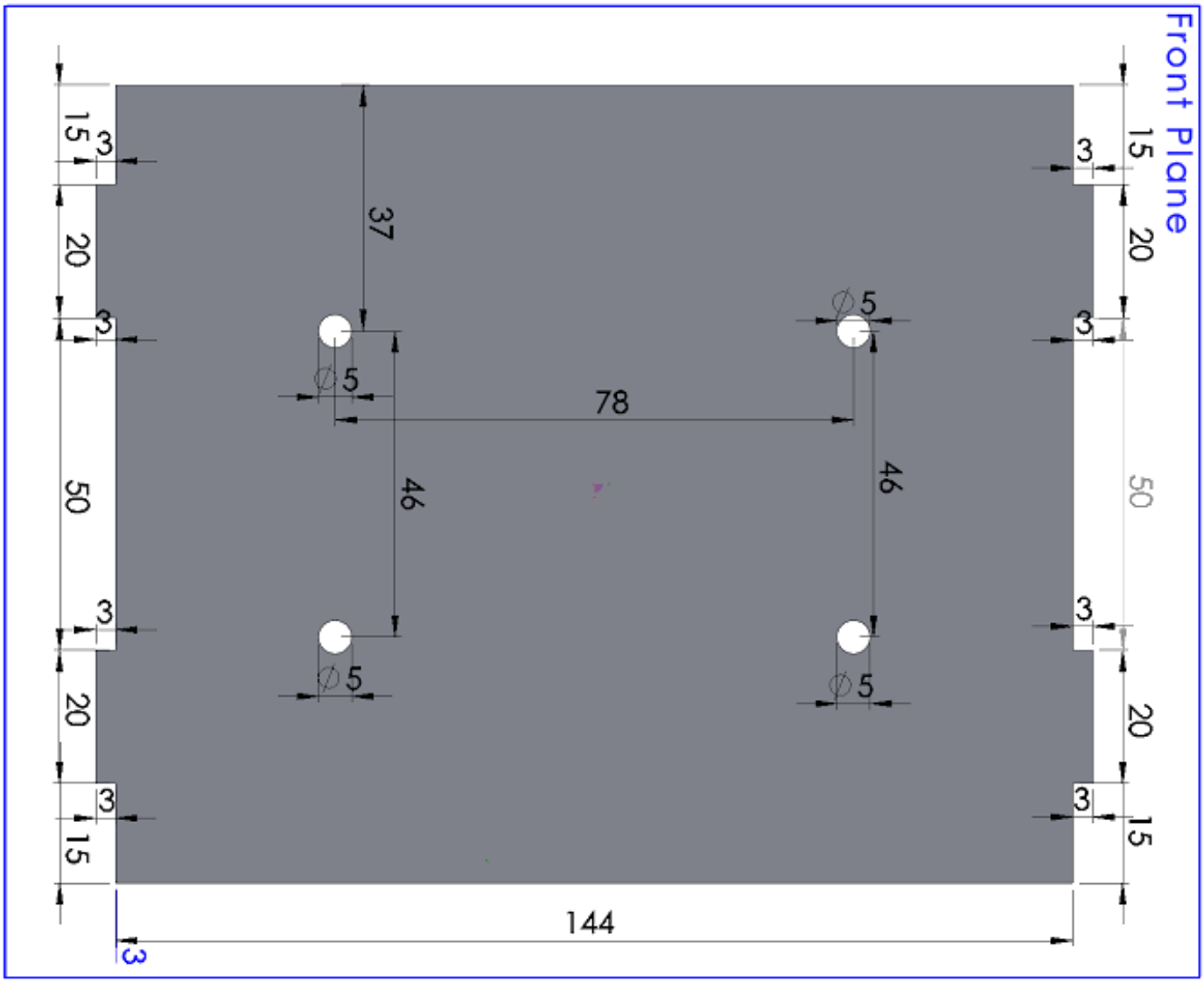
Middle plate



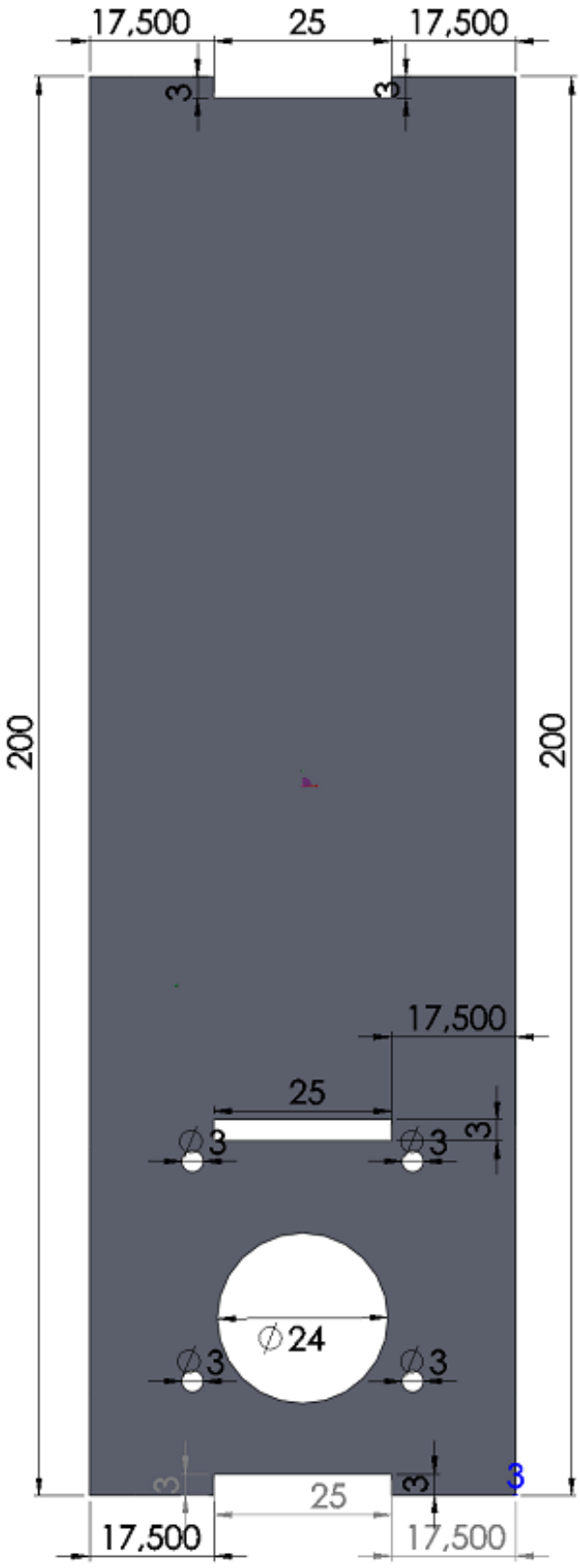
Bottem plate



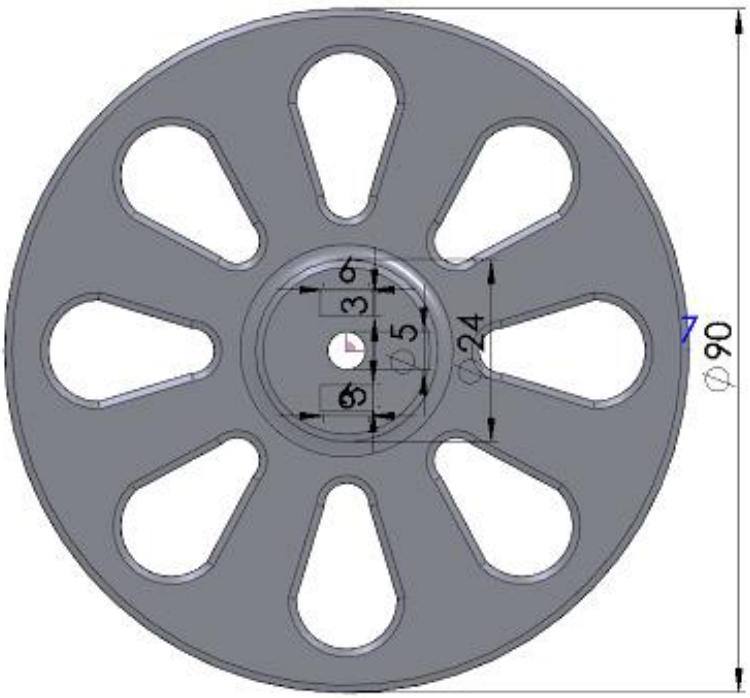
Mount



Side panel



Wheel



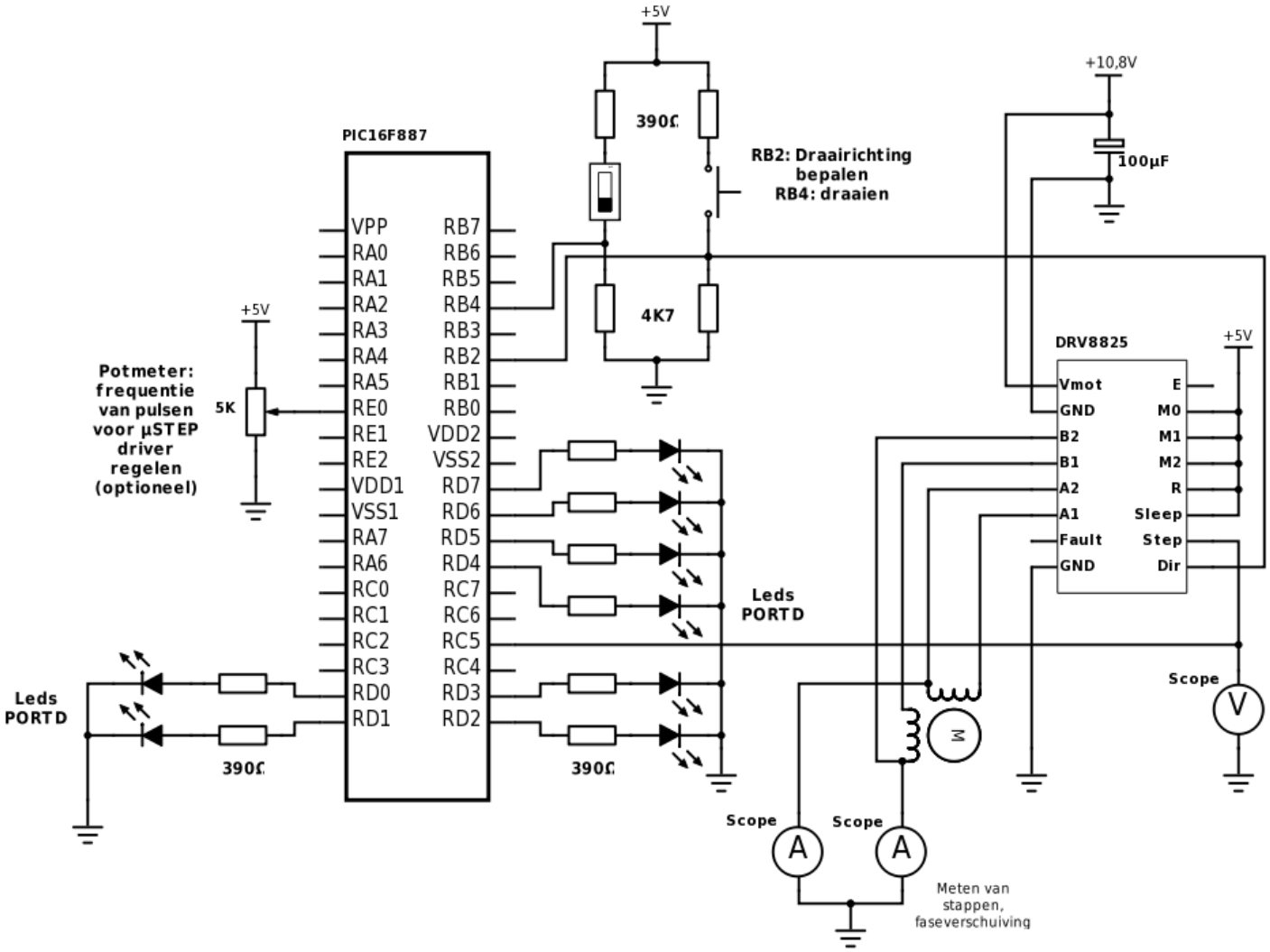
1.3. Labo Stappenmotor en Pololu DRV8825 driver

1.3.1. Deel 1: Driver algemeen en snelheden stappenmotor

Doelstelling

Het doel van dit labo is om met een microcontroller de stappenmotoren van de balancerende robot aan te sturen door gebruik te maken van de Pololu DRV8825 microstepper driver. In dit labo bespreek ik eerst de werking van de DRV8825 en tevens de verschillende snelheden.

Schema



Figuur 1: Elektronisch schema “Labo Stappenmotor” met meetpunten op “Step”-pin en de stappen van de spoelen van de motoren.

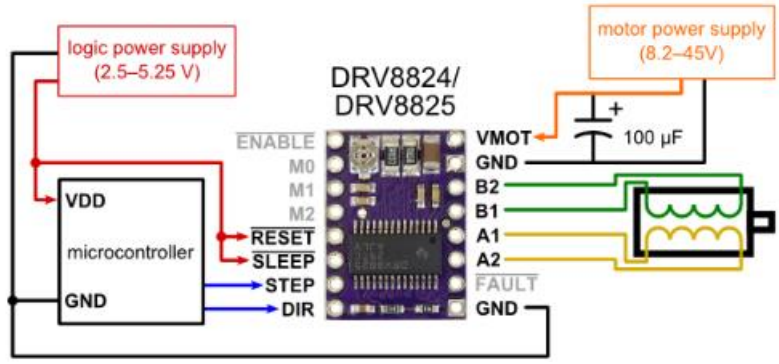
Software

```
#include <htc.h>
#define _XTAL_FREQ 19660800 // Externe klok freq

__CONFIG(0x2FF2);       // Extern XTAL HS; WDT off; PWRT off; MCLR;
                        // BOREN; IESO off; FCMEN off; HVP
__CONFIG(0x3FFF);       // BOR4V; WRToff

delay(){
    for(unsigned char i = 0 ; i < ADRESH ; i++){ // methode om een delay te hebben
met de waarde van potmeter
        __delay_us(1);
    }
}


void main(){
    TRISB = 0x14;                   // switches aan PORTB zijn inputs (microstepper
direction hardwarematig aan RB2 geconnecteerd)
    TRISC = 0x00;                   // RC5 = stepspeed
    TRISD = 0x00;                   // alle pins PORTD = Output - leds
    TRISE = 0x01;                   // E0 - AN5 = input -- A/D potmeter

    ANSEL   = 0b00100000;           // E0/AN5 is een analoge input
    ANSELH  = 0b00000000;           // alle andere AD pins = normal IO
    ADCON0  = 0b01010101;           // Fosc/8 - AN5 - ADON (Go/DONE blijft 0)
    ADCON1  = 0b00000000;           // Meting tussen VDD en VSS- Right justified

    while(1)
    {

        ADCON0 = ADCON0 | 0b00000010;   // GO/DONE = 1 met masker
        while (ADCON0 & 0b00000010){}   // doe niets zolang AD omzetting bezig is

        PORTD = ADRESH;                 // zet AD waarde op leds aan PORTD

        if(RB4){
            RC5 = 1;       // Pulsen met frequentie afhankelijk van potmeterwaarde
            //delay();
            __delay_us(10);     // maximum aan/uittijd om de stappenmotor nog goed
te laten draaien = 10µs
            RC5 = 0;
            //delay();
            __delay_us(10);
        }
    }
}
```

Datasheets/uitleg

Pololu DRV8825 Stepper Motor Driver Carrier, High Current

De motor driver wordt op aangesloten zoals weergegeven in Figuur 2. De meeste aansluitingen zijn vanzelfsprekend, anderen worden hieronder kort toegelicht en beredeneerd.

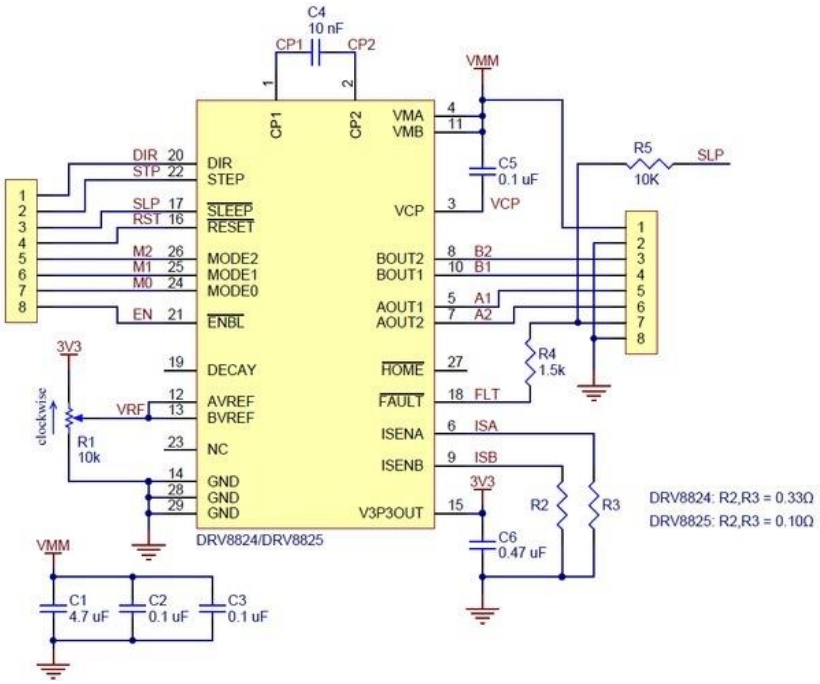


Figuur 2: Aansluitschema Pololu DRV8825.

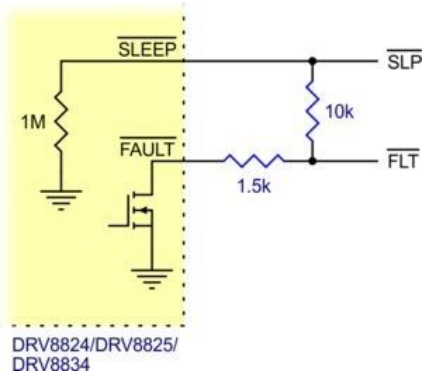
MODE0	MODE1	MODE2	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	1/4 step
High	High	Low	1/8 step
Low	Low	High	1/16 step
High	Low	High	1/32 step
Low	High	High	1/32 step
High	High	High	1/32 step

Tabel 1: Mode's met bijhorende "Microstep Resolutie".

In Tabel 1 zien we de verschillende mogelijkheden om de stappen (resoluties) in te stellen waarmee deze driver mee zou kunnen werken. Door Mode0/1/2 op verschillende manieren aan te sluiten (5V of GND) kunnen deze verschillende "resoluties" gekozen worden.



Figuur 3: Elektronisch schema Pololu DRV8825.



Figuur 4: Elektronisch schema: “SLEEP” en “FAULT” aan elkaar aangesloten.

In Figuur 3 en 4 zien we het schema van de DRV8825. Eén duidelijk gegeven dat we hieruit kunnen halen is dat de $\overline{\text{FAULT}}$ niet aan de 5V of GND mag aangesloten worden. Deze pin is namelijk een output die laag wordt wanneer er een fout optreedt.

Deze pin hangt via een 10k weerstand aan de $\overline{\text{SLEEP}}$ pin, en is dus heel de tijd 5V, wanneer er een fout optreedt, zal de transistor aan deze pin, gaan geleiden om zo de output naar 0V te trekken, daardoor gaat de DRV8825 tevens in sleep-mode.

Saelig PICO TA018 Current Clamp

Figuur 5: Pico TA018 stroom probe

The PP218 and PP264 current clamps are ideal for use with PicoScope automotive oscilloscopes for measuring currents between 10 mA and 60 A. This enables the PicoScope to display current waveforms for fuel injectors and fuel pumps (see waveforms below).

The current clamp has two calibration settings, set by a slider switch on the handle of the probe.

1 mV/10 mA (100 mV = 1 A)

— use this for testing currents up to 20A.

1 mV/100 mA (10 mV = 1 A)

— use this for testing current up to 60A.

De stroom probe in Figuur 5 wordt in dit labo gebruikt om de stappen/faseverschuiving te meten.

