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

OPDRACHT GEÏNTEGREERDE PROEF

BALANCERENDE ROBOT

Jelte Boumans ● 1
Studierichting TSO ● EA-ICT
Tweede leerjaar ● derde graad
Schooljaar 2019 ● 2020
Mentor/Leerkracht: Mr. Huyskens

Woord vooraf

Ik ben Jelte Boumans en ik ben een leerling uit het 6de jaar Elektronica-ICT aan het Sint-Jozefinstituut in Schoten.

Mijn keuze voor mijn geïntegreerde proef was het verder uitwerken van een project dat al 2 keer is gemaakt door andere leerlingen maar nog niet helemaal was afgerond. Het gaat om een balancerende robot. De delen die ik moest aanpassen waren de behuizing en de flexibiliteit van de robot. De reden dat ik dit heb gekozen is omdat ik al van kinds af aan nieuwsgierig ben hoe naar hoe een robot werkt, ik heb er dus al heel mijn leven interesse voor gehad. In mijn project heb ik veel kennis moeten toepassen van vorige jaren, maar zeker ook veel kennis van dingen die ik pas dit schooljaar heb geleerd. Ik wil met dit project bewijzen dat ik niet altijd voor de eenvoudige oplossing moet gaan, want dit project is niet heel eenvoudig maar ik weet dat ik het zou kunnen.

Dit was geen eenvoudig project om te maken, maar met de hulp die ik kreeg van de mensen rondom mij was het al zeker een stuk eenvoudiger. Ik bedank zeker meneer Huyskens om mij te helpen met de leerstof en om tips te geven in verband met vroegere fouten. Ik bedank mijn beste vriend Alex voor mij Arduino aan te leren en te lange programmeer tijden wat leuker te maken. Ik bedank zeker ook nog mijn klas om te helpen met alle individuelen projecten wanneer ik hulp nodig had. Ten slotte bedank ik zeker ook mijn moeder, om mij te steunen doorheen het jaar en helpen met planning en verslagen schrijven.

Inhoudsopgave

Inleiding

De keuze voor mijn GIP was het verder uitwerken van een balancerende robot die vergelijkbaar is met een Segway (= een zelf balancerende scooter). Het doel voor mij was om het project van de vorige leerlingen over te nemen en het te verbeteren en er extra delen er aan toe te voegen. Wat de robot uiteindelijk doet is zichzelf balanceren natuurlijk, maarde robot moest ook bestuurbaar zijn van op een afstand, via bluetooth.

Mijn GIP kan grootschaliger bekeken worden en de robot zou zo bijvoorbeeld als vervoermiddel gebruikt kunnen worden. Het was ook een nuttige opdracht voor mij want ik moest zeer recent geleerde leerstof direct kunnen toepassen (RS232, I2C, SPI) zodat alle componenten correct konden communiceren met elkaar. Voor elektronica moest ik eerst alle individuele ongekende componenten onderzoeken in labo's om ze te begrijpen en zo het concept van de oude robot proberen te verbeteren. Voor Engels moest ik foto's zoeken die met mijn GIP te maken hebben en daar een presentatie rond maken. Voor Frans moest ik een video vinden die te maken heeft met mijn GIP en daar dan ook een presentatie rond maken.