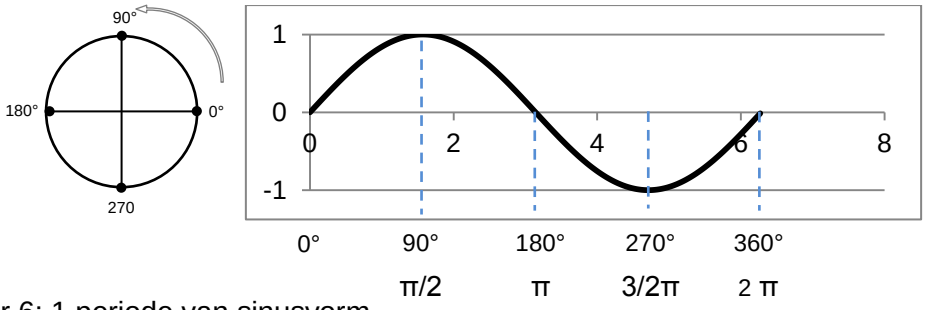
Deze werd ingesteld op 1mV/10mA, dat betekent dat 100mV = 1A.

Omdat we hier stromen meten die véél lager zijn dan 20A, kunnen we deze probe op deze stand zetten.

Scoopbeelden

Resoluties

Een stappenmotor heeft 2 spoelen. De spoelen worden in een bepaalde volgorde aangestuurd om zo de rotor te laten draaien. Deze spoelen worden bekrachtigd met een faseverschuiving van 90°.



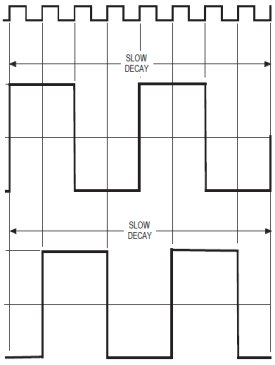
Figuur 6: 1 periode van sinusvorm.

Full step

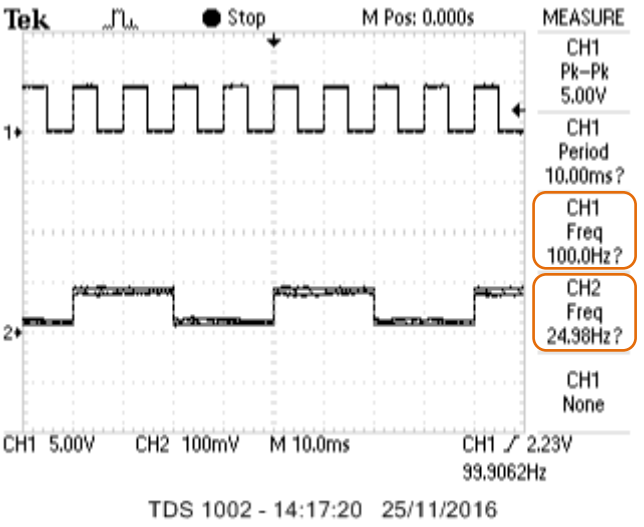
MODE0	MODE1	MODE2	Microstep Resolution
Low	Low	Low	Full step

Tabel 2: Resolution: Full step

In Tabel 2 zijn mode0/1/2 allemaal aan de GND aangesloten, hierdoor werken we nu in full step.
Door pulsen (CH1) te geven met de microcontroller op de STEP pin, worden er stappen (CH2) gegenereerd door de µstep driver.
Bij elke puls wordt er een stap gezet.
Dit is duidelijk in Figuur 7.



Figuur 7: Full step.



Figuur 8: Scoopbeeld “Full step”.

Figuur 8 toont dat de frequentie van het “stapsignaal” 1/4 is van het puls signaal afkomstig van de microcontroller.

Omdat ik geen 3 signalen op één scoopbeeld kon weergeven, en omdat we vorig jaar reeds de verschillende methodes (full step, half step e.d.) gezien hebben, leek het me voldoende de pulsen en de stroom door 1 spoel weer te geven.

Half step

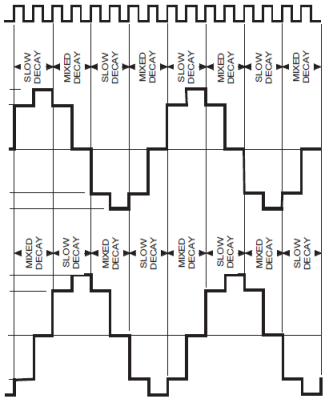
MODE0	MODE1	MODE2	Microstep Resolution
High	Low	Low	Half step

Tabel 3: Resolution: Half step

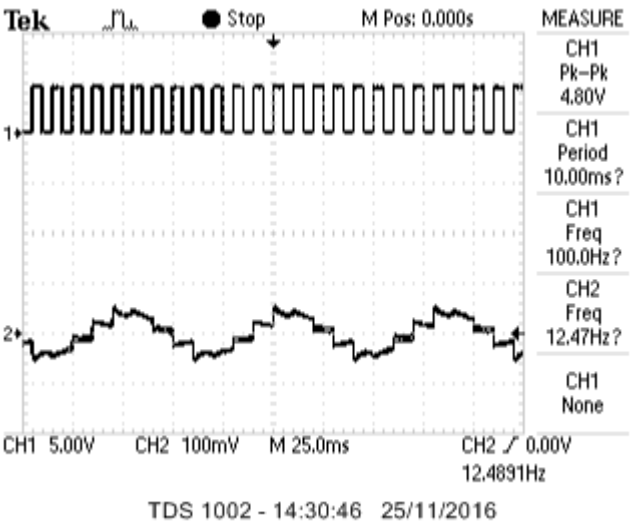
In Tabel 3 zien we dat mode0 aan de 5V aangesloten is, de andere (1 en 2) zijn aan de GND aangesloten, hierdoor werken we nu in half step.

Bij half step zie je dat er al een lichte sinusvorm te zien wordt, maar deze is nog heel gekarteld

In Figuur 9 is duidelijk te zien dat fase 1 en fase 2 90° in fase verschoven zijn ten opzichte van elkaar.



Figuur 9: Half step.



Figuur 10: Scoopbeeld “Half step”.

Het scoopbeeld in Figuur 10 geeft weer dat de frequentie van het “stapsignaal” 1/8 is, van het puls signaal, afkomstig van de microcontroller.

1/4 step

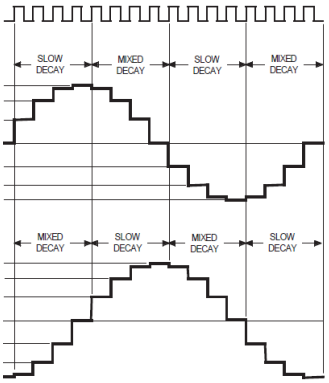
MODE0	MODE1	MODE2	Microstep Resolution
Low	High	Low	1/4 step

Tabel 4: Resolution: 1/4 step

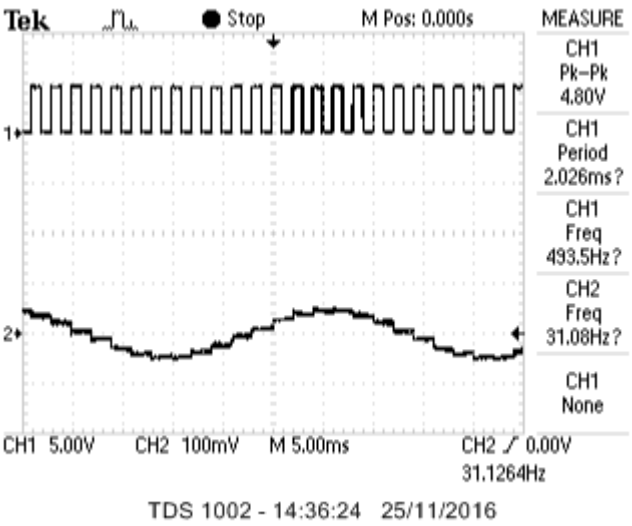
In Tabel 4 is Mode1 aan de 5V aangesloten, de andere (0 en 2) aan de GND. Hierdoor werken we nu in 1/4 step.

In Figuur 12 wordt de sinusvorm duidelijker en de kartelingen worden kleiner.

Figuur 11 toont dat fase 1 en fase 2 90° verschoven zijn ten opzichte van elkaar.



Figuur 11: 1/4 step.



Figuur 12: Scoopbeeld “1/4 step”.

We zien in Figuur 12 dat de frequentie van het “stapsignaal” 1/16 is van het puls signaal, afkomstig van de microcontroller.

$$31 * 16 \approx 494$$

1/8 step

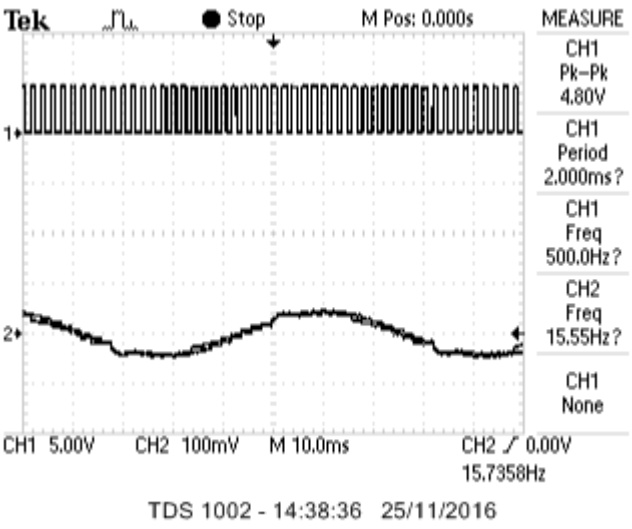
MODE0	MODE1	MODE2	Microstep Resolution
High	High	Low	1/8 step

Tabel 5: Resolution: 1/8 step.

We zien in Tabel 5 dat Mode0 en 1 beiden aan de 5V aangesloten zijn. Mode2 is aangesloten aan de GND, hierdoor werken we nu in 1/8 step.

In Figuur 14 verkrijgen we nu een duidelijke sinusvorm met kleine bekjes.

We kunnen er nu wel vanuit gaan dat fase 1 en fase 2, 90° verschoven zijn ten opzichte van elkaar.



Figuur 13: Scoopbeeld “1/8 step”.

Figuur 13 toont dat de frequentie van het “stapsignaal” 1/32 is van het signaal, afkomstig van de microcontroller.

$$15{,}55 * 32 \approx 500$$

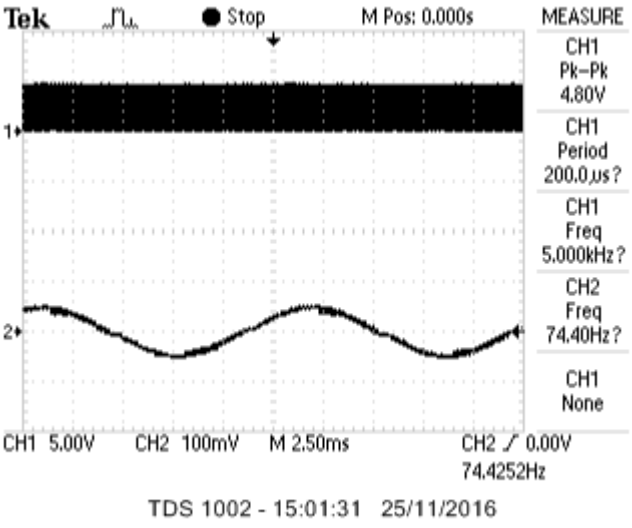
1/16 step

MODE0	MODE1	MODE2	Microstep Resolution
Low	Low	High	1/16 step

Tabel 6: Resolution: 1/16 step.

In Tabel 6 is Mode2 aan de 5V aangesloten, de andere (0 en 1) zijn aan de GND aangesloten, hierdoor werken we nu in 1/16 step.

De sinusvorm wordt duidelijker en de bekjes zijn bijna niet meer zichtbaar, dat zien we in Figuur 15.



Figuur 14: Scoopbeeld “1/16 step”.

De frequentie van het “stapsignaal” is in Figuur 14, 1/64 van het puls signaal, afkomstig van de microcontroller.

$$74{,}5 * 64 \approx 5000$$

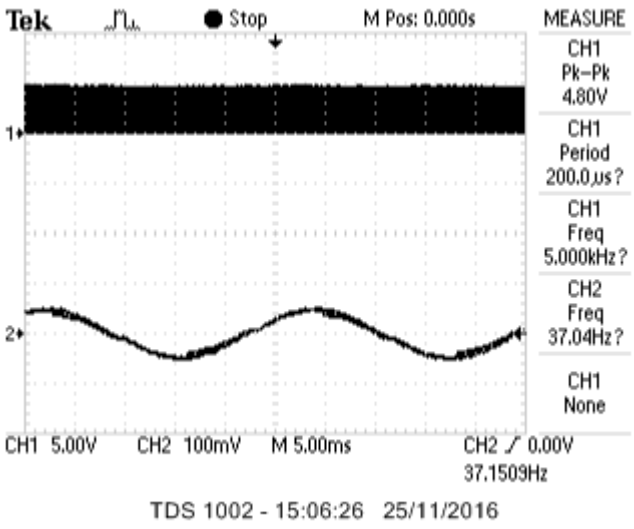
1/32 step

MODE0	MODE1	MODE2	Microstep Resolution
High	Low	High	1/32 step

Tabel 7: Resolution: 1/32.

Tabel 7 toont dat mode0 en 2 aan de 5V zijn aangesloten. Mode1 is aan de GND aangesloten, en hierdoor werken we nu in 1/32 step.

We zien in Figuur 15 nu een quasi perfecte sinusvorm, de bekjes zijn niet meer zichtbaar door de hoge pulsfrequentie.



Figuur 15: Scoopbeeld “1/32 step”.

De frequentie van het “stapsignaal” is 1/128 van het puls signaal afkomstig van de microcontroller.

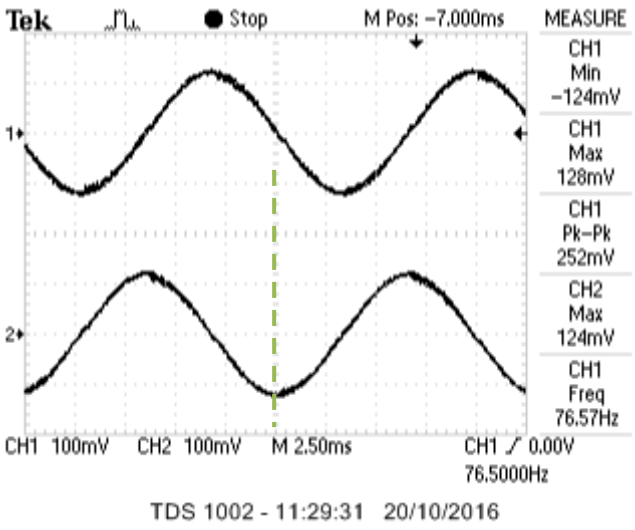
$$37 * 128 \approx 5000$$

Snelheden

In dit onderdeel van het labo, bespreek ik de maximale frequentie van de signalen op de twee spoelen op verschillende snelheden. De microstepper driver staat op de kleinste stand (1/32 step). Op deze manier ben ik te weten gekomen wat de maximum snelheid om pulsen te sturen is, om de motor nog “goed” te laten functioneren.

In het eerste beeld in Figuur 16 geef ik pulsen van 50µs aan- en uittijd (frequentie = 1/100µs = 10kHz). Je ziet hier twee mooie sinussen die 90° in fase verschoven zijn ten opzichte van elkaar.

De frequentie van het gemeten signaal is 76,57Hz.

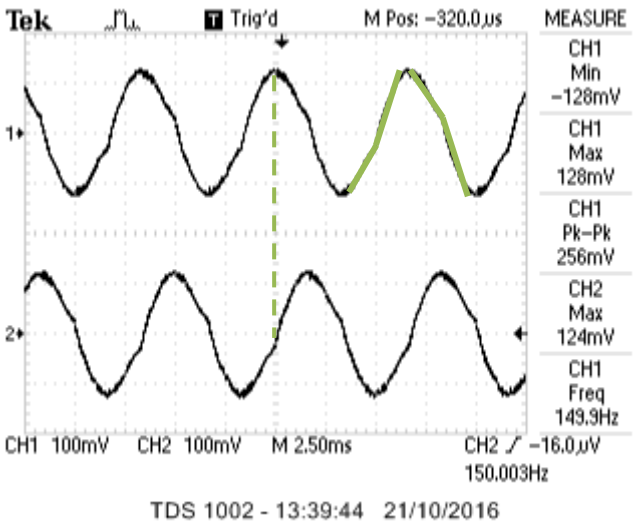


Figuur 16: Scoopbeeld snelheid: 10kHz.

In deze situatie geef ik pulsen van 25µs (frequentie = 1/50µs = 20kHz), dubbel zo snel dus. Je ziet dat er al een lichte knik in het signaal komt en dus geen perfect mooie sinus meer is.

In Figuur 17 is te zien dat de twee signalen wel mooi 90° in fase verschoven zijn, ten opzichte van elkaar. **Ik heb in dit beeld de twee fasen omgedraaid (CH1 uit vorige situatie is hier CH2).**

De frequentie is hier 149,9Hz, ongeveer dubbel zo snel als bij de eerste situatie.



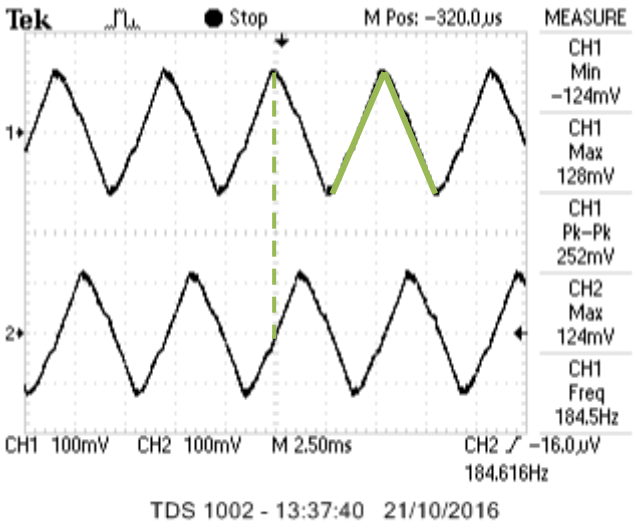
Figuur 17: Scoopbeeld snelheid: 20kHz.

In Figuur 18 stuur ik pulsen van 20µs (frequentie = 1/40µs = 25kHz), dat is maar "iets" sneller als daarnet. We zien dat de spoelen, door hun inductie, de frequentie niet meer kunnen volgen waardoor er geen sinus meer is, maar eerder een driehoek vorm.

De signalen zijn wel nog steeds 90° in fase verschoven ten opzichte van elkaar.

De frequentie is hier 184.5Hz.

$$\frac{25}{20} = 1{,}25 \approx \frac{184{,}5}{149{,}9}$$



Figuur 18: Scoopbeeld snelheid: 25kHz.

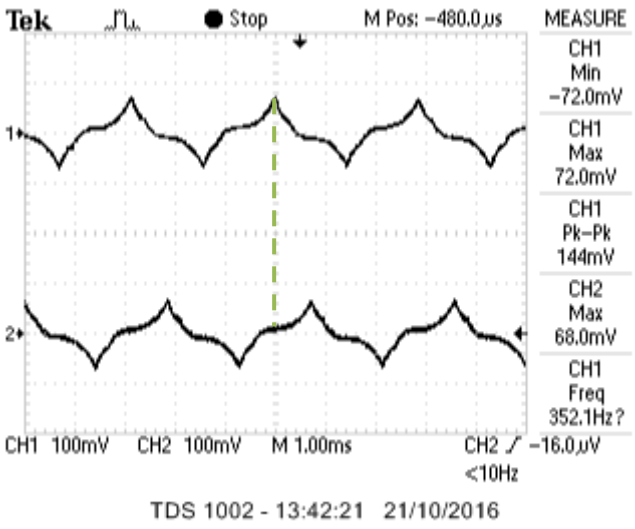
Om tot het uiterste te gaan: Figuur 19.

Hier geef ik pulsen van 10µs (freq = 1/20µs = 50kHz).

Op dit moment draait de motor op zijn snelst, de amplitude van het signaal is veel kleiner geworden.

Als ik nog vlugger pulsen geef, dan wordt de amplitude te klein waardoor de motor niet genoeg kracht meer heeft om vloeiend te kunnen draaien.

Hier zie je dat de inductie van de spoelen ervoor zorgt dat de stroom door de spoelen niet snel genoeg kan stijgen. De knik in het signaal die vooral duidelijk was in het beeld van de pulsen van 25µs, is hier een piek geworden.



Figuur 19: Scoopbeeld snelheid: 50kHz.

Nog steeds wel mooi 90° in fase verschoven t.o.v. elkaar.

PIC16F887 registers

ANSEL register

REGISTER 3-3: ANSEL: ANALOG SELECT REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
ANS7(2)	ANS6(2)	ANS5(2)	ANS4	ANS3	ANS2	ANS1	ANS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **ANS<7:0>**: Analog Select bits
Analog select between analog or digital function on pins AN<7:0>, respectively.
1 = Analog input. Pin is assigned as analog input(1).
0 = Digital I/O. Pin is assigned to port or special function.

Figuur 20: Datasheet ANSEL register

Gegeven: Figuur 20 stellen we AN5 (E0) in als analoge input pin, om de analoge waarde van de potmeter te kunnen inlezen.

ADCON0 register

REGISTER DEFINITIONS: ADC CONTROL

REGISTER 9-1: ADCON0: A/D CONTROL REGISTER 0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

bit 7-6 **ADCS<1:0>:** A/D Conversion Clock Select bits
00 = Fosc/2
01 = Fosc/8
10 = Fosc/32
11 = FRC (clock derived from a dedicated internal oscillator = 500 kHz max)

bit 5-2 **CHS<3:0>: Analog Channel Select bits**
0000 = AN0
0001 = AN1
0010 = AN2
0011 = AN3
0100 = AN4
0101 = AN5
0110 = AN6
0111 = AN7
1000 = AN8
1001 = AN9
1010 = AN10
1011 = AN11
1100 = AN12
1101 = AN13
1110 = CVREF
1111 = Fixed Ref (0.6V fixed voltage reference)

bit 1 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
1 = A/D conversion cycle in progress. Setting this bit starts an A/D conversion cycle.
This bit is automatically cleared by hardware when the A/D conversion has completed.
0 = A/D conversion completed/not in progress

bit 0 **ADON:** ADC Enable bit
1 = ADC is enabled
0 = ADC is disabled and consumes no operating current

Figuur 21 toont de datasheet met de stellingen om de AD conversie op punt te krijgen. Hierbij kiezen we voor een interrupt-frequentie van Fosc/8, we zetten uiteraard AN5 als analoge input, en we moeten natuurlijk ook de AD conversie “enablen” door de ADC enable bit op 1 te zetten. De GO/DONE bit wordt in het programma op 1 en op 0 gezet om de AD conversie te laten gebeuren en terug af te zetten.

Figuur 21: Datasheet ADCON0 register.

Besluiten

De software die ik geschreven heb om dit labo op punt te zetten is vrij simpel, maar ik denk dat bij een labo de hardware en het testen ervan op de eerste plaats komt. Dit is echter anders bij een project van ICT.

Tijdens dit labo heb ik veel bijgeleerd over de DRV8825 microstepper driver van Pololu en de werking ervan. Hij is niet moeilijk aan te sluiten en aan te sturen. Maar zeker wel onmisbaar om in het geval van de robot, vloeiende en nauwkeurige bewegingen te kunnen maken.

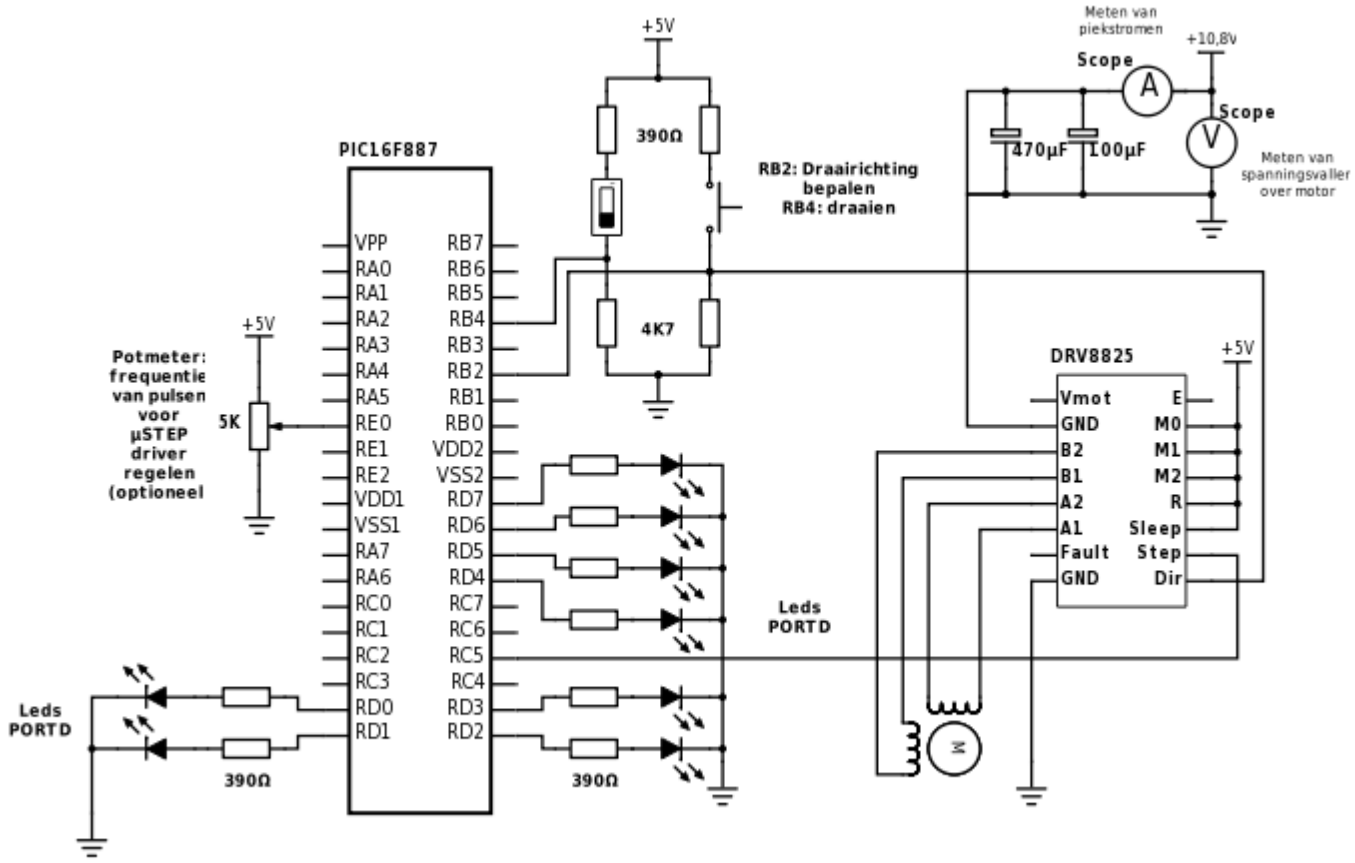
Ik heb ook geleerd dat een stappenmotor vanaf een bepaalde snelheid niet meer naar behoren functioneert. Dit komt door de inductie van de spoelen, en dat de stroom daardoor niet snel genoeg kan stijgen en dalen.

1.3.2. Deel 2: Piekstromen en spanningsdrops

Doelstelling

Het doel van dit labo is om met de Pololu DRV8825 microstepper driver, de stappenmotoren van de balancerende robot aan te sturen. Hierbij is het de belangrijk om te zien hoeveel stroom deze motoren trekken en hoeveel de spanning inzakt in extreme situaties. Ook dient deze schakeling aangepast te worden zodat deze stroompieken en spanningsdrops kunnen worden opgevangen. De code en datasheets in dit deel zijn hetzelfde als die van het vorige deel.

Schema



Figuur 22: Elektronisch schema “Labo Stappenmotor” met spannings- en stroommetingen op voedingspinnen.

Saelig PICO TA018 Current Clamp (zie Figuur 5)

Deze stroom probe wordt in dit labo gebruikt om de stroompieken te meten. Deze werd ingesteld op 1mV/10mA. Omdat we hier stromen meten die véél lager zijn dan 20A, kunnen we deze probe op deze stand zetten.

Scoopbeelden

Stroompieken en spanningsdrops

Condensator 100µF en 470µF

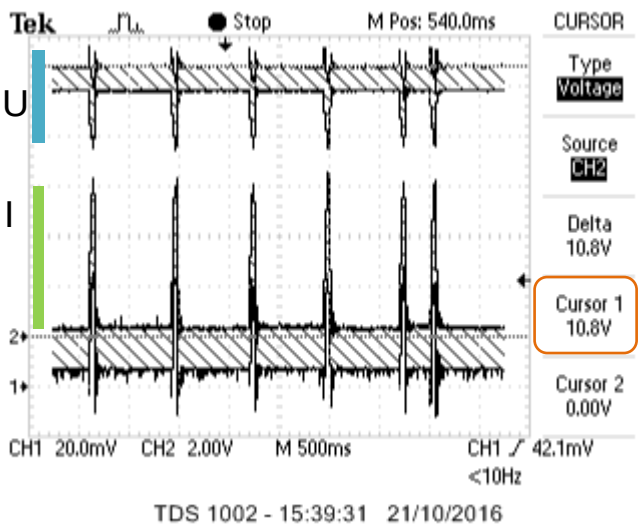
Op CH1 in Figuur 23 meet ik de stroom, en dus ook de piekstromen als de motor in extreme situaties verkeert (puls van 10µs, vooruit achteruit vooruit achteruit...).

Op CH2 wordt de spanning gemeten (en dus ook de spanningsdrops).

Deze spanningdrops dienen zoveel mogelijk weggewerkt te worden door middel van de juiste condensatoren te gebruiken.

Ik heb gekozen voor een bronspanning van 10,8V, dit zou in principe de drie 3,6V batterijen simuleren.

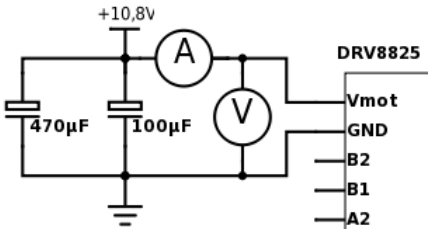
$$3{,}6 * 3 = 10{,}8V$$



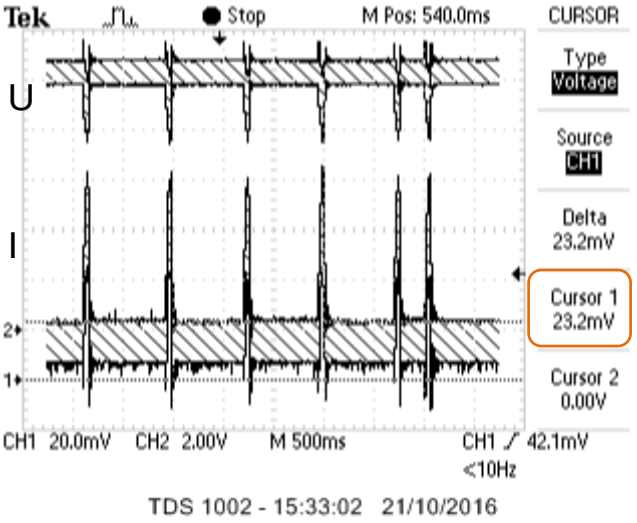
Figuur 23: Scoopbeeld bronspanning.

De stroom werd gemeten met een PICO TA018 stroom probe. In de datasheet staat dat 1mV overeen komt met 10mA. Factor 10 dus.

Uit Figuur 25 kunnen we afleiden dat de stroom als de motor gewoon draait, 232mA is.



Figuur 24: Schakeling 100µF en 470µF.



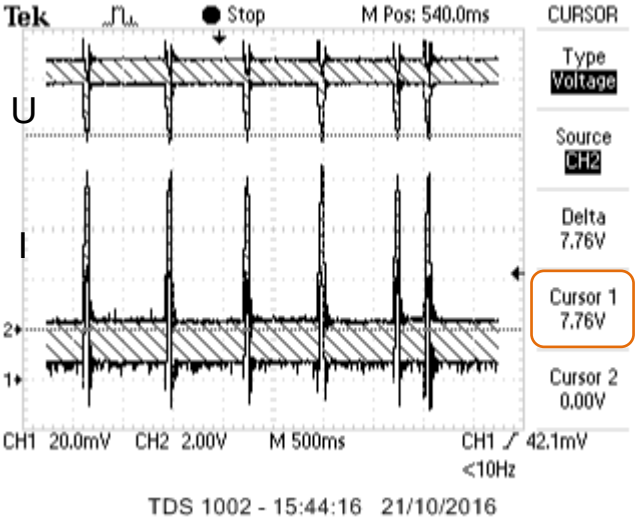
Figuur 25: Scoopbeeld stroom bij normaal bedrijf 100µF en 470µF.

Wanneer de draaizin van de motor plots verandert, doen er zich stroompieken voor die zorgen voor spanningsdrops.

In Figuur 26 worden de spanningsdrops gemeten. Met deze condensatoren resulteert dit in drops tot 7,76V

$$10{,}8 - 7{,}76 = 3{,}04V$$

Een spanningsdrop van 3,04V dus. Dat is te veel dus moeten we andere condensatoren zoeken.

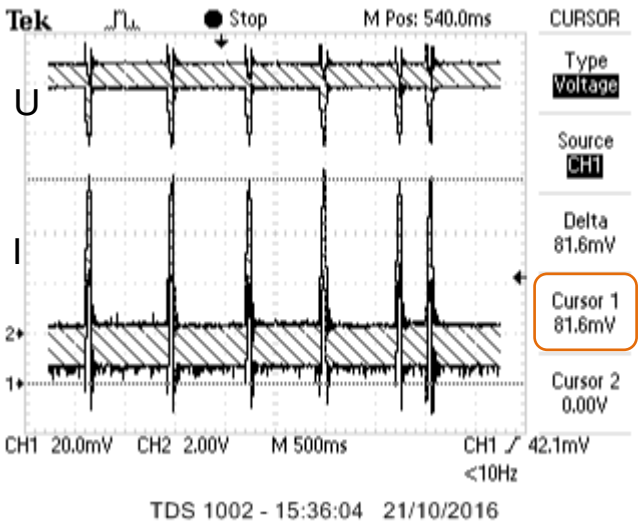


Figuur 26: Scoopbeeld spanningsdrops 100µF en 470µF.

Op CH1 in Figuur 27, meet ik de stroompieken als de motor plots van draaizin verandert.

Deze stroompieken bedragen bijna 1 ampère, namelijk 816mA. Dit is heel veel voor gewone batterijen. Daarom ga ik speciale lithium ion batterijen gebruiken.

In dit labo heb ik niet met deze batterijen gewerkt, maar ik ging ervan uit dat de beste situatie met de voeding, ook de beste situatie met de batterijen zou zijn.



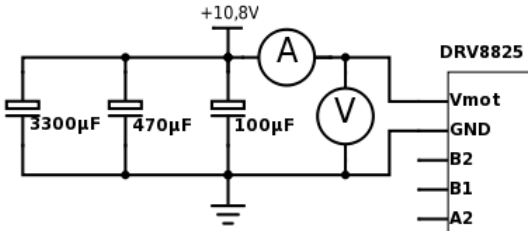
Figuur 27: Scoopbeeld stroompieken 100µF en 470µF.

Condensator 100µF, 470µF en 3300µF

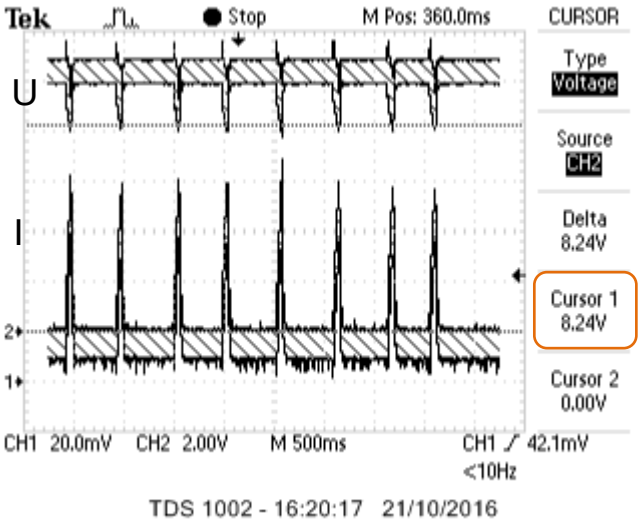
Om de spanningsdrop een beetje op te lossen probeer ik verschillende condensatoren parallel bij te plaatsen. Met een condensator van 3300µF is er al enige verbetering. De spanning zakt in Figuur 29 in tot 8,24V.

$$10{,}8 - 8{,}24 = 2{,}56V$$

Een drop van 2,56V dus.



Figuur 28: Schakeling 100µF, 470µF en 3300µF.



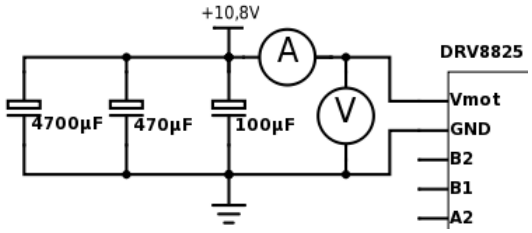
Figuur 29: Scoopbeeld spanningsdrops 100µF, 470µF en 3300µF.

Condensator 100µF, 470µF en 4700µF

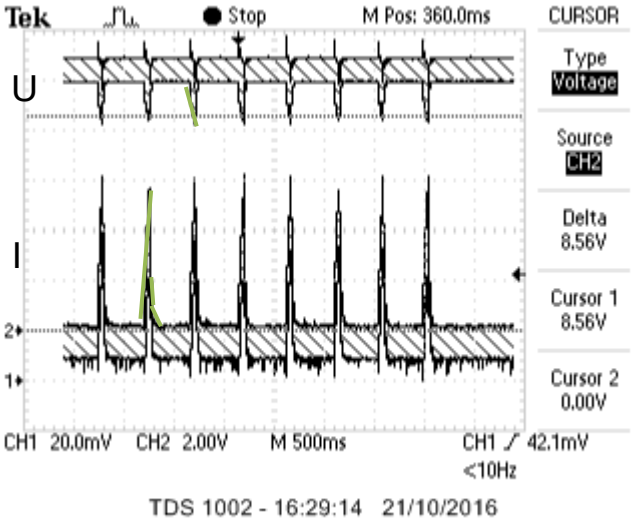
In Figuur 31 zien we: de spanningsdrop met een condensator van 4700µF bedraagt:

$$10{,}8 - 8{,}56 = 2{,}24V$$

Een nog grotere condensator verbetert niet veel, de spanningsdrop is echter iets kleiner, maar je ziet dat deze “grote” condensatoren eigenlijk te traag zijn, door de afronding van de pieken (aangeduid).



Figuur 30: Schakeling 100µF, 470µF en 4700µF.



Figuur 31: Scoopbeeld spanningsdrops 100µF, 470µF en 4700µF.

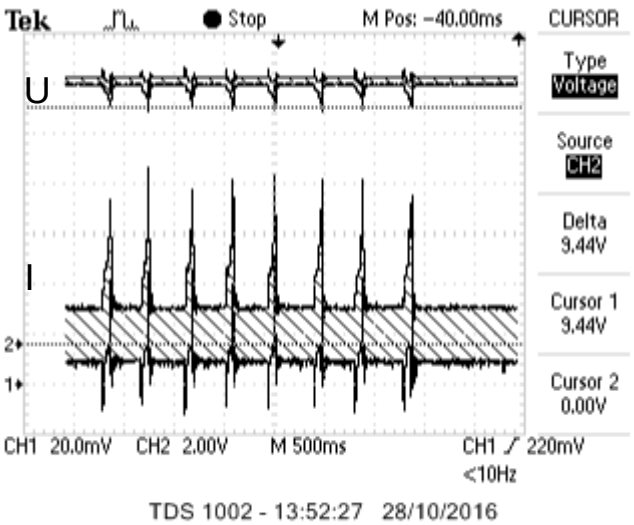
Condensator 100µF, 470µF en 470µF

Ik koos er uiteindelijk voor om een “kleinere” condensator van 470µF te gebruiken. In Figuur 32 is te zien dat het probleem opgelost is.

Hier is de dropt de spanning tot 9,44V.