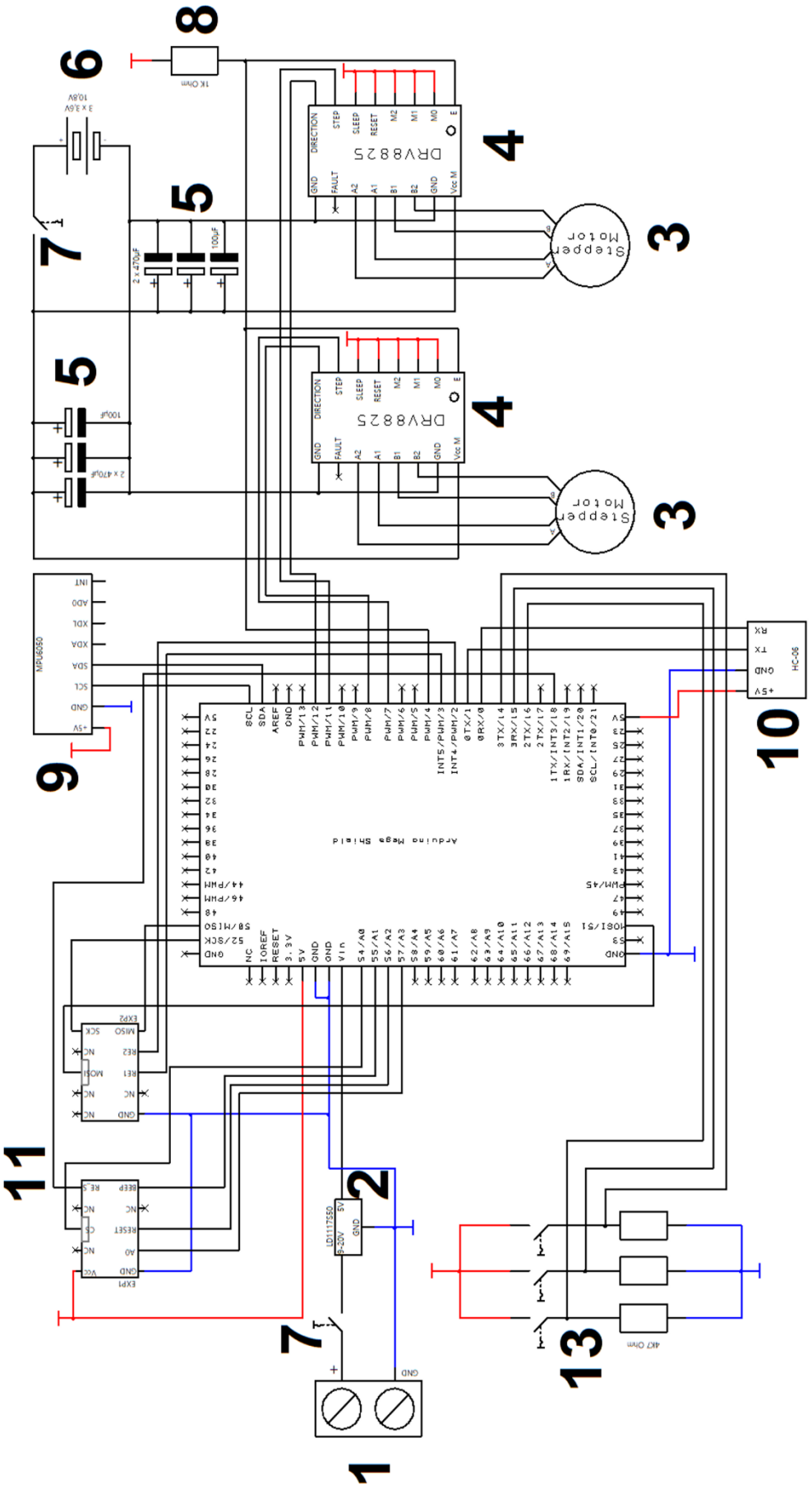
In deze bundel vindt u onder andere de labo's waar ik het eerder over had, met schema's, theoretische uitleg en meer. U vindt ook nog de componenten waar de robot uit is gemaakt en de code die de robot bestuurt. Ten slotte vindt u hier ook mijn verschillende vakopdrachten.

Projecten

Bestellijst

Componenten:	Farnell nummer:	Aantal:	Prijs:	Totaalprijs:
Arduino Mega 2560	Banggood	1	12,09 €	12,09 €
HC-06	Banggood	1	2,70 €	2,70 €
GY-521/MPU-6050	HobbyElectronica	1	3,95 €	3,95 €
DRV8825	Pololu	2	8,95 €	17,90 €
Stappenmotor	2507563	2	26,38 €	52,76 €
9V batterij	2503729	1	3,00 €	3,00 €
9V batterij connector	2646477	1	1,24 €	1,24 €
3,6V batterij (18650)	Conrad	3	7,70 €	23,10 €
3,6V batterij houder	Banggood	3	2,78 €	8,34 €
Condensator 470µF	1144616	4	0,26 €	1,04 €
Condensator 100µF	8767122	2	0,15 €	0,30 €
Weerstand 1KΩ	2614353	1	0,15 €	0,15 €
Male pin headers	9729038	1	1,67 €	1,67 €
Female pin headers	1593472	1	0,50 €	0,50 €
Schakelaar	7674180	2	3,66 €	7,32 €
			Totaalprijs van alles →	136,06 €