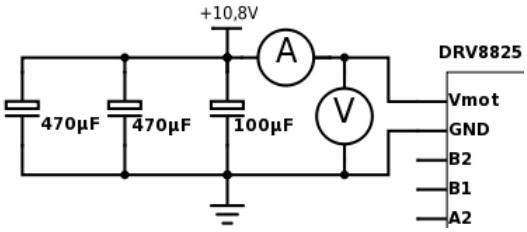
$$10{,}8 - 9{,}44 = 1{,}36V$$

Door er nog andere condensatoren bij te plaatsen, wordt het niet speciaal veel beter. Eén condensator van 100µF en twee van 470µF, parallel over elkaar, lijken de beste oplossing.



Figuur 32: Scoopbeeld spanningsdrops 100µF, 470µF en 470µF.

Eindresultaat:



Figuur 33: Schakeling 100µF, 470µF en 470µF.

Besluiten

Bij het testen ben ik eerst naar kleinere condensatoren gegaan, maar daar veranderde echter niets aan het beeld, dus ben ik groter gegaan. De extreem grote condensatoren deden wel iets, maar ik zag dat ze eigenlijk al te traag waren aan de manier waarop het beeld gevormd werd. Toen had ik thuis even de tijd genomen om eens logisch over deze metingen na te denken en besloot een 470µF condensator (dezelfde die ik er al standaard had bijgezet) nog extra parallel over te plaatsen. Hiermee leek het probleem opgelost te zijn.

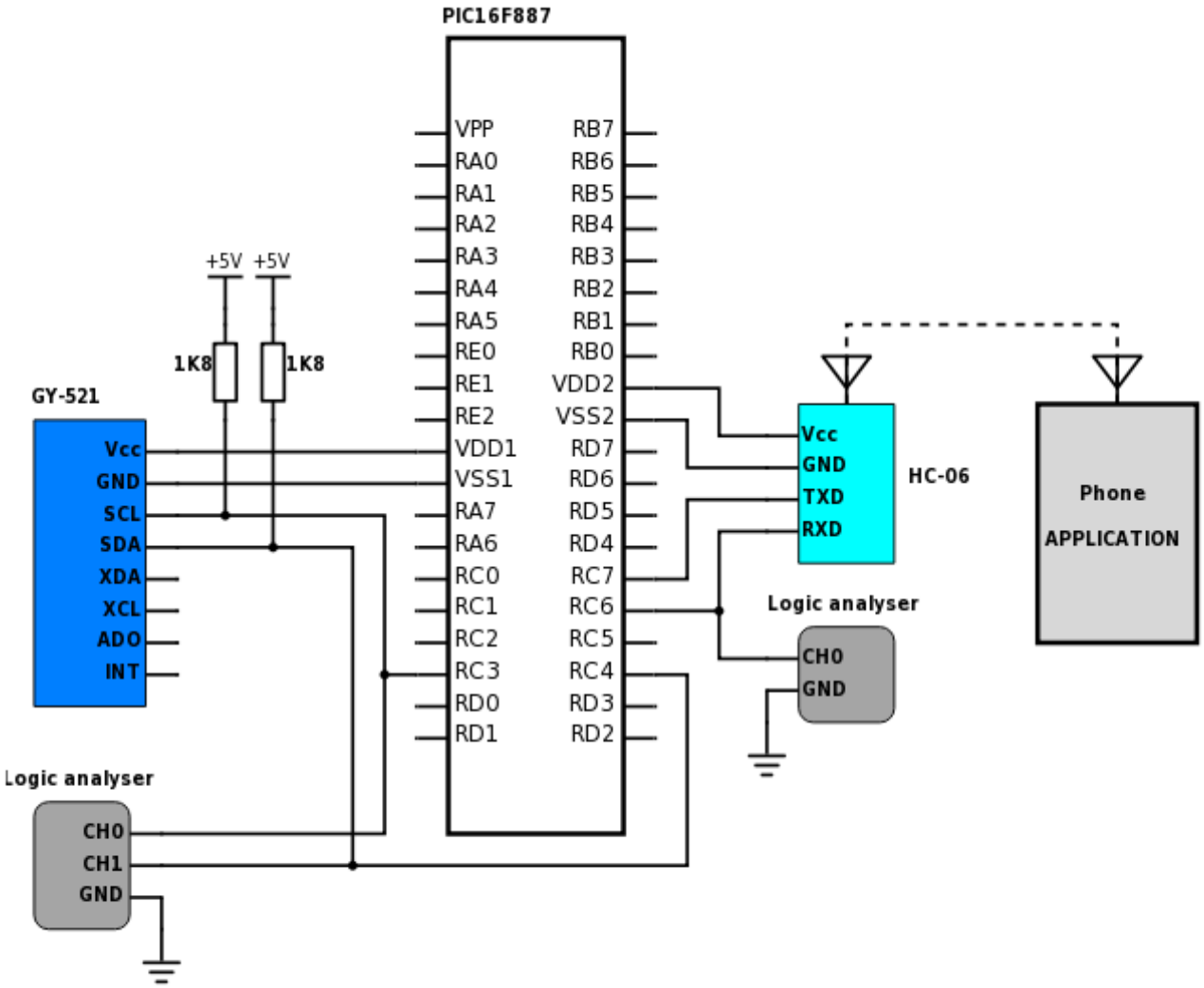
De reden waarom ik over de stappenmotor twee aparte verslagen gemaakt heb, is omdat dit over een totaal verschillend onderwerp gaat. De code, datasheets en dergelijke van beide verslagen zijn hetzelfde, daarom heb ik deze niet dubbel gebruikt.

1.4. Labo GY-521 met MPU-6050 gyrosensor & accelerometer

Doelstelling

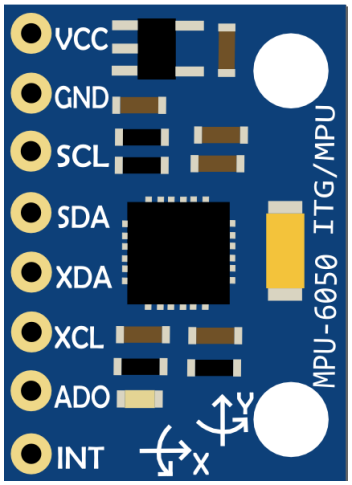
Het doel van dit labo is om aan te tonen dat ik met de GY-521 kan communiceren met I²C commando's en de uitgelezen data kan begrijpen. In dit labo lees ik verschillende waarden van de verschillende registers (accelero, gyro, en temperatuur) in. Deze geef ik vervolgens weer op grafieken op een smart device. De bedoeling hiervan is om per as te kijken hoe de gyrosensor en de accelerometer reageren en zo uit te zoeken in welke situaties, welke sensor het best gebruikt kan worden.

Schema

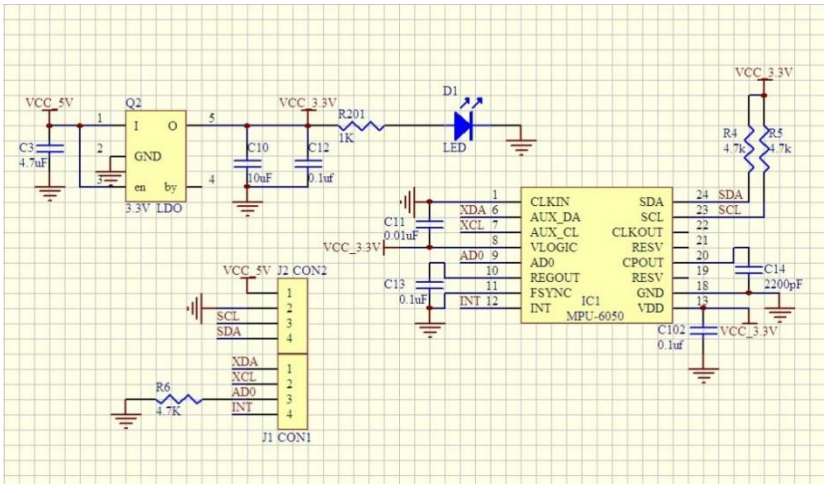


Figuur 34: Elektronisch schema om metingen te doen op de GY-521 met MPU-6050 gyrosensor-accelerometer.

GY-521 breakout board met MPU-6050 gyro sensor en accelerometer

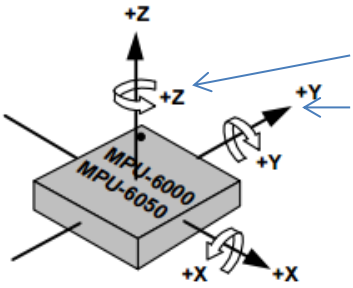


Figuur 35: Lay-out van het GY-521 breakout board.



Figuur 36: Elektronisch schema GY-521 breakout board met MPU-6050 gyrosensor-accelerometer.

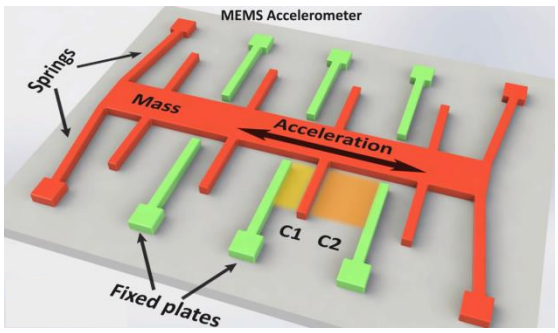
We sturen de sensor aan via I²C, dat betekent dat we enkel de Vcc (5V), GND, SCL en de SDA lijnen gebruiken. De andere lijnen kunnen gebruikt worden om deze sensor aan te sturen via SPI. De MPU-6050 gyro/accelero meet op deze manier:



De gyrosensor meet draaiingen t.o.v de zwaartekracht.
De accelerometer meet lineaire bewegingsverschillen.

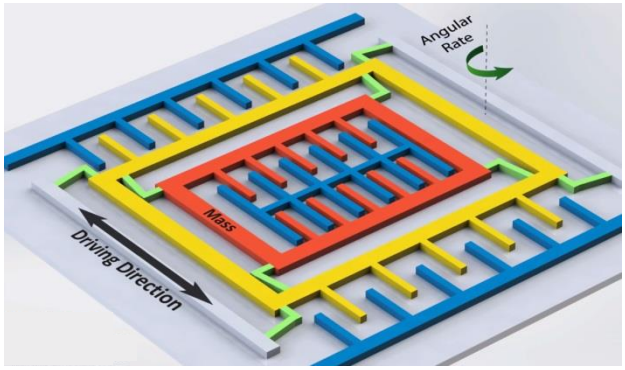
Figuur 37: Voorstelling uit de datasheet van hoe de gyrosensor en accelerometer meten.

Gegeven: Figuur 38: Een accelerometer bevat vaste plaatjes en een beweegbare massa die via veren vasthangt. Deze massa, tussen de vaste plaatjes, creëert een condensator. Deze veranderende capaciteit wordt gemeten en duidt op een beweging. De accelerometer meet dus lineaire snelheidsverschillen.



Figuur 38: Principewerking accelerometer.

Figuur 39 toont dat een gyrosensor een constant bewegende (oscillerende) kern heeft. Wanneer er zich een draaibeweging voordoet, verschuift de binnenste massa van plaats. Dan wordt de capaciteit tussen twee platen gemeten en zo kan de sensor de richting van de beweging achterhalen.



Figuur 39: Principewerking gyroscoop sensor.

MPU-6050

Slave address

⇨ 0x68 = 0110 1000 maar dit moet allemaal een plaatsje naar links geschoven worden omdat dit een **7 bit adres** is, **PLUS** de bit om te zeggen of we gaan **lezen of zenden**, als alles dan wordt opgeschoven, dan wordt dit: 1101 0000 = **0xD0**

4.34 Register 117 – Who Am I
WHO_AM_I

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
75	117	-	WHO_AM_I[6:1]						-

Description:

This register is used to verify the identity of the device. The contents of *WHO_AM_I* are the upper 6 bits of the MPU-60X0's 7-bit I^2C address. The least significant bit of the MPU-60X0's I^2C address is determined by the value of the AD0 pin. The value of the AD0 pin is not reflected in this register.

The default value of the register is 0x68.

Figuur 40: Datasheet van het MPU-6050 Slave address.

Index registers

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3A	58	INT_STATUS	R	-	MOT_INT	-	FIFO _OFLOW _INT	I2C_MST _INT	-	-	DATA _RDY_INT
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT[7:0]							

Figuur 41: Datasheet van alle registers die wij gebruiken in dit labo om de sensorwaarden in

Je ziet op Figuur 41 dat elke waarde bestaat uit twee bytes.

Software

Hoofdprogramma

```
#include <htc.h>
#include "I2C_LIB.h"
#include "HITECH_LCD.h"
#define _XTAL_FREQ  19660800
__CONFIG(0x2FF2);  // set HS Xtal etc
__CONFIG(0x3FFF);

unsigned char x , y;
unsigned int temp;
int GyX, GyY, GyZ, AcX, AcY, AcZ;
unsigned char SGyX, SGyY, SGyZ, SAcX, SAcY, SAcZ;

void main()
{

    ANSEL = 0;          // AD inputs normal IO
    ANSELH = 0;         // AD inputs normal IO
    OPTION_REG = 0xC0;
    TRISD = 0x00;

    __delay_ms(250);    //wait for stable power supply

    I2C_INIT();         // initialize I2C

    LCD_Start();

/**************SETTINGS UART**************/
    // 511 is getal voor baud rate - Xtal 19.660.800 en 9600 bps
    // FORMULE berekening baudrate: ((19660800 / 9600bps)/4)-1 = 511

    SPBRGH = 511 >> 8;      // vul baud rate register HIGH met 8 MSB's van spbrg
    SPBRG = 511 & 0xFF;     // vul baud rate register LOW met 8 LSB's van spbrg

    TRISC6 = 0;             // Pin C6 (pin 25) = TX als output zetten
    TRISC7 = 1;             // PIN C7 (Pin 26) = RX als input zetten

    BAUDCTL = 0x08;         // SCKP(0): Transmit non inverted data to TX pin /
BRG16(1): 16 bit Baud rate gen

    RCSTA = 0x90;           // SPEN(1): Serial Port Enabled / CREN(1): Enables
Receiver
    TXSTA = 0x24;           // TX9(0):8 bit / TXEN(1)Tansmit Enable /
SYNC(0):Asynchr / BRGH(1): High Speed

/**************SETTINGS UART**************/


    I2C_START_MESSAGE();            // start transaction
    I2C_TRANSMIT_BYTE(0xD0);        // slave address
    I2C_TRANSMIT_BYTE(0x6B);        // PWR_MGMT_1 reg
    I2C_TRANSMIT_BYTE(0);           // wake up MPU
    I2C_STOP_MESSAGE();             // stop transaction

    while(1){

        //// ACCELEROMETER ////
        I2C_START_MESSAGE();            // start transaction
        I2C_TRANSMIT_BYTE(0xD0);        // slave address
        I2C_TRANSMIT_BYTE(0x3B);        // index
        I2C_RESTART();
        I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
        x = I2C_RECEIVE_BYTE(1);
        I2C_STOP_MESSAGE();             // stop transaction
```

```
I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x3C);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
y = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction

AcX = y;
AcX |= x<<8;
SAcX = x;
PORTD = 0b00000001;

I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x3D);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
x = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction

I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x3E);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
y = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction

AcY = y;
AcY |= x<<8;
SAcY = x;
PORTD = 0b00000010;

I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x3F);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
x = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction

I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x40);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
y = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction

AcZ = y;
AcZ |= x<<8;
SAcZ = x;
PORTD = 0b00000100;


//// GYROSENSOR ////
I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x43);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
x = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction
```

```
I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x44);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
y = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction

GyX = y;
GyX |= x<<8;
SGyX = x;
PORTD = 0b00001000;

I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x45);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
x = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction

I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x46);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
y = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction

GyY = y;
GyY |= x<<8;
SGyY = x;
PORTD = 0b00010000;

I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x47);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
x = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction

I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x48);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
y = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction

GyZ = y;
GyZ |= x<<8;
SGyZ = x;
PORTD = 0b00100000;


//// TEMPERATURE ////
I2C_START_MESSAGE();            // start transaction
I2C_TRANSMIT_BYTE(0xD0);        // slave address
I2C_TRANSMIT_BYTE(0x41);        // index
I2C_RESTART();
I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
x = I2C_RECEIVE_BYTE(1);
I2C_STOP_MESSAGE();             // stop transaction
```

```
        I2C_START_MESSAGE();            // start transaction
        I2C_TRANSMIT_BYTE(0xD0);        // slave address
        I2C_TRANSMIT_BYTE(0x42);        // index
        I2C_RESTART();
        I2C_TRANSMIT_BYTE(0xD1);        // slave address + 1
        y = I2C_RECEIVE_BYTE(1);
        I2C_STOP_MESSAGE();             // stop transaction

        temp = y;
        temp |= x<<8;
        temp = temp/340+36,53;
        temp = temp/10;
        PORTD = 0b01000000;



        //// PRINT TO LCD ////
        LCD_Cursor(0,0);            // positie op LCD
        LCD_PrintString("GX:");     // print "GX:"
        LCD_PrintNumber(GyX);       // print GyX
        LCD_PrintString("  ");

        LCD_Cursor(9,0);
        LCD_PrintString("GY:");
        LCD_PrintNumber(GyY);       // print GyY

        LCD_Cursor(0,1);
        LCD_PrintString("GZ:");
        LCD_PrintNumber(GyZ);       // print GyZ
        LCD_PrintString("  ");

        LCD_Cursor(10,1);
        LCD_PrintString("T:");
        LCD_PrintNumber(temp);      // print temperatuur
        LCD_PrintString("C");

        PORTD = 0b10000000;


        //// SEND VIA RS232 ////

        while (!TXIF){}     // zolang TXIF (uit PIR1 register) bit laag is, mag er
geen nieuwe data naar TXREG verstuurd worden.
        TXREG = 0xAA;       // controlebyte om aan te geven dat we de data gaan
versturen
        while (!TXIF){}     // idem als hierboven
        TXREG = 0xAA;       // tweede controlebyte
        while (!TXIF){}     // idem als hierboven
        TXREG = SGyX;       // verzend 8 MSB's van het GyX register
        while (!TXIF){}     // idem als hierboven
        TXREG = SGyY;       // verzend 8 MSB's van het GyY register
        while (!TXIF){}     // idem als hierboven
        TXREG = SGyZ;       // verzend 8 MSB's van het GyZ register
        while (!TXIF){}     // idem als hierboven
        TXREG = SAcX;       // verzend 8 MSB's van het AcX register
        while (!TXIF){}     // idem als hierboven
        TXREG = SAcY;       // verzend 8 MSB's van het AcY register
        while (!TXIF){}     // idem als hierboven
        TXREG = SAcZ;       // verzend 8 MSB's van het AcZ register
        while (!TXIF){}     // idem als hierboven
        TXREG = temp;       // verzend de temperatuur
    }

}
```

I2C_lib

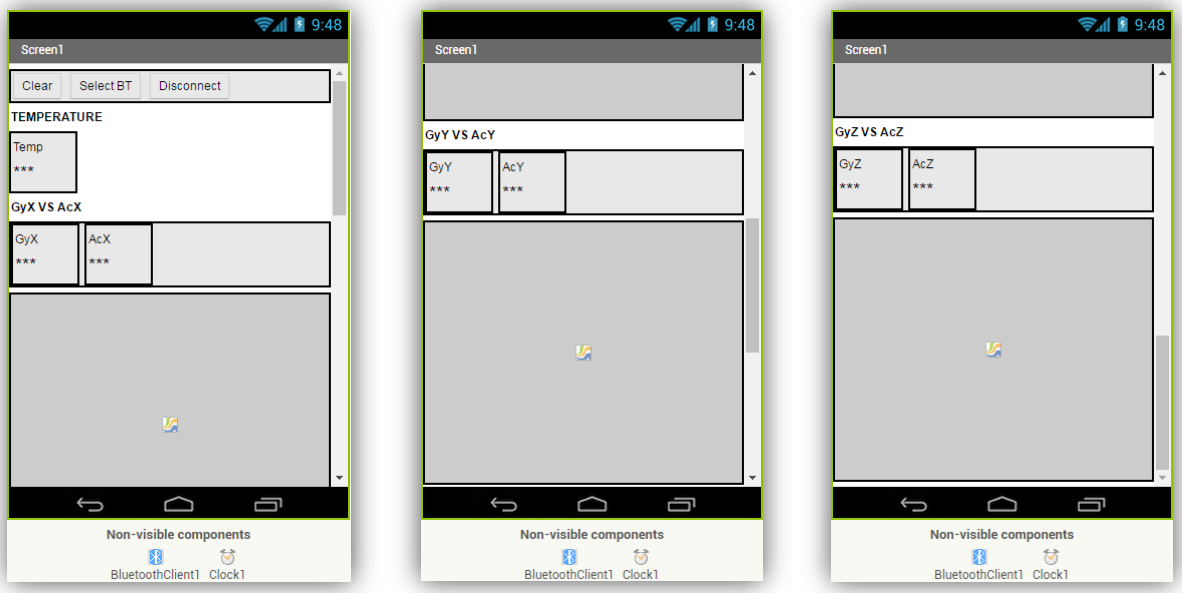
```
#include <htc.h>
#define _XTAL_FREQ  19660800     // oscillator frequency for _delay()

//Initialise PIC16F887 for Hardware I2C operation
void I2C_INIT()
{
    SSPSTAT |=0b10000000;       //_st_bit (SSPSTAT,SMP),Slew Rate Control Disabled
    SSPSTAT |=0b01000000;       //_cr_bit (SSPSTAT,CKE),Disable SMBus specific
inputs
    SSPCON = 0x28;              //Setup I2C into Master Mode
    SSPADD = 98;                //Set the Baud Rate
    SSPCON2 = 0x00;             //Clear the control bits
    GIE = 1;                    //_st_bit(INTCON, GIE);
    TRISC = TRISC|0b00001000;   //Configure SCL as Input
    TRISC = TRISC|0b00010000;   //Configure SDA as Input
}

// send a start bit sequence
void I2C_START_MESSAGE()
{
    PIR1 &=~0b00001000;         //cr_bit(PIR1,SSPIF),Clear SSP interrupt flag
    SSPCON2 |= 0x01;            //st_bit(SSPCON2,SEN),Initiate start condition
    while(SSPCON2 & 0x01);      // - as long as SEN is High - Wait for start bit to
be generated
}

// send a repeated start bit sequence
void I2C_RESTART()
{
    PIR1 &=~0b00001000;         //cr_bit(PIR1,SSPIF),Clear SSP interrupt flag
    SSPCON2 |= 0x02;            //st_bit(SSPCON2,RSEN),Initiate restart condition
    while(SSPCON2 & 0x02);      //  while(ts_bit(SSPCON2,RSEN)),Wait for restart
bit to be generated
}

// send a stop bit sequence
void I2C_STOP_MESSAGE()
{
    PIR1 &=~0b00001000;         //cr_bit(PIR1,SSPIF),Clear SSP interrupt flag
    SSPCON2 |= 0x04;            //st_bit(SSPCON2,PEN),Initiate stop condition
    while(SSPCON2 & 0x02);      //while(ts_bit(SSPCON2,PEN)),Wait for stop bit to
be generated
    __delay_ms(10);             //Wait before reusing the I2C BUS
}

// transmit a byte (this can be a devide adress - an index or just data - of 8 bit
// if return = 1 : No ack bit was received from the slave - if return = 0 = a
correct acknowledge bit was received
char I2C_TRANSMIT_BYTE(char Data)
{
    PIR1 &=~0b00001000;                 //cr_bit(PIR1,SSPIF),Clear SSP interrupt
flag
    SSPBUF=Data;                        //Send byte
        while((PIR1 & 0x08) == 0);      //Wait for control bit to be sent
            if(SSPCON2 & 0b01000000)    //Check Acknowledgement
                return (1);             //No Acknowledgement
                else return (0);        //Acknowledgement received
}



//receive a byte of data - requested by the master and sent by the slave
// returns a byte of incoming data
// if last = 1 - last byte was received - If last = 0 - the master expects further
I2C_RECEIVE_BYTE instructions
```

```
char I2C_RECEIVE_BYTE(char Last)
{
    PIR1 &=~0b00001000;                 //cr_bit(PIR1,SSPIF);
    SSPCON2 |=0x08;                     //st_bit(SSPCON2,RCEN),Initiate Read
    while((PIR1 & 0x08) == 0);          //while(ts_bit(PIR1,SSPIF) == 0),Wait for
data read
        if (Last)
            SSPCON2 |=0b00100000;       //st_bit(SSPCON2,ACKDT),Send Nack
        else SSPCON2 &=~0b00100000;     //cr_bit(SSPCON2,ACKDT),Send Ack
            SSPCON2 |=0b00010000;       //st_bit(SSPCON2,ACKEN),Initiate Nack
    while(SSPCON2&0b00010000);          //(ts_bit(SSPCON2,ACKEN)) Wait for data
read
    return(SSPBUF);                     //Store incoming data
}
```

Applicatie

App Inventor

Designer

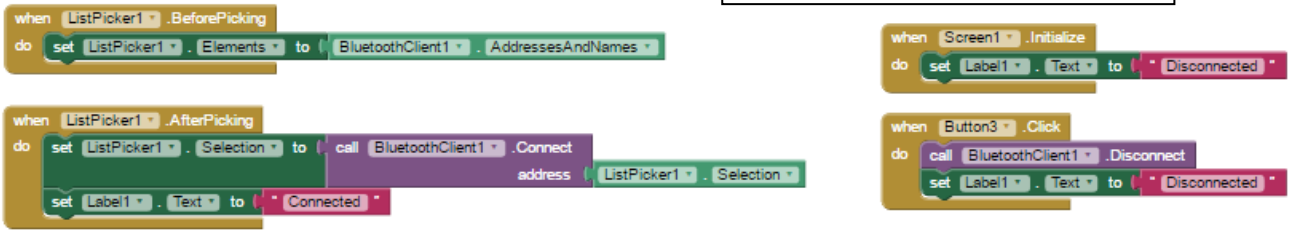


Figuur 42: Screenshots van applicatie in App Inventor.

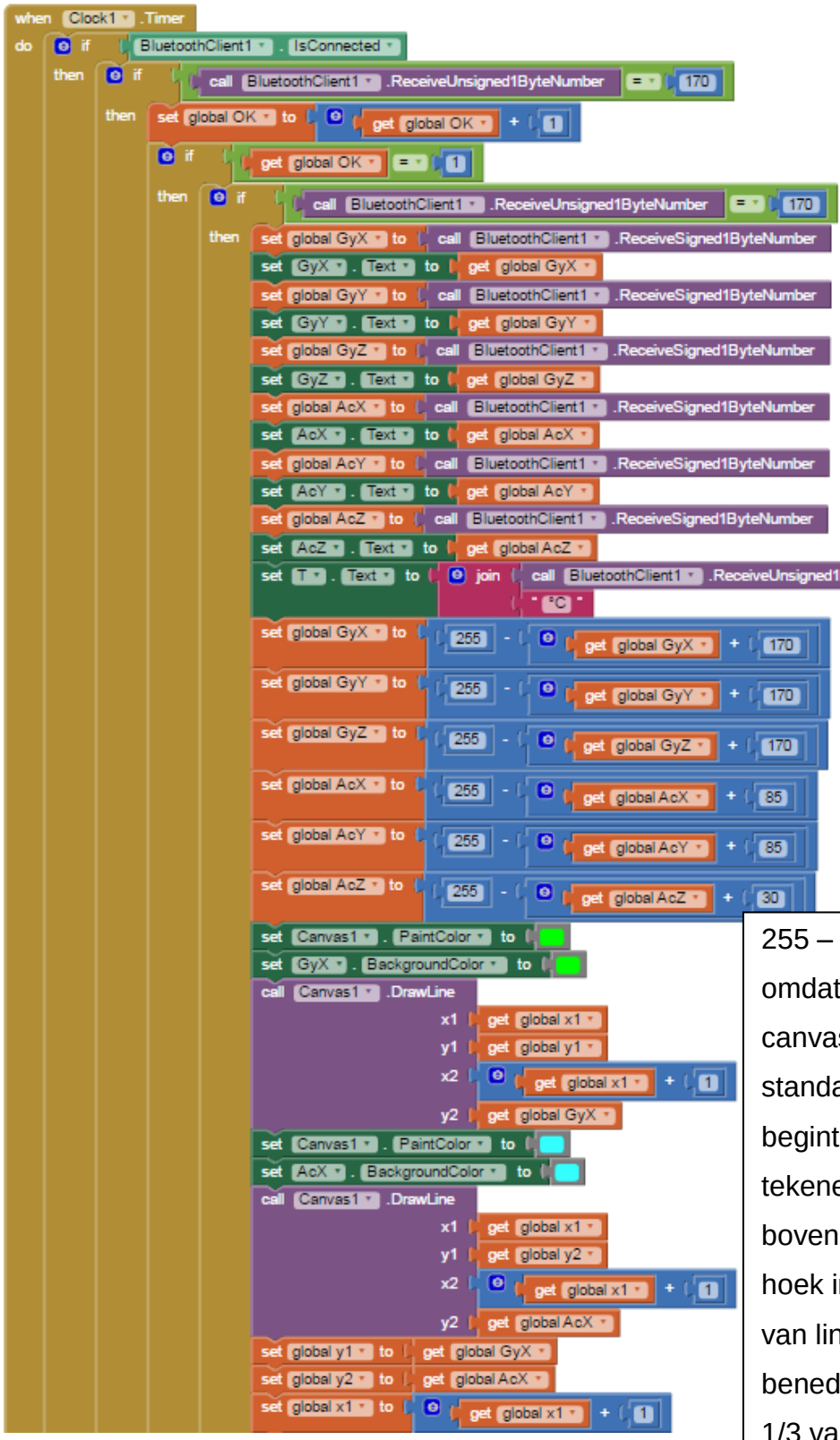
In Figuur 42 is duidelijk dat de bedoeling van deze app, is om de gyrosensor en de accelerometer van elke as met elkaar te vergelijken om zo te zien welke sensor in welke situatie het beste benut zou kunnen worden.

Blocks

Bluetooth communicatie tot stand brengen

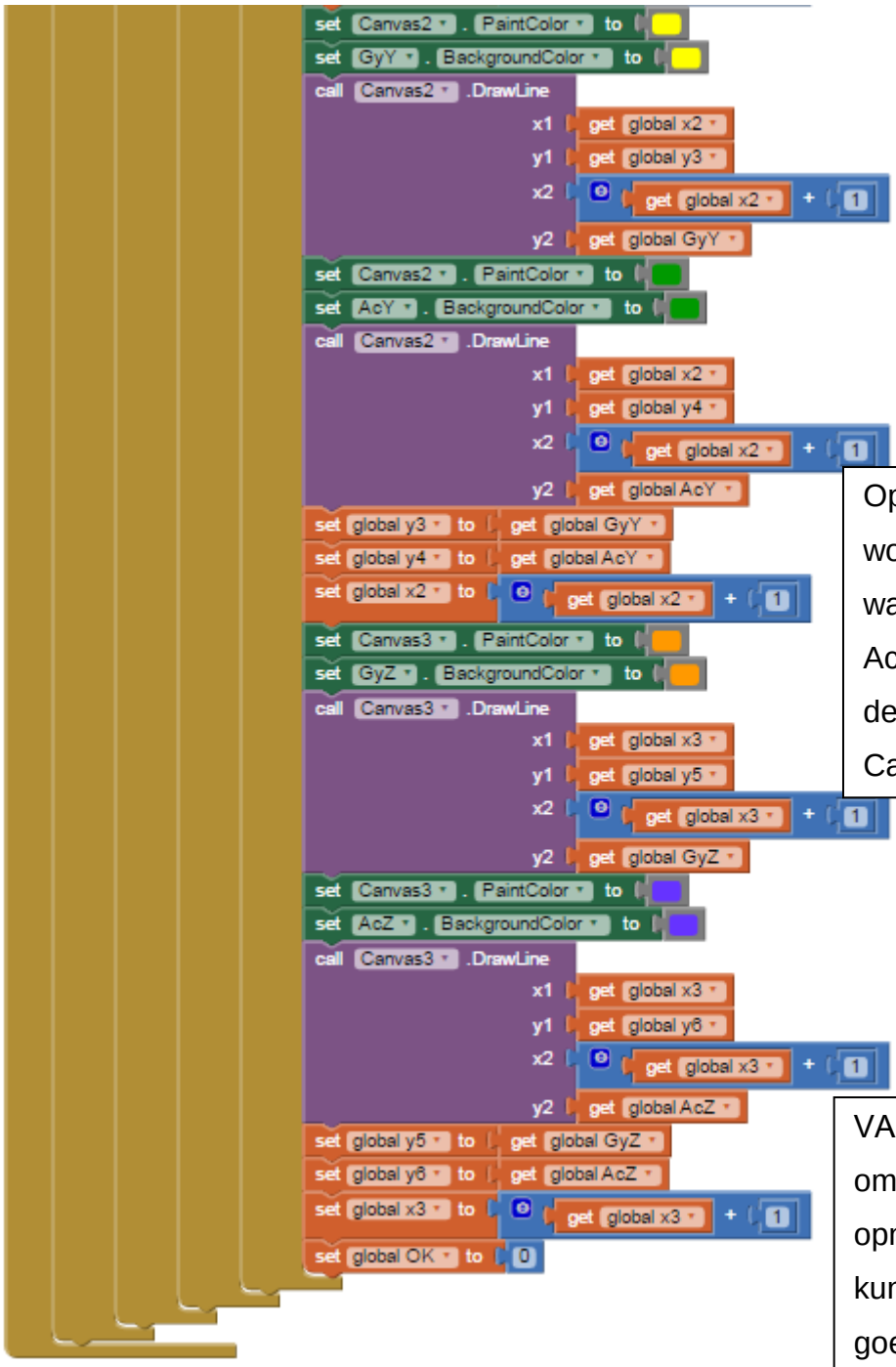


Wanneer de ontvangen data 170 (0xAA) is, dan wordt de variabele OK “1” gemaakt. Als OK dan “1” geworden is, dan wordt er gekeken of de ontvangen data een tweede keer 0xAA is. Als dat zo is, dan worden de volgende bytes binnengelezen.



Variabelen GyX en AcX worden voorgesteld op de eerste canvas. De lijnen voor deze voorstelling worden getekend door VAR “x1” telkens +1 te doen met daarbij ook een nieuwe Y-waarde, ingelezen via de ontvangen gyro en accelero waarde.

255 – VAR, omdat de canvas standaard begint te tekenen links bovenaan in de hoek in plaats van links beneden.
1/3 van 255 = 85
2/3 van 255 = 170
Om de waarden mooi boven elkaar weer te geven.



Figuur 43: Programma voor de applicatie in App Inventor.

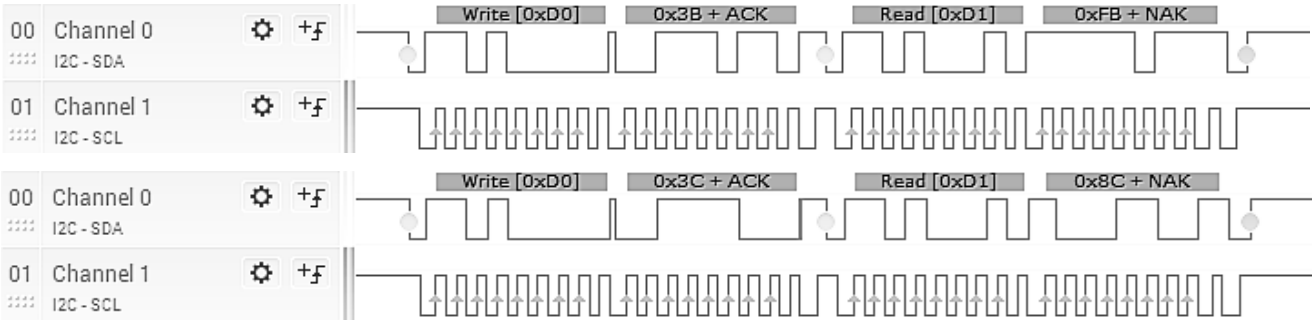
Op Canvas2 en Canvas3 worden respectievelijk de waarden GyY, AcY en GyZ, AcZ grafisch voorgesteld op dezelfde manier als bij Canvas1.

VAR “OK” wordt op “0” gezet om aan het begin van de loop opnieuw de controle te kunnen uitvoeren om een goede dataoverdracht tot stand te brengen.

Logic analyser

I²C

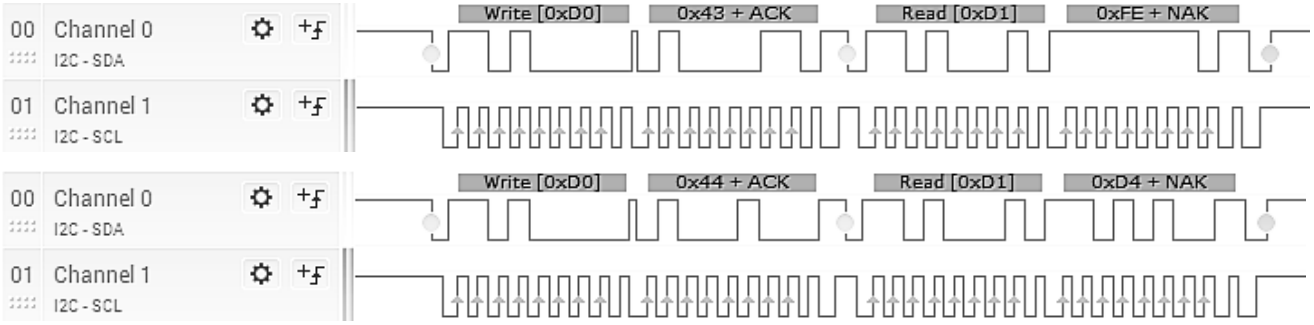
AcX



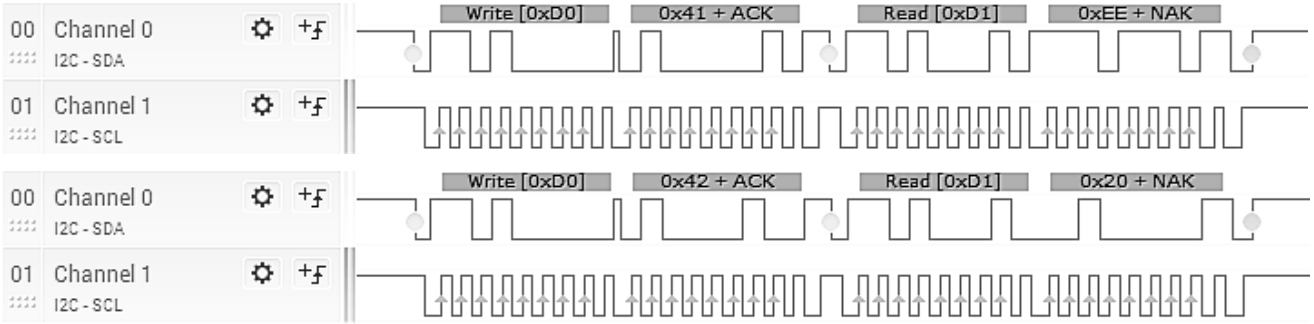
Figuur 44: Logic analyser beeld van de I²C communicatie met de MPU-6050.

In Figuur 44 worden de twee bytes van het AcX register ingelezen. Eerst “roep” ik de sensor door het slave address (0xD0) te sturen. Dan zend ik de index (0X3B en 0x3C) waaruit ik wil lezen. Door het slave address plus één te doen (0xD1), wordt ervoor gezorgd dat de data van de sensor, die in deze index staat, naar de microcontroller gestuurd wordt.

GyX

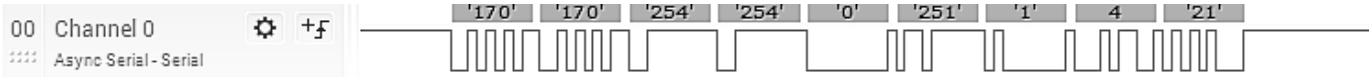


Hier worden de twee bytes van het AcY register ingelezen. Dit werkt volgens identiek hetzelfde principe als hierboven.



Op dit beeld wordt de temperatuur ingelezen, tevens op dezelfde manier. Zo geldt dit ook voor register AcY, AcZ, GyY en GyZ.