Schema’s


1
2
3
4
5
6
7
8
9
10
11
12
13
Stepper Motor
DRV8825
Arduino Mega Shield
MPU6050
HC-06
LD1117S50
2 x 470uF
100uF
10K Ohm
470 Ohm

Bespreking schema GIP

1. De schroefconnector waaraan een 9V batterij zal worden verbonden die de Arduino Mega van voeding zal voorzien.

2. Een low-dropout regulator (staat op Arduino) die de 9V van de batterij zal omvormen naar 5V voor de brainbox.

3. De stappenmotoren die de robot zijn wielen zullen aansturen.

4. De stappenmotordrivers (DRV8825) die het correcte signaal naar de motoren sturen om hun snelheid en draairichting te bepalen.

5. De 3 parallel geschakelde filter condensatoren die aan de voeding van beide drivers zijn aangesloten om spanningspieken te filteren (2 x 470µF, 1 x 100µF).

6. De 3 in parallel geschakelde batterijen (3 x 3.6V en 5000mAh, 18650) die dienen om de drivers en motoren van voeding te voorzien.

7. De 2 schakelaars die de voeding voor de motoren en Arduino zullen schakelen om zo de robot aan en uit te zetten.

8. Pull-up weerstaande voor de “Enable” ingang van de driver aan te sturen.

9. De MPU6050 die met zijn gyrosensor en accelerometer alle nodige bewegingen van de robot zal meten en naar de brainbox sturen via I2C.

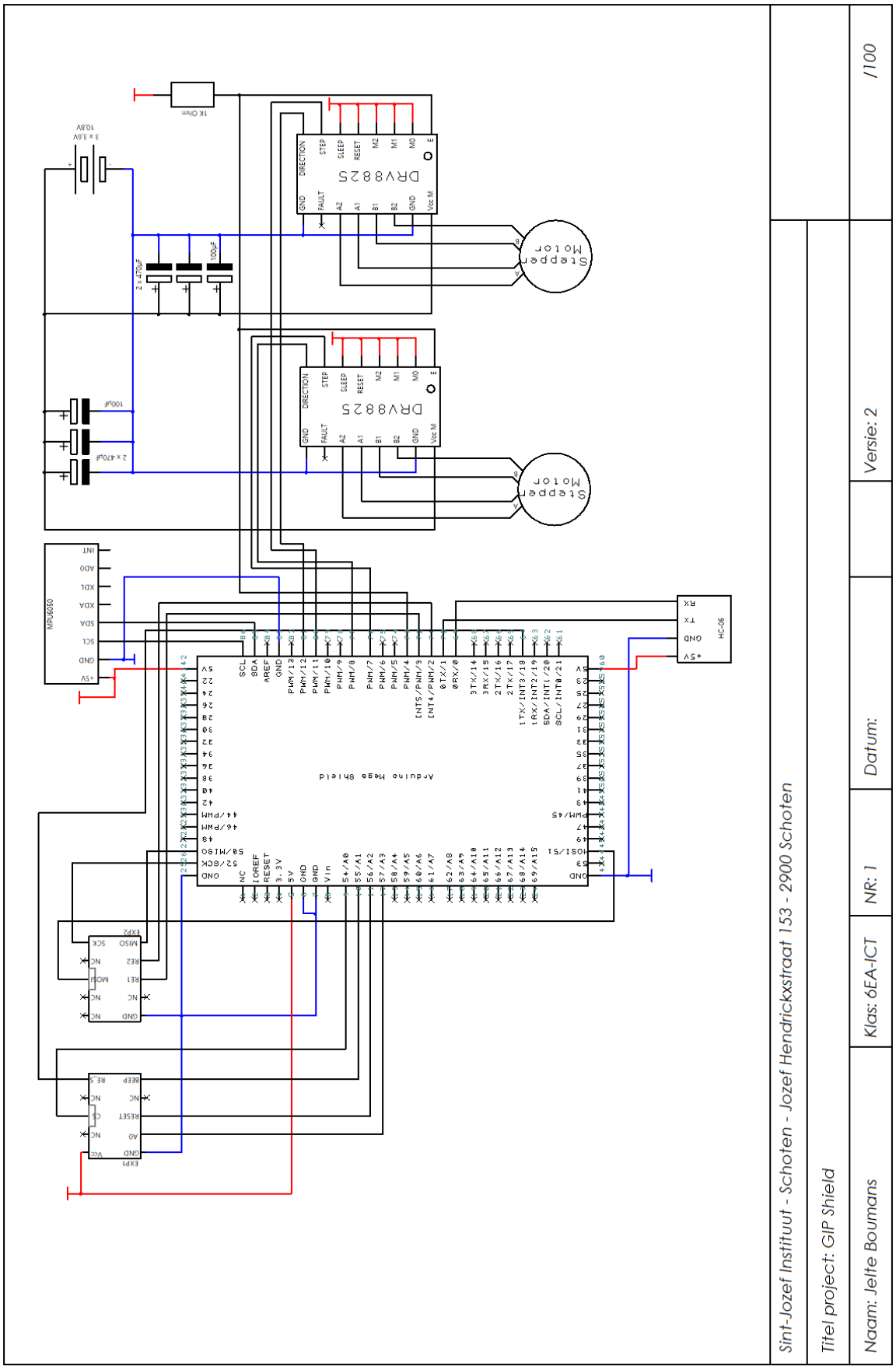
10. De bluetooth module (HC06) die verbinding zal maken via de zelfgemaakte app op je gsm om de robot via bluetooth te kunnen besturen.

11. De 2 connectoren (EXP1 en EXP2) van de GLCD die met SPI en I/O pinnen van de brainbox worden aangestuurd.

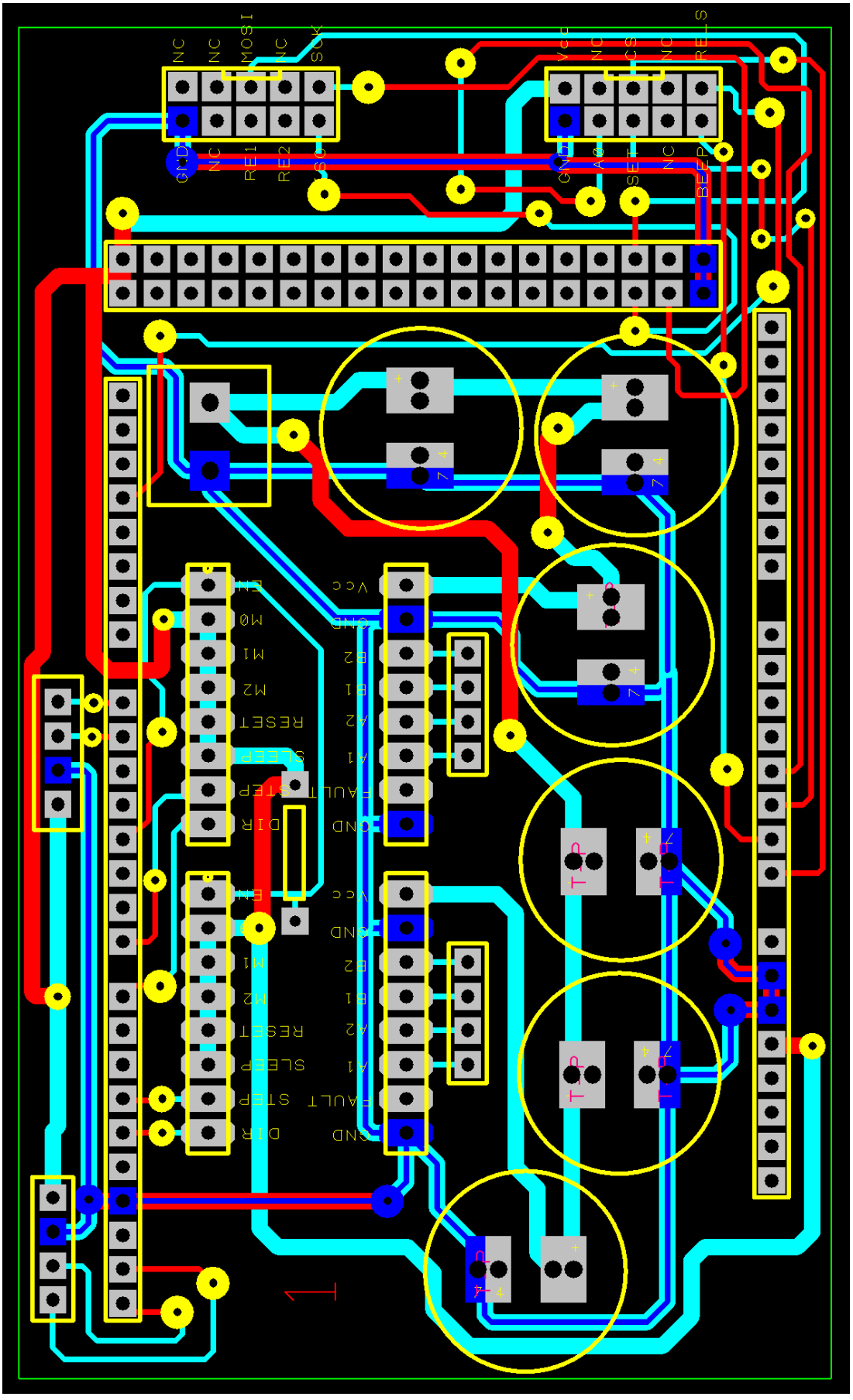
12. Het zelfgetekende Arduino Shield dat aan de Arduino Mega zal verbonden zijn, de drivers, HC06, MPU6050, etc. zijn hierop verbonden.

13. 3 Druknoppen voor te navigeren in het menu op de GLCD (de reden hiervoor is omdat de rotary encoder die origineel gebruikt ging worden kapot gegaan is).