RS232



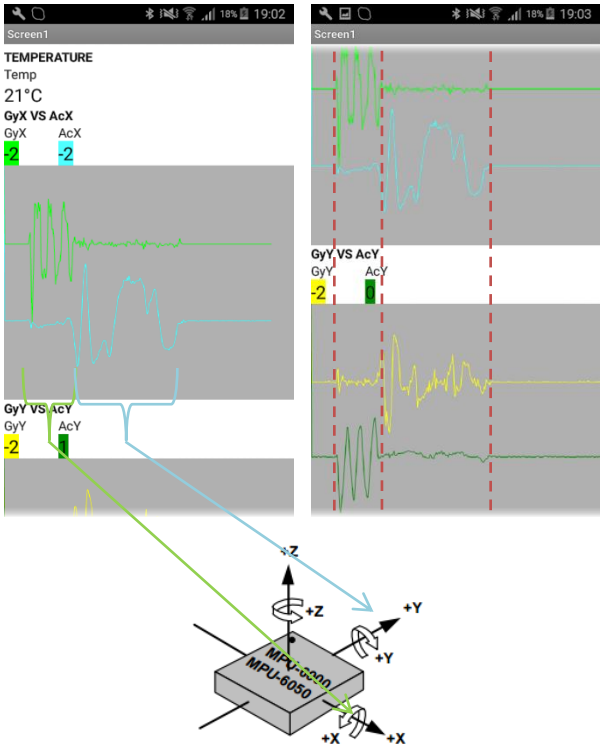
Figuur 45: Logic analyser beeld van de RS232 communicatie met de MPU-6050.

In Figuur 45 zien we data die via RS232 gecommuniceerd wordt. Eerst worden twee controlebytes (0xFF = 170) doorgestuurd om ervoor te zorgen dat de data juist overgezonden zal worden. Dit is ook duidelijk in de “code” van App Inventor. Verder wordt gewoon de data overgezonden zoals duidelijk te zien is in de “C” code. In respectievelijke volgorde: GyX, GyY, GyZ, AcX, AcY, AcZ en temperatuur.

Metingen app

Op de screenshots in Figuur 46 is te zien hoe de sensoren op verschillende bewegingen reageren.

Wat we ook zien is dat de gemeten waarden eigenlijk niet juist kunnen zijn. Eerst dacht ik dat er verkeerde data werd overgezonden, of ontvangen door de applicatie. Maar door de gemeten data weer te geven op het LCD, wist ik dat het probleem zich niet voordeed bij het verzenden via Bluetooth en de applicatie want de data was nog steeds fout. Ik vermoed dus dat de fout zich zou voordoen bij het inlezen van de registers van de sensor. Maar aangezien ik hierbij ook geen fouten gemaakt heb, begrijp ik niet hoe dit kan.

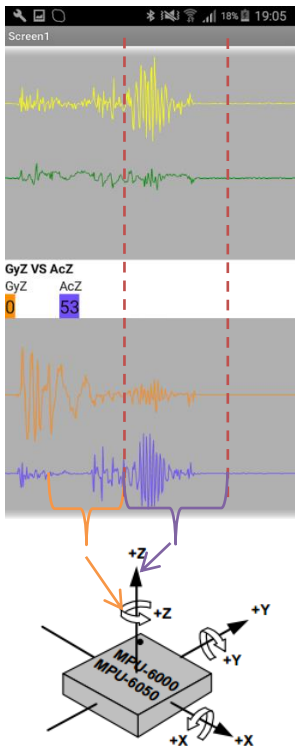


Figuur 46: Screenshots, afbeelding MPU-6050 Je ziet dat GyX en AcY met elkaar bewegen en dat AcY eigenlijk een gyrosensor is (waarde blijft constant t.o.v. de zwaartekracht) en GyX een accelerometer (versnelling verandert => waarde verandert). Hetzelfde geldt voor AcX en GyY. Daarbij constateer ik dat AcX een gyrosensor is en GyY een accelerometer.

In het eerste deel van het beeld in Figuur 47 is te zien dat GyZ effectief reageert op een draaiende beweging rond de Z-as.

Wanneer de sensor op en neer bewogen wordt (de beweging van de accelerometer van de Z-as, zien we dat AcZ wel effectief reageert op deze beweging, maar ook GyY.

Van GyX en AcX heb ik geen beeld genomen, omdat deze niet speciaal reageerden op deze beweging.



Figuur 47: Screenshot, afbeelding MPU-6050.

Wat ik wel kan afleiden uit deze beelden.

Als je een tijdje bezig bent met deze metingen te bekijken, wordt snel duidelijk dat de accelerometer beter reageert bij bruuske bewegingen zoals schudden e.d. De gyroscoop sensor daarentegen, meet trager en kan deze bewegingen niet volgen.

Oplossing?

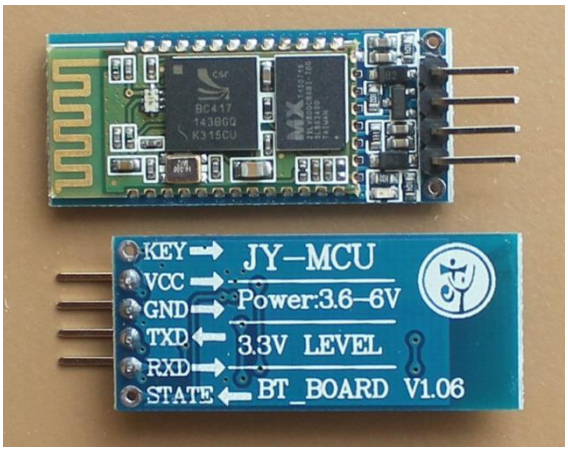
Mijn interpretatie van de foute data, is dat misschien de verkeerde registers ingelezen worden, hieronder geef ik mijn voorstellen.

GyX is een accelerometer, maar reageert wel bij een X-beweging.	GyX ⇨ AcX
AcX is een gyrosensor en beweegt bij een Y-beweging.	AcX ⇨ GyY
GyY is een accelerometer, maar reageert wel bij een Y-beweging.	GyY ⇨ AcY
AcY is een gyrosensor en beweegt bij een X-beweging.	AcY ⇨ GyX
GyZ is vermoedelijk een gyrosensor, maar de waarde gaat wel terug naar nul, en reageert bij een Z-beweging.	GyZ ⇨ GyZ
AcX is een accelerometer en beweegt bij een Z-beweging.	AcZ ⇨ AcZ

Wanneer ik deze MPU-6050 gebruik met Arduino, geeft deze sensor wel de juiste metingen door. Hoewel de code van Arduino en de C code van dit project exact hetzelfde is, is het tot op heden nog steeds een raadsel voor mij waarom het met deze code niet werkt.

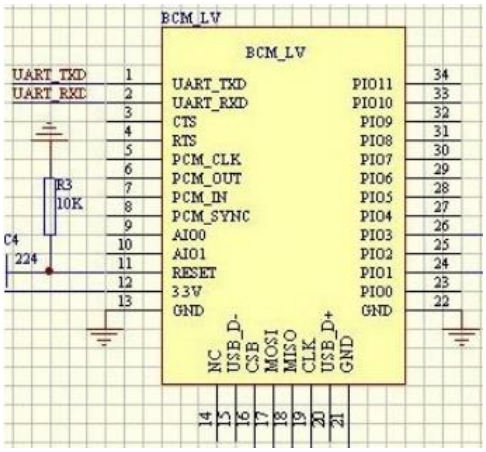
Extra datasheets

HC-06



Figuur 48: HC-06 bluetooth module.

In Figuur 48 zien we de HC-06 bluetooth module. De +5V van de µC moet aan de VCC gehangen worden en de GND's aan elkaar verbonden. Verder dient de TXD pin van deze module aan de RX pin van de µC gehangen worden, en de RXD pin aan de TX pin van de microcontroller.



Figuur 49: Elektronisch schema van de microcontroller op de HC-06 module.

Microcontroller registers

I²C

SSPSTAT

REGISTER 13-1: SSPSTAT: SSP STATUS REGISTER

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7 1	1						bit 0

bit 7 **SMP**: Sample bit
SPI Master mode:
1 = Input data sampled at end of data output time
0 = Input data sampled at middle of data output time
SPI Slave mode:
SMP must be cleared when SPI is used in Slave mode
In I²C Master or Slave mode:
1 = Slew rate control disabled for standard speed mode (100 kHz and 1 MHz)
0 = Slew rate control enabled for high speed mode (400 kHz)

Slew rate control wordt gedisabled door bit 7 op “1” te zetten.

bit 6 **CKE**: SPI Clock Edge Select bit
CKP = 0:
1 = Data transmitted on falling edge of SCK
0 = Data transmitted on rising edge of SCK
CKP = 1:
1 = Data transmitted on rising edge of SCK
0 = Data transmitted on falling edge of SCK

bit 0 **BF**: Buffer Full Status bit
Receive (SPI and I²C modes):
1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty
Transmit (I²C mode only):
1 = Data transmit in progress (does not include the $\overline{ACK}$ and Stop bits), SSPBUF is full
0 = Data transmit complete (does not include the $\overline{ACK}$ and Stop bits), SSPBUF is empty

De buffer (bit 0) wordt gebruikt bij het verzenden van data.

Figuur 50: Datasheet SSPSTAT register.

SSPCON

REGISTER 13-2: SSPCON: SSP CONTROL REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7		1		1			bit 0

bit 5 **SSPEN**: Synchronous Serial Port Enable bit
In both modes, when enabled, these pins must be properly configured as input or output
In SPI mode:
1 = Enables serial port and configures SCK, SDO, SDI and $\overline{SS}$ as the source of the serial port pins
0 = Disables serial port and configures these pins as I/O port pins
In I²C mode:
1 = Enables the serial port and configures the SDA and SCL pins as the source of the serial port pins
0 = Disables serial port and configures these pins as I/O port pins

bit 3-0 **SSPM<3:0>**: Synchronous Serial Port Mode Select bits
0000 = SPI Master mode, clock = Fosc/4
0001 = SPI Master mode, clock = Fosc/16
0010 = SPI Master mode, clock = Fosc/64
0011 = SPI Master mode, clock = TMR2 output/2
0100 = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control enabled
0101 = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control disabled, $\overline{SS}$ can be used as I/O pin
0110 = I²C Slave mode, 7-bit address
0111 = I²C Slave mode, 10-bit address
1000 = I²C Master mode, clock = Fosc / (4 * (SSPADD+1))
1001 = Load Mask function
1010 = Reserved
1011 = I²C firmware controlled Master mode (Slave idle)
1100 = Reserved
1101 = Reserved
1110 = I²C Slave mode, 7-bit address with Start and Stop bit interrupts enabled
1111 = I²C Slave mode, 10-bit address with Start and Stop bit interrupts enabled

De SSPEN bit zorgt ervoor dat de SDA en SCL pinnen ingesteld zijn om te gebruiken.
SSPM (bit 3:0) dient om de I²C in master mode of slave mode te zetten. Wij opteren hier voor de master mode; de klok hiervan is Fosc/4.

Figuur 51: Datasheet SSPCON register.

SSPCON2

REGISTER 13-3: SSPCON2: SSP CONTROL REGISTER 2

R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							1 bit 0

bit 6 **ACKSTAT:** Acknowledge Status bit (in I^2C Master mode only)
In Master Transmit mode:
1 = Acknowledge was not received from slave
0 = Acknowledge was received from slave

bit 5 **ACKDT:** Acknowledge Data bit (in I^2C Master mode only)
In Master Receive mode:
Value transmitted when the user initiates an Acknowledge sequence at the end of a receive
1 = Not Acknowledge
0 = Acknowledge

bit 4 **ACKEN:** Acknowledge Sequence Enable bit (in I^2C Master mode only)
In Master Receive mode:
1 = Initiate Acknowledge sequence on SDA and SCL pins, and transmit ACKDT data bit. Automatically cleared by hardware.
0 = Acknowledge sequence idle

bit 3 **RCEN:** Receive Enable bit (in I^2C Master mode only)
1 = Enables Receive mode for I^2C
0 = Receive idle

bit 2 **PEN:** Stop Condition Enable bit (in I^2C Master mode only)
SCK Release Control:
1 = Initiate Stop condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Stop condition Idle

bit 1 **RSEN:** Repeated Start Condition Enabled bit (in I^2C Master mode only)
1 = Initiate Repeated Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Repeated Start condition Idle

bit 0 **SEN:** Start Condition Enabled bit (in I^2C Master mode only)
In Master mode:
1 = Initiate Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Start condition Idle
In Slave mode:
1 = Clock stretching is enabled for both slave transmit and slave receive (stretch enabled)
0 = Clock stretching is disabled

Figuur 52: Datasheet SSPCON2 register.

Eerst wordt bit 0 gebruikt om de start conditie aan te geven (start message).
Bit 1 wordt gebruikt om de restart conditie aan te geven.
Bit 2 beheert de stop conditie (stop message).
Bit 3 dient om de I²C module in leesmode te zetten (receive byte).
Bit 4: Initialiseer acknowledge bit.
Bit 5: Zend Ack/Nack bij ontvangst
Bit 6: Kijk of Ack/Nack ontvangen is.

PIR1

REGISTER 2-6: PIR1: PERIPHERAL INTERRUPT REQUEST REGISTER 1

U-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 3 **SSPIF**: Master Synchronous Serial Port (MSSP) Interrupt Flag bit
1 = The MSSP interrupt condition has occurred, and must be cleared in software before returning from the Interrupt Service Routine. The conditions that will set this bit are:
SPI
A transmission/reception has taken place
I^2C Slave/Master
A transmission/reception has taken place
I^2C Master
The initiated Start condition was completed by the MSSP module
The initiated Stop condition was completed by the MSSP module
The initiated restart condition was completed by the MSSP module
The initiated Acknowledge condition was completed by the MSSP module
A Start condition occurred while the MSSP module was Idle (Multi-master system)
A Stop condition occurred while the MSSP module was Idle (Multi-master system)
0 = No MSSP interrupt condition has occurred

Figuur 53: Datasheet PIR1 register.

Met bit 3 wordt de SSP interrupt flag 1 en 0 gemaakt op de juiste momenten in het programma.

RS232

BAUD RATE berekenen

TABLE 12-3: BAUD RATE FORMULAS

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	Fosc/[64 (n+1)]
0	0	1	8-bit/Asynchronous	Fosc/[16 (n+1)]
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	Fosc/[4 (n+1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

Tabel 8: Datasheet BAUD RATE formuletabel.

$$\rightarrow \quad \frac{\frac{19660800}{9600bps}}{4} - 1 = 511$$

BAUD RATE register

REGISTER 12-3: BAUDCTL: BAUD RATE CONTROL REGISTER

R-0	R-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN
bit 7							bit 0

bit 7 **ABDOVF:** Auto-Baud Detect Overflow bit
Asynchronous mode:
1 = Auto-baud timer overflowed
0 = Auto-baud timer did not overflow
Synchronous mode:
Don't care

bit 6 **RCIDL:** Receive Idle Flag bit
Asynchronous mode:
1 = Receiver is Idle
0 = Start bit has been received and the receiver is receiving
Synchronous mode:
Don't care

bit 5 **Unimplemented:** Read as '0'

bit 4 **SCKP:** Synchronous Clock Polarity Select bit
Asynchronous mode:
1 = Transmit inverted data to the RB7/TX/CK pin
0 = Transmit non-inverted data to the RB7/TX/CK pin
Synchronous mode:
1 = Data is clocked on rising edge of the clock
0 = Data is clocked on falling edge of the clock

bit 3 **BRG16:** 16-bit Baud Rate Generator bit
1 = 16-bit Baud Rate Generator is used
0 = 8-bit Baud Rate Generator is used

bit 2 **Unimplemented:** Read as '0'

bit 1 **WUE:** Wake-up Enable bit
Asynchronous mode:
1 = Receiver is waiting for a falling edge. No character will be received byte RCIF will be set. WUE will automatically clear after RCIF is set.
0 = Receiver is operating normally
Synchronous mode:
Don't care

bit 0 **ABDEN:** Auto-Baud Detect Enable bit
Asynchronous mode:
1 = Auto-Baud Detect mode is enabled (clears when auto-baud is complete)
0 = Auto-Baud Detect mode is disabled
Synchronous mode:
Don't care

Figuur 54: Datasheet BAUD RATE register.

In Figuur 55 is te zien dat, om te kunnen zenden en ontvangen, het BAUD RATE register dient ingesteld te worden. Hierbij zeggen we dat de data niet geïnverteerd moet worden en er een 16-bit Baud Rate Generator gebruikt wordt.

RSCTA

REGISTER 12-2: RCSTA: RECEIVE STATUS AND CONTROL REGISTER(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RX/DT and TX/CK pins as serial port pins)
0 = Serial port disabled (held in Reset)

bit 6 **RX9:** 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN:** Single Receive Enable bit
Asynchronous mode:
Don't care
Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode – Slave
Don't care

bit 4 **CREN:** Continuous Receive Enable bit
Asynchronous mode:
1 = Enables receiver
0 = Disables receiver
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enable interrupt and load the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit
Asynchronous mode 8-bit (RX9 = 0):
Don't care

bit 2 **FERR:** Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR:** Overrun Error bit
1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** Ninth bit of Received Data
This can be address/data bit or a parity bit and must be calculated by user firmware.

Figuur 55: Datasheet RCSTA register.

Om te kunnen ontvangen moet het RCSTA register aangepast worden, hier moeten we de serial port "enablen" door deze bit op 1 te zetten. Ook moeten we ervoor zorgen dat de Receiver aan staat.

REGISTER DEFINITIONS: EUSART CONTROL

REGISTER 12-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC**: Clock Source Select bit
Asynchronous mode:
Don't care
Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit(1)
1 = Transmit enabled
0 = Transmit disabled

bit 4 **SYNC**: EUSART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **SENDB**: Send Break Character bit
Asynchronous mode:
1 = Send Sync Break on next transmission (cleared by hardware upon completion)
0 = Sync Break transmission completed
Synchronous mode:
Don't care

bit 2 **BRGH**: High Baud Rate Select bit
Asynchronous mode:
1 = High speed
0 = Low speed
Synchronous mode:
Unused in this mode

bit 1 **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D**: Ninth bit of Transmit Data
Can be address/data bit or a parity bit.

Figuur 56: Datasheet EUSART control register.

Om te zenden moeten we de in de instellingen van TXSTA register een aantal aanpassingen doen. Deze zijn: kiezen voor een datatransmissie van 8 bits, uiteraard moet “Transmit” aan staan (anders kunnen we niet zenden). We kiezen ook voor een High speed dataoverdracht.

Besluiten

Bij de realisatie van dit project heb ik veel tijd gestoken in het uitzoeken hoe de MPU-6050 module in elkaar zit, met name de registers en dergelijke. De informatie hierover is vrij moeilijk te vinden, waardoor ik daardoor vrij veel tijd verloren heb. Hierdoor diende de deadline voor een andere deelopdracht verschoven te worden.
Ik heb in dit labo veel bijgeleerd over het tot stand brengen van de goede communicatie via Bluetooth. Ik heb een functie geschreven om controlebytes in te lezen en zo ervoor te zorgen dat alle data op het juiste moment werd verzonden en ontvangen.
Wat de foute data bevat, heb ik een voorstel gegeven over hoe dit zou kunnen opgelost worden. Maar het lijkt me sowieso niet normaal dat er hierdoor andere registers zouden moeten ingelezen worden dan in de datasheet vermeld staan. Misschien moet er een bepaalde reset of kalibratie van de module toegepast worden alvorens men zeker kan zijn dat de juiste data wordt uitgelezen. Ik heb over dit onderwerp op het internet gezocht maar hierover is niets te vinden. Ik hoop dus dat we dit eens samen eens goed kunnen bekijken en dit spoedig oplossen, zodat ik dit probleem naar de volgende deadline toe, kan wegwerken en dit uiteraard zeer goed kan gaan uitleggen in de “verbetering” van dit verslag.

1.5. Labo Energieverbruik

Doelstelling

Het doel van dit labo is om te gaan kijken hoeveel de totale GIP verbruikt, aangezien dit tegenwoordig een belangrijk aspect van een project wordt. Er dient zoveel mogelijk rekening gehouden te worden met “verbruiken” door de opwarming van de aarde. Daarom moeten veel toestellen gekeurd en nagekeken worden op zuinigheid en dergelijke.

Oplijsting componenten, Berekeningen & metingen

Brainbox Arduino

Na verschillende bronnen te onderzoeken, en er vanuit gaande dat de Brainbox Arduino ongeveer even veel verbruikt als een gewone Arduino, ben ik erop uit gekomen dat deze shield standaard, zonder extra belastingen, zo’n 30mA verbruikt bij een bron van 5V.