Elektronisch Schema Shield

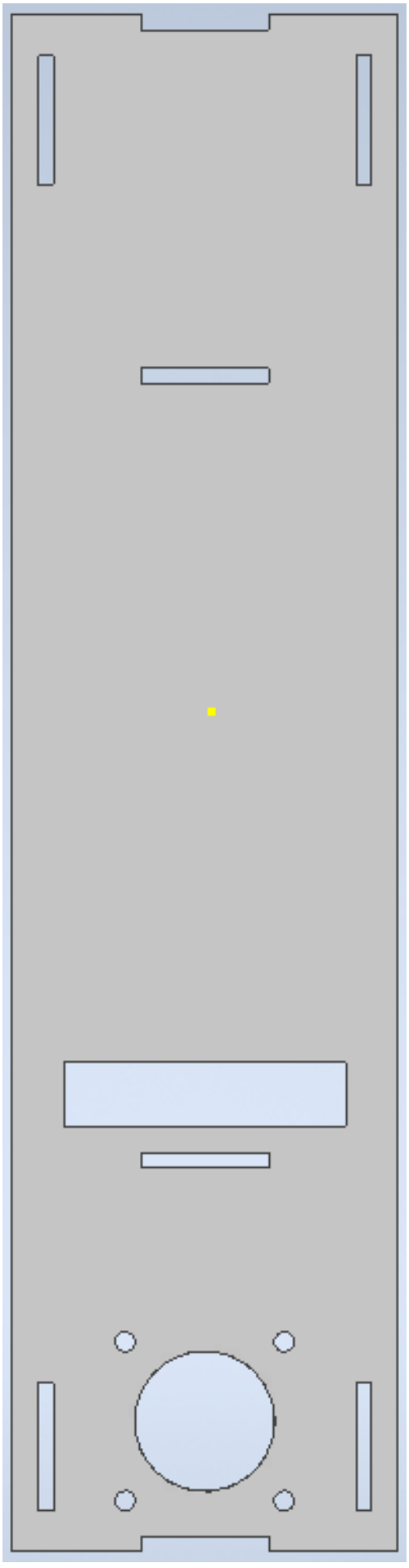


PCB Shield

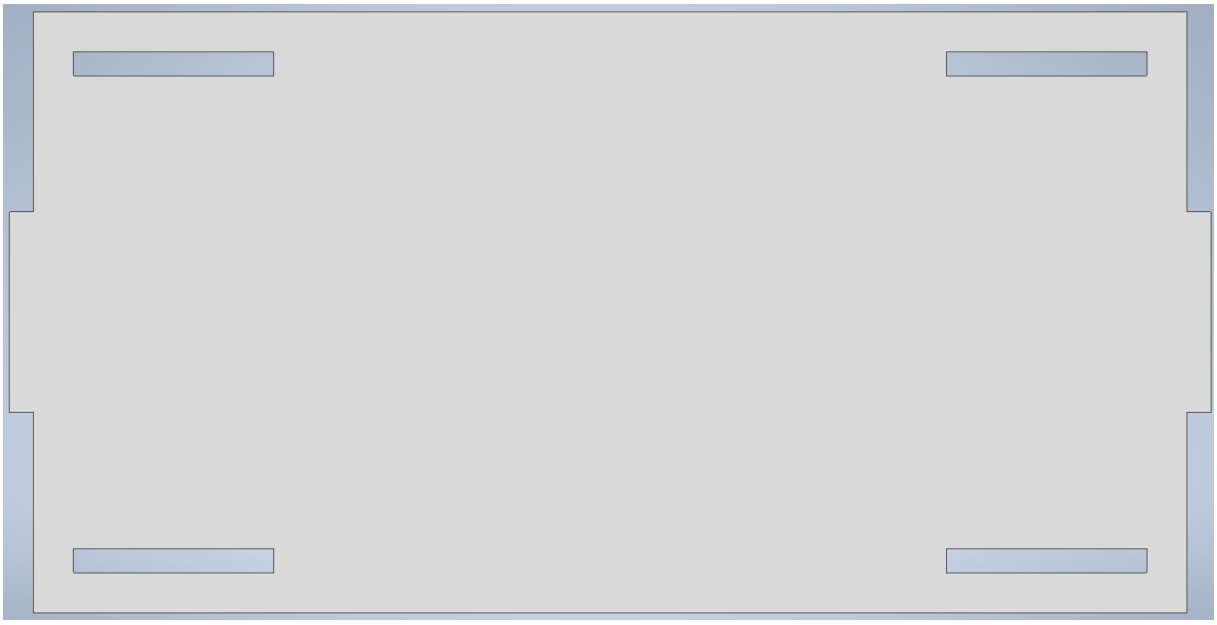

NC
NC
MOSI
NC
SCK
GND
NC
RE1
RE2
SCK
GND
VCC
A0
NC
SET
CS
NC
BUSY
RES
EN
M0
M1
M2
RESET
SLEEP
STEP
DIR
VCC
GND
E2
E1
A2
A1
FAULT