$P = U * I = 5 * 0,3 = 150mW$

Ik heb dit gemeten, maar ik mat 77mA.

In de praktijk resulteert dit dus in een vermogen van:

$P = U * I = 5 * 0,77 = 485mW$

HC-06 Bluetooth Module

Tijdens het “pairen” verbruikt deze Bluetooth module 30 – 40mA. Voor de rest, tijdens dataoverdracht is dit slechts 8mA. Deze module wordt gevoed met 5V.

$P = U * I = 5 * 0,03 = 150mW$

$P = U * I = 5 * 0,008 = 40mW$

Mijn gemeten waarden komen exact overeen met de datasheet. Ik mat een licht onstabiele spanning tussen 38mA en 42mA tijdens het pairen en ongeveer 8mA in normaal bedrijf.

GY-521 breakout board met MPU-6050 gyro-accelerometer

In de datasheet van de GY-521 is te zien dat deze schakeling ongeveer 20mA verbruikt met een bronspanning van 5V.

$P = U * I = 5 * 0,02 = 100mW$

Ik mat 40mA.

In de praktijk:

$P = U * I = 5 * 0,04 = 200mW$

SG90 Servomotor

In de datasheet van de gebruikte servomotor staat dat de maximum stroom 700mA is. Wetend dat deze motor gevoed wordt door 5V:

$P = U * I = 5 * 0{,}7 = 3{,}5W$

Ik mat 400mA.

$P = U * I = 5 * 0{,}4 = 2W$

Pololu DRV8825 microstepper driver met stappenmotor

Eerder hebben we dit al besproken in een vorig labo. Dit labo heeft uitgewezen dat de combinatie van deze twee componenten, een stroom van 232mA trekt in normale toestand. Bij twee motoren is dat dan het dubbel: 464mA (afgerond 500mA). De bronspanning is hier wel 10,8V

$P = U * I = 10{,}8 * 0{,}5 = 5{,}4W$

Bij spieken kan de stroom door 1 motor zelfs oplopen tot 800mA. Dat maakt 1,6A door de twee.

$P = U * I = 10{,}8 * 1{,}6 = 17{,}28W$

Voor deze metingen kan je teruggaan naar het labo "Labo stappenmotor en Pololu DRV8825 driver: Deel2: Piekstromen en spanningsdrops".

Totaal		**Theoretisch**		**Praktisch**	
		Stroom	Vermogen	Stroom	Vermogen
Batterij 9V, omgezet naar 5V.	Brainbox Arduino	0,03A	0,15W	0,077A	0,485W
	HC-06	0,04A	0,15W	0,04A	0,15W
	GY-521	0,02A	0,1W	0,04A	0,2W
	SG90	0,7A	3,5W	0,4A	2W
Subtotaal		0,8A	3,9W	**0,6A**	2,8W
Batterij 11,1V	Pololu DRV8825 & stappenmotor	0,5A en 1,6A	Tussen 5,4W en 17,28W	0,5A en **1,6A**	Tussen 5,4W en 17,28W
TOTAAL		Afgerond tussen 1,3A en 2,4A.	Afgerond tussen 9W en 22W.	Afgerond tussen 1,1A en 2,2A.	Afgerond tussen 8W en 21W.

Besluiten

De robot wordt gevoed met twee batterijen: één 9V batterij waarvan de spanning via een 5V LDO naar 5V wordt omgezet. De andere batterijen zijn drie krachtige, herlaadbare 18650 batterijen (3,7V x 3 = 11,1V) met een capaciteit van 5000 mAh die de stroompieken, afkomstig van de motoren, kunnen verdragen.

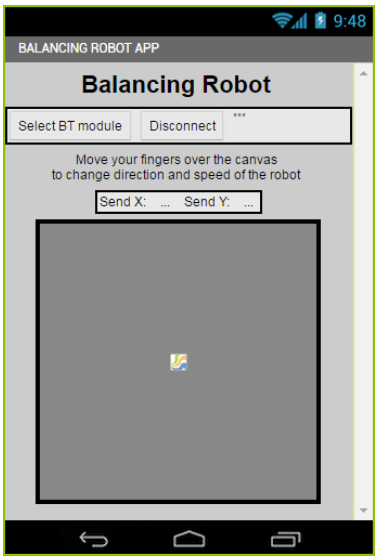
Stel dat onze 9V batterij een capaciteit heeft van 550mAh, dat betekent dat we (600/550) 0,9 uur ofwel 55 minuten aan één stuk deze robot kunnen laten functioneren alvorens de batterij te moeten vervangen.

Bij de 18650 batterijen is dat maar liefst (5000/1600) 3,125 uur ofwel 187 minuten! Het spreekt voor zich dat de motoren het meeste verbruiken, maar ondanks hun grote stroomverbruik, beschikken ze wel over zeer krachtige batterijen die dit niet laten merken.

1.6. Programma

1.6.1. Applicatie

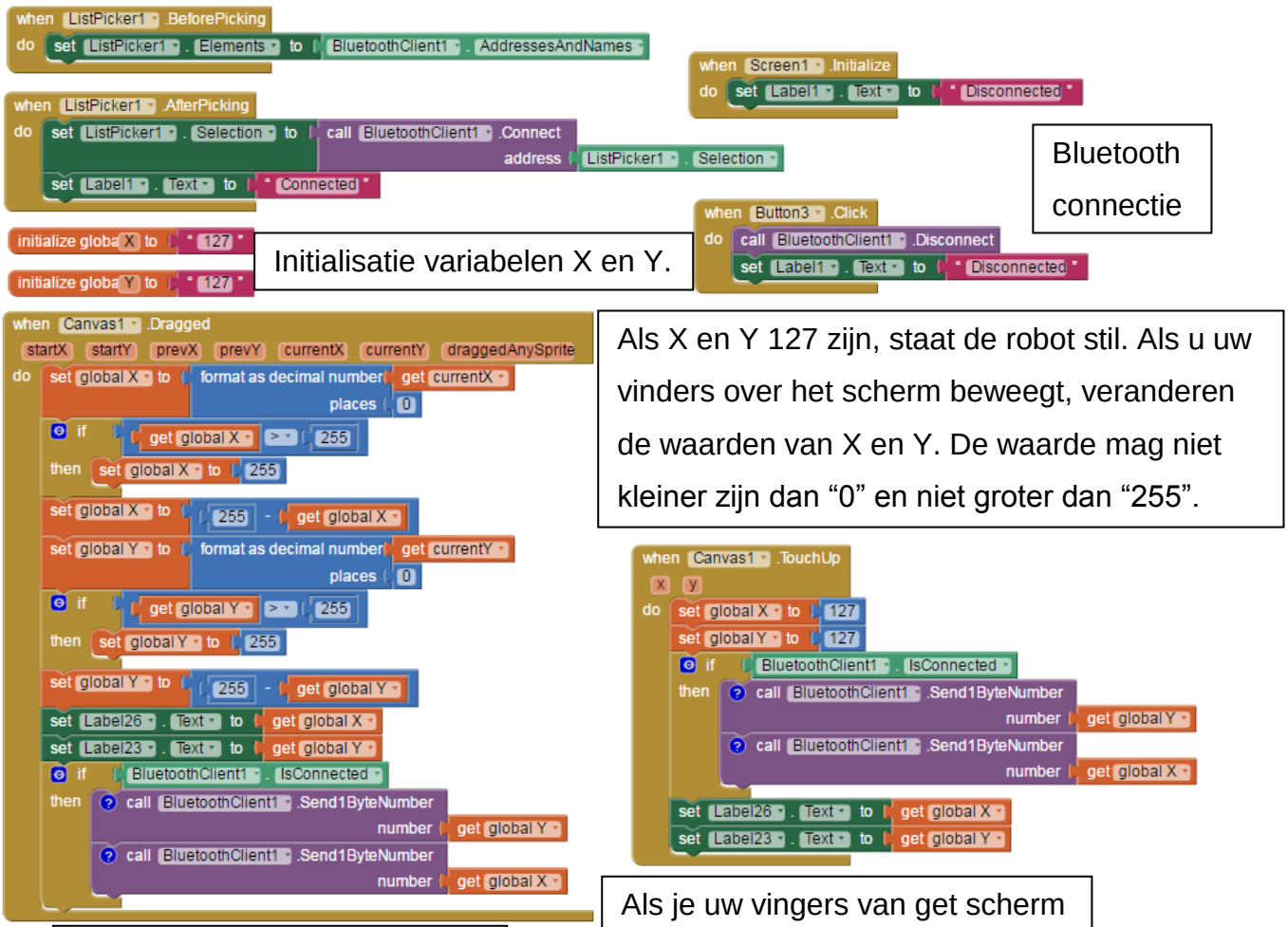
Designer



In Figuur 57 zie je dat er bovenaan het scherm staat de titel van de applicatie. Daaronder kan men de Bluetooth module selecteren (in een lijst) en hiermee connecteren. De tekst daaronder legt kort uit hoe je de robot moet besturen. Daaronder staat de x- en y-waarde die via Bluetooth naar de robot doorgestuurd zullen worden. De canvas onderaan het beeld is het veld waarover je met uw vingers kan bewegen en hiermee de robot kan besturen.

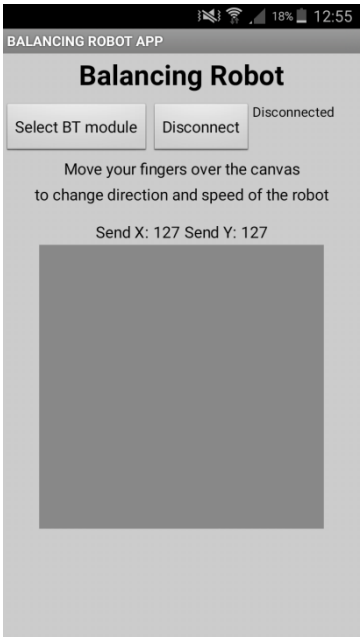
Figuur 57: Lay out app

Blocks



Figuur 58: Blokprogramma van de applicatie

Screenshot



Je ziet in Figuur 59 dat er op de plaats van de 3 sterretjes nu ofwel “Connected” ofwel “Disconnected” komt te staan.

Op het moment van de screenshot werd het scherm niet aangeraakt, waardoor de standaardwaarde voor X en Y (127) werd doorgestuurd.

Figuur 59: Screenschot van de applicatie

1.6.2. Software

```
// SELF BALANCING ARDUINO ROBOT WITH STEPPER MOTORS
// (Brainbox Arduino + Homemade shield v2 + STEPPER MOTOR drivers + MPU-6050)
// You need to install libraries I2Cdev and MPU6050 (from Github repository)
// Author: JJROBOTS.COM (Jose Julio & Juan Pedro)
// Last updated: 05/05/2017
// Version: 3.0
// Remember to update the libraries

// The board needs at least 10-15 seconds with no motion (robot steady) at beginning to give good values...
// MPU6050 IMU using internal DMP processor. Connected via I2C bus
// Angle calculations and control part is running at 200Hz from DMP solution
// DMP is using the gyro_bias_no_motion correction method.

// The robot is OFF when the angle is high (robot is horizontal). When you start raising the robot it
// automatically switch ON and start a RAISE UP procedure.
// To switch OFF the robot you could manually put the robot down on the floor (horizontal)

// We use a standard PID controllers (Proportional, Integral derivative controller) for robot stability
// We have a PI controller for speed control and a PD controller for stability (robot angle)
// The output of the control (motors speed) is integrated so it´s really an acceleration not an speed.
```

```
// controls:
//    Throttle (0.0-1.0)
//    Steering (0.0-1.0)
//    toggle1: Enable PRO mode. On PRO mode steering and throttle are more
aggressive

#include <Wire.h>
#include <I2Cdev.h>                // I2Cdev lib from www.i2cdevlib.com
#include <avr/interrupt.h>
#include <JJ_MPU6050_DMP_6Axis.h>  // Modified version of the MPU6050
library to work with DMP (see comments inside)
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE ((( F_CPU / 16) + ( USART_BAUDRATE / 2) ) / (
USART_BAUDRATE )) - 1)
// This version optimize the FIFO (only contains quaternion) and minimize
code size

// NORMAL MODE PARAMETERS (MAXIMUM SETTINGS)
#define MAX_THROTTLE 580
#define MAX_STEERING 150
#define MAX_TARGET_ANGLE 12

// PRO MODE = MORE AGGRESSIVE (MAXIMUM SETTINGS)
#define MAX_THROTTLE_PRO 980 //680
#define MAX_STEERING_PRO 250
#define MAX_TARGET_ANGLE_PRO 40 //20

// Default control terms
#define KP 0.19
#define KD 45
#define KP_THROTTLE 0.07
#define KI_THROTTLE 0.04


// Control gains for raiseup (the raiseup movement requiere special control
parameters)
#define KP_RAISEUP 0.16
#define KD_RAISEUP 36
#define KP_THROTTLE_RAISEUP 0   // No speed control on raiseup
#define KI_THROTTLE_RAISEUP 0.0

#define MAX_CONTROL_OUTPUT 500

// Servo definitions
#define SERVO_AUX_NEUTRO 1550  // Servo neutral position
#define SERVO_MIN_PULSEWIDTH 650
#define SERVO_MAX_PULSEWIDTH 2600

#define DEBUG 0   // 0 = No debug info (default)

#define CLR(x,y) (x&=(~(1<<y)))
#define SET(x,y) (x|=(1<<y))

#define ZERO_SPEED 65535
#define MAX_ACCEL 8        // Maximun motor acceleration (MAX RECOMMENDED
VALUE: 8) (default:7)

#define MICROSTEPPING 16   // 8 or 16 for 1/8 or 1/16 driver microstepping
(default:16)

#define I2C_SPEED 400000L  // 400kHz I2C speed
```

Hier worden libraries toegevoegd en enkele standaardinstellingen ingesteld, zoals: Baudrate snelheid (9600bps) (voor Bluetooth), Maximum parameters, PID parameters, servomotor,...

```
#define RAD2GRAD 57.2957795
#define GRAD2RAD 0.01745329251994329576923690768489

#define ITERM_MAX_ERROR 25   // Iterm windup constants for PI control //40
#define ITERM_MAX 8000       // 5000

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 =
success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (for us 18 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[18]; // FIFO storage buffer
Quaternion q;

long timer_old;
long timer_value;
int debug_counter;
float debugVariable;
float dt;

// class default I2C address is 0x68 for MPU6050
MPU6050 mpu;

// Angle of the robot (used for stability control)
float angle_adjusted;
float angle_adjusted_Old;

// Default control values from constant definitions
float Kp = KP;
float Kd = KD;
float Kp_thr = KP_THROTTLE;
float Ki_thr = KI_THROTTLE;
float Kp_user = KP;
float Kd_user = KD;
float Kp_thr_user = KP_THROTTLE;
float Ki_thr_user = KI_THROTTLE;
float PID_errorSum;
float PID_errorOld = 0;
float PID_errorOld2 = 0;
float setPointOld = 0;
float target_angle;
float throttle;
float steering;
float max_throttle = MAX_THROTTLE;
float max_steering = MAX_STEERING;
float max_target_angle = MAX_TARGET_ANGLE;
float control_output;

uint8_t mode;  // mode = 0 Normal mode, mode = 1 Pro mode (More agressive)

int16_t motor1;
int16_t motor2;

int16_t speed_M1, speed_M2;        // Actual speed of motors
int8_t  dir_M1, dir_M2;            // Actual direction of steppers motors
int16_t actual_robot_speed;        // overall robot speed (measured from
steppers speed)
int16_t actual_robot_speed_Old;
float estimated_speed_filtered;    // Estimated robot speed
```

Variabelen die gebruikt worden bij het ontvangen van de informatie van de gyrosensor, hierbij wordt DMP (Digital Motion Processing) toegepast.

Standaardvariabelen die aangeroepen kunnen worden in het programma om de code te “vereenvoudigen”.

De robot gebruikt, voor de throttle en steering, waarden tussen 0,5 en -0,5. Deze meegegeven waarde zal straks - 0,5 gedaan worden, waardoor het resultaat 0 wordt.

```
float throttleHC = 0.5;
float steeringHC = 0.5;

// DMP FUNCTIONS
// This function defines the weight of the accel on the sensor fusion
// default value is 0x80
// The official invensense name is inv_key_0_96 (??)
void dmpSetSensorFusionAccelGain(uint8_t gain){
  // INV_KEY_0_96
  mpu.setMemoryBank(0);
  mpu.setMemoryStartAddress(0x60);
  mpu.writeMemoryByte(0);
  mpu.writeMemoryByte(gain);
  mpu.writeMemoryByte(0);
  mpu.writeMemoryByte(0);
}
```

We gaan beginnen lezen in register 0x60 van de MPU-6050.

```
// Quick calculation to obtain Phi angle from quaternion solution (from DMP
internal quaternion solution)
float dmpGetPhi() {
  mpu.getFIFOBytes(fifoBuffer, 16); // We only read the quaternion
  mpu.dmpGetQuaternion(&q, fifoBuffer);
  mpu.resetFIFO();  // We always reset FIFO

  //return( asin(-2*(q.x * q.z - q.w * q.y)) * 180/M_PI); //roll
  //return Phi angle (robot orientation) from quaternion DMP output
  return (atan2(2 * (q.y * q.z + q.w * q.x), q.w * q.w - q.x * q.x - q.y *
q.y + q.z * q.z) * RAD2GRAD);
}
```

We lezen de quaternio-nen uit het FIFO register en doen hier een berekening mee.

```
// PD controller implementation (Proportional, derivative). DT is in
milliseconds
float stabilityPDControl(float DT, float input, float setPoint,  float Kp,
float Kd)
{
  float error;
  float output;

  error = setPoint - input;

  // Kd is implemented in two parts
  //    The biggest one using only the input (sensor) part not the SetPoint
input-input(t-2)
  //    And the second using the setpoint to make it a bit more aggressive
setPoint-setPoint(t-1)
  output = Kp * error + (Kd * (setPoint - setPointOld) - Kd * (input -
PID_errorOld2)) / DT;
  //Serial.print(Kd*(error-PID_errorOld));Serial.print("\t");
  PID_errorOld2 = PID_errorOld;
  PID_errorOld = input;  // error for Kd is only the input component
  setPointOld = setPoint;
  return (output);
}
```

Dit is de PD functie voor de stabiliteit, het blijven rechtstaan van de robot dus.

Hierboven zien we een softwarematige PD regelaar. Door de vooraf ingestelde “Kp” en “Kd” parameter, zal de robot heviger of minder hevig reageren.

```
// PI controller implementation (Proportional, integral). DT is in
milliseconds
float speedPIControl(float DT, float input, float setPoint,  float Kp,
float Ki)
{
  float error;
  float output;

  error = setPoint - input;
  PID_errorSum += constrain(error, -ITERM_MAX_ERROR, ITERM_MAX_ERROR);
  PID_errorSum = constrain(PID_errorSum, -ITERM_MAX, ITERM_MAX);

  //Serial.println(PID_errorSum);

  output = Kp * error + Ki * PID_errorSum * DT * 0.001; // DT is in
milliseconds...
  return (output);
}

// 16 single cycle instructions = 1us at 16Mhz
void delay_1us()
{
  __asm__ __volatile__ (
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop");
}

// TIMER 1 : STEPPER MOTOR1 SPEED CONTROL
ISR(TIMER1_COMPA_vect){
  if (dir_M1 == 0) // If we are not moving we dont generate a pulse
    return;
  // We generate 1us STEP pulse
  SET(PORTB, 7); // STEP MOTOR 1
  delay_1us();
  CLR(PORTB, 7);
}
// TIMER 3 : STEPPER MOTOR2 SPEED CONTROL
ISR(TIMER3_COMPA_vect){
  if (dir_M2 == 0) // If we are not moving we dont generate a pulse
    return;
  // We generate 1us STEP pulse
  SET(PORTD, 6); // STEP MOTOR 2
  delay_1us();
  CLR(PORTD, 6);
}
```

Dit is de PI functie voor de snelheid, bewegen, vooruit en achteruit van de robot dus.

Softwarematige PI regelaar. Door de vooraf ingestelde “Kp” en “Ki” parameter, zal de robot heviger of minder hevig reageren.