Inventor

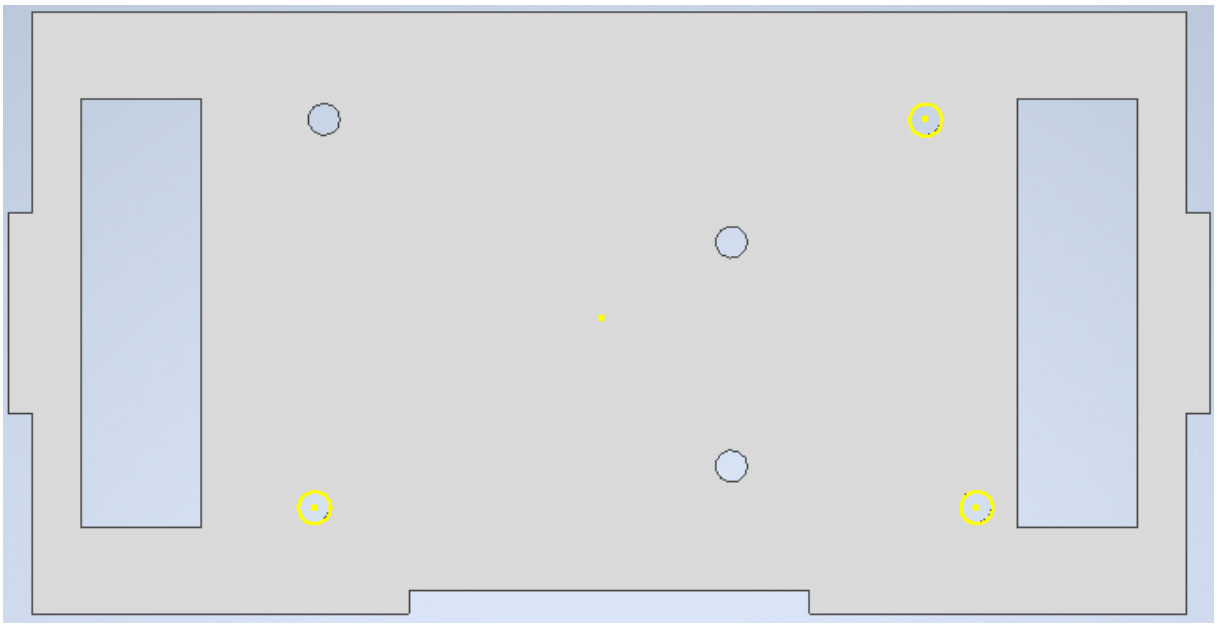
Muren



Level 1