We geven pulsen van 1µs op de twee pins waaraan de motoren zijn aangesloten om deze te laten draaien.

```
// Set speed of Stepper Motor1
// tspeed could be positive or negative (reverse)
void setMotorSpeedM1(int16_t tspeed){
  long timer_period;
  int16_t speed;

  // Limit max speed?

  // WE LIMIT MAX ACCELERATION of the motors
  if ((speed_M1 - tspeed) > MAX_ACCEL)
    speed_M1 -= MAX_ACCEL;
  else if ((speed_M1 - tspeed) < -MAX_ACCEL)
    speed_M1 += MAX_ACCEL;
  else
    speed_M1 = tspeed;

#if MICROSTEPPING==16
  speed = speed_M1 * 46; // Adjust factor from control output speed to real
motor speed in steps/second
#else
  speed = speed_M1 * 23; // 1/8 Microstepping
#endif

  if (speed == 0){
    timer_period = ZERO_SPEED;
    dir_M1 = 0;
  } else if (speed > 0){
    timer_period = 2000000 / speed; // 2Mhz timer
    dir_M1 = 1;
    SET(PORTB, 4); // DIR Motor 1 (Forward)
  } else {
    timer_period = 2000000 / -speed;
    dir_M1 = -1;
    CLR(PORTB, 4); // Dir Motor 1
  }
  if (timer_period > 65535)   // Check for minimum speed (maximum period
without overflow)
    timer_period = ZERO_SPEED;

  OCR1A = timer_period;
  // Check  if we need to reset the timer...
  if (TCNT1 > OCR1A)
    TCNT1 = 0;
}


// Set speed of Stepper Motor2
// tspeed could be positive or negative (reverse)
void setMotorSpeedM2(int16_t tspeed){
  long timer_period;
  int16_t speed;

  // Limit max speed?

  // WE LIMIT MAX ACCELERATION of the motors
  if ((speed_M2 - tspeed) > MAX_ACCEL)
    speed_M2 -= MAX_ACCEL;
  else if ((speed_M2 - tspeed) < -MAX_ACCEL)
    speed_M2 += MAX_ACCEL;
  else
    speed_M2 = tspeed;
```

Functie om de maximale snelheid niet te overschrijden bij motor 1.

De draaizin van de motoren wordt ingesteld en op de juiste pin uitgestuurd om motor 1 aan te sturen.

Functie om de maximale snelheid niet te overschrijden bij motor 2.

```
#if MICROSTEPPING==16
  speed = speed_M2 * 46; // Adjust factor from control output speed to real
motor speed in steps/second
#else
  speed = speed_M2 * 23; // 1/8 Microstepping
#endif

  if (speed == 0)
  {
    timer_period = ZERO_SPEED;
    dir_M2 = 0;
  }
  else if (speed > 0)
  {
    timer_period = 2000000 / speed; // 2Mhz timer
    dir_M2 = 1;
    CLR(PORTC, 6);   // Dir Motor2 (Forward)
  }
  else
  {
    timer_period = 2000000 / -speed;
    dir_M2 = -1;
    SET(PORTC, 6);  // DIR Motor 2
  }
  if (timer_period > 65535)   // Check for minimun speed (maximun period
without overflow)
    timer_period = ZERO_SPEED;

  OCR3A = timer_period;
  // Check if we need to reset the timer...
  if (TCNT3 > OCR3A)
    TCNT3 = 0;
}

// INITIALIZATION
void setup()
{
  // STEPPER PINS ON JJROBOTS BROBOT BRAIN BOARD
  pinMode(4, OUTPUT); // ENABLE MOTORS
  pinMode(11, OUTPUT); // STEP MOTOR 1 PORTE,6
  pinMode(8, OUTPUT); // DIR MOTOR 1  PORTB,4
  pinMode(12, OUTPUT); // STEP MOTOR 2 PORTB,7
  pinMode(5, OUTPUT); // DIR MOTOR 2  PORTC,6
  digitalWrite(4, HIGH);  // Disable motors
  //-- Serial.begin(9600);
  //Serial1.begin(9600); //initialize RS232 comms on TX & RX pin to
communicate with HC06 Bluetooth module
  UCSR1B = (1 << RXEN1 ) | (1 << TXEN1 );   // Enable uart1 for transmit
and receive
  UCSR1C = (1 << UCSZ10 ) | (1 << UCSZ11 );
  UBRR1H = (BAUD_PRESCALE >> 8);       // Set baud rate register
  UBRR1L = BAUD_PRESCALE;              // Set baud rate register
  UCSR1B |= (1 << RXCIE1);// Enable interrupt when receive byte is complete
  sei();                               // Enable global interrupts

  // Initialize I2C bus (MPU6050 is connected via I2C)
  Wire.begin();
  // I2C 400Khz fast mode
  TWSR = 0;
  TWBR = ((16000000L / I2C_SPEED) - 16) / 2;
  TWCR = 1 << TWEN;
```

De draaizin van de motoren wordt ingesteld en op de juiste pin uitgestuurd om motor 2 aan te sturen.

De juiste out- en inputs worden geïnitialiseerd.
Als pin 4 “0” is, danwordt de geïnverteerde “enable” pin laag getrokken waardoor de motoren aan staan

Bluetooth instellen

I²C instellen

Wacht even om zeker te zijn dat alles in orde is, en dat alle connecties op punt staan.

```
#if DEBUG > 0
  delay(9000);
#else
  delay(2000);
#endif
  //--Serial.println("BALANCING ROBOT by Wannes Op de Beeck v3.0");
  //--Serial.println("Initializing I2C devices...");
  // mpu.initialize();
  // Manual MPU initialization... accel=2G, gyro=2000°/s, filter=20Hz BW,
output=1kHz
  mpu.setClockSource(MPU6050_CLOCK_PLL_ZGYRO);
  mpu.setFullScaleGyroRange(MPU6050_GYRO_FS_2000);
  mpu.setFullScaleAccelRange(MPU6050_ACCEL_FS_2);
  mpu.setDLPFMode(MPU6050_DLPF_BW_10);  //10,20,42,98,188  // Default
factor for BROBOT:10
  mpu.setRate(0);   // 0=1khz 1=500hz, 2=333hz, 3=250hz 4=200hz
  mpu.setSleepEnabled(false);
  delay(500);

  //--Serial.println("Initializing DMP...");
  devStatus = mpu.dmpInitialize();
  if (devStatus == 0) {
    // turn on the DMP, now that it's ready
    //--Serial.println("Enabling DMP...");
    mpu.setDMPEnabled(true);
    mpuIntStatus = mpu.getIntStatus();
    dmpReady = true;
  } else { // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    //--Serial.print("DMP Initialization failed (code ");
    //--Serial.print(devStatus);
    //--Serial.println(")");
  }

  // Gyro calibration, the robot must be steady during initialization
  delay(500);
  //--Serial.println("Gyro calibration!!  Dont move the robot in 10
seconds... ");
  delay(500);

  timer_old = millis();
  // STEPPER MOTORS INITIALIZATION
  // MOTOR1 => TIMER1
  TCCR1A = 0;             // Timer1 CTC mode 4, OCxA,B outputs disconnected
  TCCR1B = (1 << WGM12) | (1 << CS11); // Prescaler=8, => 2Mhz
  OCR1A = ZERO_SPEED;     // Motor stopped
  dir_M1 = 0;
  TCNT1 = 0;

  // MOTOR2 => TIMER3
  TCCR3A = 0;             // Timer3 CTC mode 4, OCxA,B outputs disconnected
  TCCR3B = (1 << WGM32) | (1 << CS31); // Prescaler=8, => 2Mhz
  OCR3A = ZERO_SPEED;     // Motor stopped
  dir_M2 = 0;
  TCNT3 = 0;
```

Je kan via Arduino automatisch de “mpu.initialize()” functie oproepen, maar omdat wij zeker willen zijn van de juiste instellingen, doen we dit “handmatig”. We kiezen hier ook voor de hoogste frequentie voor het verzenden van de meetresultaten.

```
  //Adjust sensor fusion gain
  dmpSetSensorFusionAccelGain(0x20);

  delay(200);

  // Enable stepper drivers and TIMER interrupts
  digitalWrite(4, LOW);    // Enable stepper drivers
  // Enable TIMERs interrupts
  TIMSK1 |= (1 << OCIE1A); // Enable Timer1 interrupt
  TIMSK3 |= (1 << OCIE1A); // Enable Timer1 interrupt


  // Little motor vibration and servo move to indicate that robot is ready
  for (uint8_t k = 0; k < 5; k++){
    setMotorSpeedM1(5);
    setMotorSpeedM2(5);
    delay(200);
    setMotorSpeedM1(-5);
    setMotorSpeedM2(-5);
    delay(200);
  }

  mpu.resetFIFO();
  timer_old = millis();
  mode = 0;
}


// MAIN LOOP
void loop()
{
  timer_value = millis();

  throttle = (throttleHC - 0.5) * max_throttle;
  // We add some exponential on steering to smooth the center band
  steering = steeringHC - 0.5;
  if (steering > 0){
    steering = (steering * steering + 0.5 * steering) * max_steering;
  } else {
    steering = (-steering * steering + 0.5 * steering) * max_steering;
  }


  // New DMP Orientation solution?
  fifoCount = mpu.getFIFOCount();
  if (fifoCount >= 18){
    if (fifoCount > 18){ // If we have more than one packet we take the
easy path: discard the buffer and wait for the next one
      mpu.resetFIFO();
      return;
    }
    dt = (timer_value - timer_old);
    timer_old = timer_value;

    angle_adjusted_Old = angle_adjusted;
    // Get new orientation angle from IMU (MPU6050)
    angle_adjusted = dmpGetPhi();
```

Twee verschillende interrupts voor de twee motoren.

De via Bluetooth ontvangen waarden worden in de formules voor kantelen en sturen gestoken. Bij sturen zorgt de formule voor een exponentieel resultaat, waardoor het sturen geleidelijk aan gebeurt.

De maximum grootte van een FIFO pakket is 18, als het groter is, dient de FIFO gereset te worden en wordt er een nieuwe waarde ingelezen.

Hier wordt de ogenblikkelijke hoek waarin de robot zich bevindt, ingelezen.

```
    /*Serial.print(throttle);
    Serial.print(" ");
    Serial.print(steering);
    Serial.print(" ");
    Serial.println(mode);

    Serial.println(angle_adjusted);*/

    mpu.resetFIFO();  // We always reset FIFO

    // We calculate the estimated robot speed:
    // Estimated_Speed = angular_velocity_of_stepper_motors(combined) -
angular_velocity_of_robot(angle measured by IMU)
    actual_robot_speed_Old = actual_robot_speed;
    actual_robot_speed = (speed_M1 + speed_M2) / 2; // Positive: forward

    int16_t angular_velocity = (angle_adjusted - angle_adjusted_Old) *
90.0; // 90 is an empirical extracted factor to adjust for real units
    int16_t estimated_speed = -actual_robot_speed_Old - angular_velocity;
// We use robot_speed(t-1) or (t-2) to compensate the delay
    estimated_speed_filtered = estimated_speed_filtered * 0.95 +
(float)estimated_speed * 0.05;  // low pass filter on estimated speed


    // SPEED CONTROL: This is a PI controller.
    //    input:user throttle, variable: estimated robot speed, output:
target robot angle to get the desired speed
    target_angle = speedPIControl(dt, estimated_speed_filtered, throttle,
Kp_thr, Ki_thr);
    target_angle = constrain(target_angle, -max_target_angle,
max_target_angle); // limited output

    // Stability control: This is a PD controller.
    //    input: robot target angle(from SPEED CONTROL), variable: robot
angle, output: Motor speed
    //    We integrate the output (sumatory), so the output is really the
motor acceleration, not motor speed.
    control_output += stabilityPDControl(dt, angle_adjusted, target_angle,
Kp, Kd);
    control_output = constrain(control_output, -MAX_CONTROL_OUTPUT,
MAX_CONTROL_OUTPUT); // Limit max output from control


    // The steering part from the user is injected directly on the output
    motor1 = control_output + steering;
    motor2 = control_output - steering;

    // Limit max speed (control output)
    motor1 = constrain(motor1, -MAX_CONTROL_OUTPUT, MAX_CONTROL_OUTPUT);
    motor2 = constrain(motor2, -MAX_CONTROL_OUTPUT, MAX_CONTROL_OUTPUT);

    // NOW we send the commands to the motors
    if ((angle_adjusted < 70) && (angle_adjusted > -70)) // Is robot ready
(upright?)
    {
      // NORMAL MODE
      digitalWrite(4, LOW);  // Motors enable
      setMotorSpeedM1(motor1);
      setMotorSpeedM2(motor2);
```

Hier wordt de nodige snelheid berekend om recht te blijven met de hoek die we net gemeten hebben.

Het exponentiële sturen van daarnet, wordt hier rechtstreeks op de output gezet.

Als de gemeten hoek tussen 70 en -70 graden zit, dan mogen de motoren werken.

```
      // Normal condition?
      if ((angle_adjusted < 45) && (angle_adjusted > -45)){
        Kp = Kp_user;             // Default user control gains
        Kd = Kd_user;
        Kp_thr = Kp_thr_user;
        Ki_thr = Ki_thr_user;
      } else {   // We are in the raise up procedure => we use special
control parameters
        Kp = KP_RAISEUP;          // CONTROL GAINS FOR RAISE UP
        Kd = KD_RAISEUP;
        Kp_thr = KP_THROTTLE_RAISEUP;
        Ki_thr = KI_THROTTLE_RAISEUP;
      }
    } else {  // Robot not ready (flat), angle > 70° => ROBOT OFF
      digitalWrite(4, HIGH);  // Disable motors
      setMotorSpeedM1(0);
      setMotorSpeedM2(0);
      PID_errorSum = 0;  // Reset PID I term
      Kp = KP_RAISEUP;   // CONTROL GAINS FOR RAISE UP
      Kd = KD_RAISEUP;
      Kp_thr = KP_THROTTLE_RAISEUP;
      Ki_thr = KI_THROTTLE_RAISEUP;
    }
  } // End of new IMU data
}

// interrupt routine for Bluetooth is called when the first byte is
received, We know that the second byte will follow imediately after the
first one so we read this second byte in the interrupt routine
ISR (USART1_RX_vect)      // interrupt routine that executes every time a
new byte is received
{
    UCSR1B &= ~(1 << RXCIE1);   // disable RX interrupt

    throttleHC = UDR1;                    // Fetch the received byte value
into the variable " ByteReceived " - the RXint flaq is automatically
cleared now
    throttleHC = throttleHC/255;
    while ((UCSR1A & (1 << RXC1)) == 0) {};  // wait until byte 2 is
received
                                          // Do nothing until data have
been received and is ready to be read from UDR
    steeringHC = UDR1;                    // receive byte 2
    steeringHC = steeringHC/255;
    UCSR1B |= (1 << RXCIE1);    //Re-enable RX interrupt

}
```

Als de robot tussen de 45 en -45 graden gekanteld staat, zullen de normale parameters gebruikt worden, anders worden de “raiseup” parameters gebruikt.

Throttle en steering worden ingelezen via Bluetooth met een interrupt routine.

Totaal besluit

De belangrijkste conclusies die we kunnen trekken uit voorgaande labo’s is dat spanningsdrops (veroorzaakt door inductiestromen van stappenmotoren) effectief kunnen opgevangen worden door de juiste condensatoren in de schakeling te integreren. Dit zijn er uiteindelijk wel vrij veel zijn (3 per motor), maar ik wou uiteraard ook zo’n compact mogelijke PCB. Het resultaat daarvan is dat ik mijn PCB te klein had gemaakt waardoor de componenten te dicht bij elkaar stonden, omdat ik niet

genoeg rekening had gehouden met de grootte van de componenten. Daardoor pasten ze niet allemaal op de PCB. Door deze stommiteit heb ik tot wel twee keer toe een nieuwe print moeten maken.
Het is raar dat de code voor I²C met de MPU-6050, enkel werkt bij Arduino, ondanks het feit dat de code in C, net hetzelfde is.
Het heeft even geduurd vooraleer ik de voorgeschreven code volledig snapte, omdat het ten eerste in Arduino IDE geschreven is, een programmeertaal waarin ik niet thuis ben, en met alle libraries wordt het één grote soep van benamingen waar soms moeilijk het nut van te achterhalen is. Het bevatte ook een verbinding via Wi-Fi, iets wat nog niet aan bod is gekomen in de lessen Elektronica/ICT.
Uit het energieverslag kunnen we afleiden dat de motoren (zoals verwacht) de grootste verbruikers zijn, maar dat mijn robotje, ondanks het grote vermogen van de motoren, toch vrij lang kan blijven werken zonder dat de batterijen dienen opgeladen te worden.
Ik persoonlijk, heb het maken van deze GIP zeer leuk gevonden. Al sinds ik een kleine jongen was, was ik gefascineerd door robotjes en wat voor leuke dingen je er allemaal mee kan doen. Het was dus een grote droom van mij om zulke dingen zelf te kunnen maken. Ik heb zeer veel geleerd uit de labo's die ik gedaan heb. Het is wel zo dat, door al deze labo's over alle aparte onderdelen, de GIP zelf pas helemaal op het einde vorm kreeg. Dat leidde wel tot enige stress om alles op tijd af te krijgen. Ik ben zeer tevreden met het resultaat en dat het werkt, want dat was nog wel even spannend toen de laatste maanden ingingen.

Bronnenlijst

Aliexpress (2016). MPU-6050 MPU6050 Module 3 Axis analoge gyrosensoren + 3 Axis Accelerometer Module. Geraadpleegd op 3 december 2016, https://nl.aliexpress.com/item/Free-Shipping-GY-521-3205-ITG-3205-module-Three-axis-gyroscope-module-In-stock/691012552.html?spm=2114.010208.3.1.rz1m6A&ws_ab_test=searchweb0_0,searchweb201602_3_10065_10068_10000009_10084_10083_10080_10082_10081_10110_10111_10112_10060_10113_10114_10062_10056_10055_10054_10059_10099_10078_10079_10000012_10103_10073_10102_10000015_10096_10052_10053_10107_10050_10106_10051,searchweb201603_10,afswitch_3,single_sort_0_price_asc&btsid=0f4d50b6-8932-49f9-8944-1204c85dbbbd

Bart Huyskens (2012). VIDEOLESSEN DEEL 3 COMMUNICATIE. Geraadpleegd op 24 november 2016, http://e2cre8.be/?page_id=13

Deckers, G. (2006). Signaalbewerking. Schoten: St. Jozefinstituut

Dejan Nedelkovski (18/11/2015). How MEMS Accelerometer Gyroscope Magnetometer Work & Arduino Tutorial. Geraadpleegd op 20 december 2016, https://www.youtube.com/watch?v=eqZgxR6eRjo

E2CRE8 (2016). Brainbox Arduino. Geraadpleegd op 21 oktober 2016, http://e2cre8.be/?page_id=21

Farnell (2016). NANOTEC ST4118M1804-A Stepper Motor, High Torque, DC, 0.28 N-m, 1.8 A, Two, 1.1 ohm, 1.85 mH. Geraadpleegd op 23 september 2016, http://be.farnell.com/nanotec/st4118m1804-a/stepper-motor-2vdc-1-8a/dp/2507563?ost=2507563&searchView=table&iscrfnonsku=false&ddkey=http%3Anl-BE%2FElement14_Belgium%2Fsearch

HobbyElectronica (2016). MPU-6050. Geraadpleegd op 30 september 2016, https://www.hobbyelectronica.nl/product/mpu-6050/

Julio J. (2016). THE B-ROBOT EVO (the self balancing robot). Geraadpleegd op 2 november 2016, http://www.jjrobots.com/b-robot-mounting-instructions/

Microchip (2016). Datasheet PIC16F887. Geraadpleegd op 14 oktober 2016, http://ww1.microchip.com/downloads/en/DeviceDoc/41291D.pdf

Pololu (2016). DRV8825 Stepper Motor Driver Carrier, High Current. Geraadpleegd op 14 oktober 2016, https://www.pololu.com/product/2133

Saelig (2016). PICO TA018 Current Clamp (60A AC/DC) with BNC and 3 metre screened lead (PP264). Geraadpleegd op 7 oktober 2016, http://www.saelig.com/product/PSA013.htm

University of Washington (2016). CSE 466 Lab 4: I2C Gyroscope control. Geraadpleegd op 18 november 2016, https://courses.cs.washington.edu/courses/cse466/14au/labs/l4/l4.html