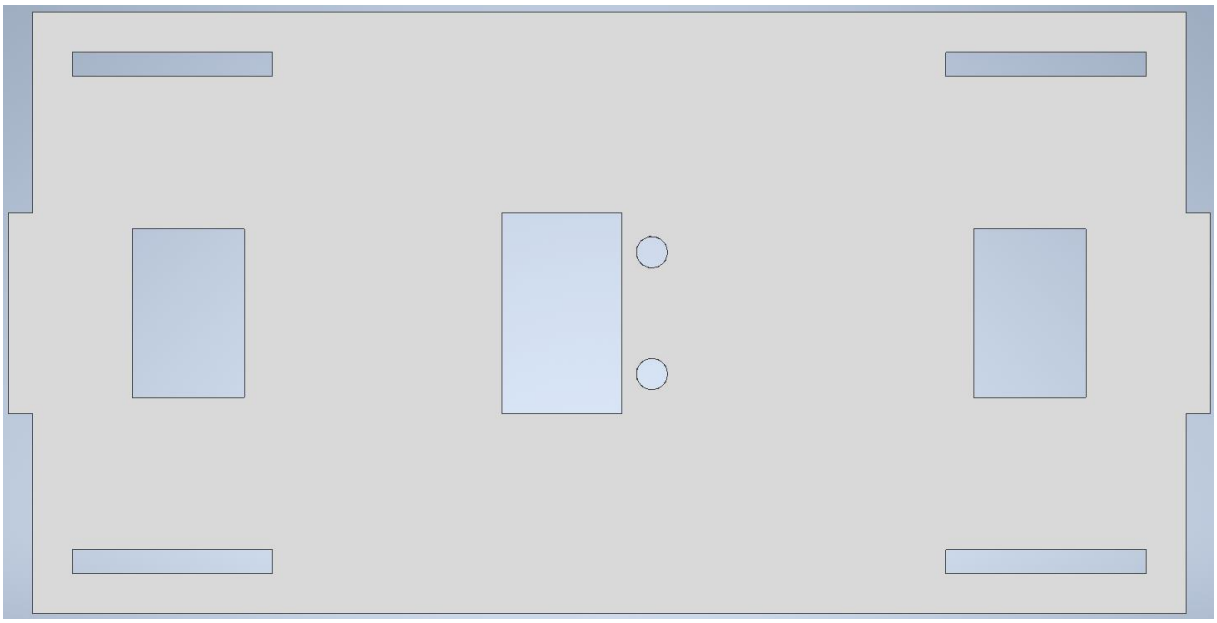


Level 2

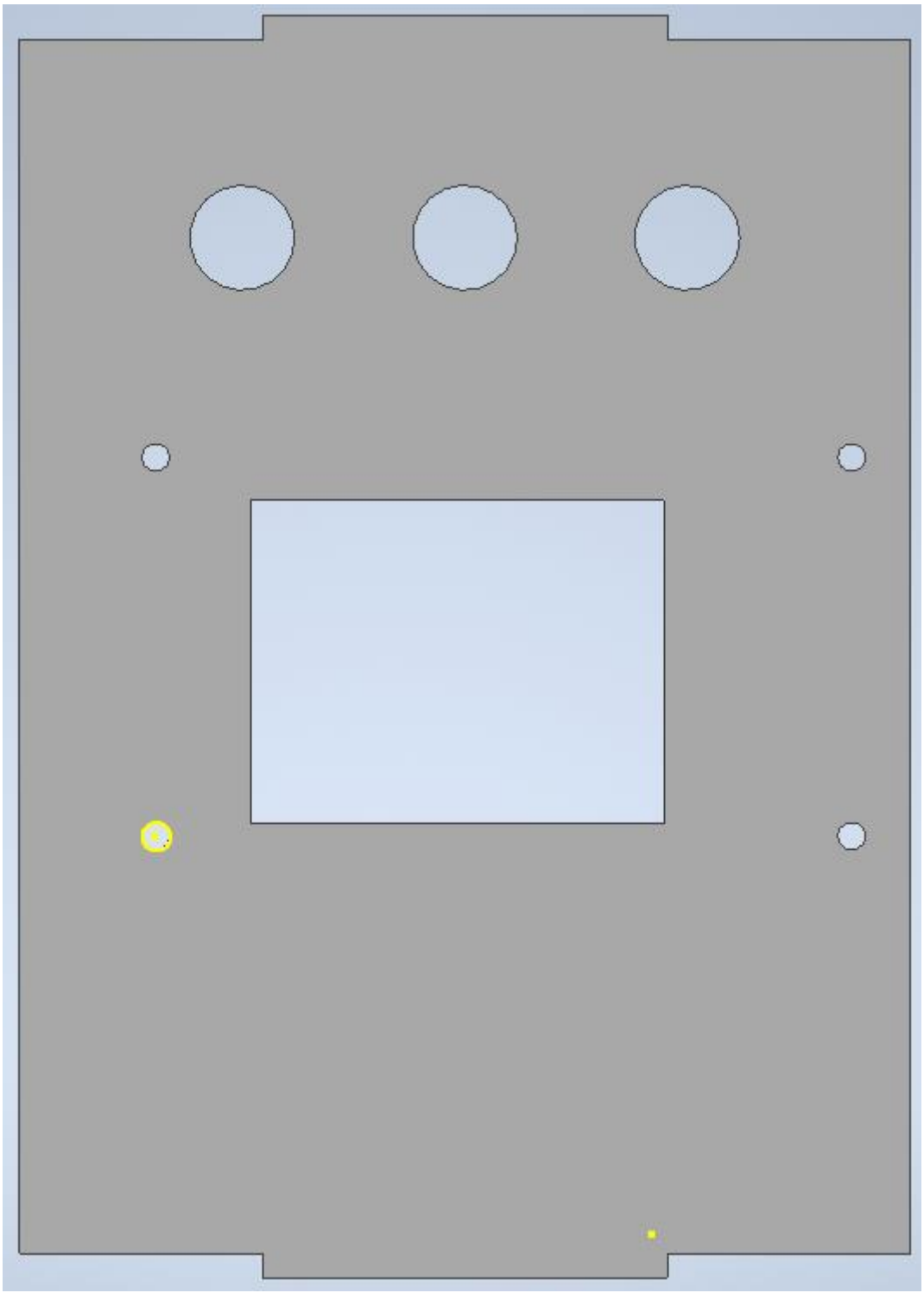


Level 3

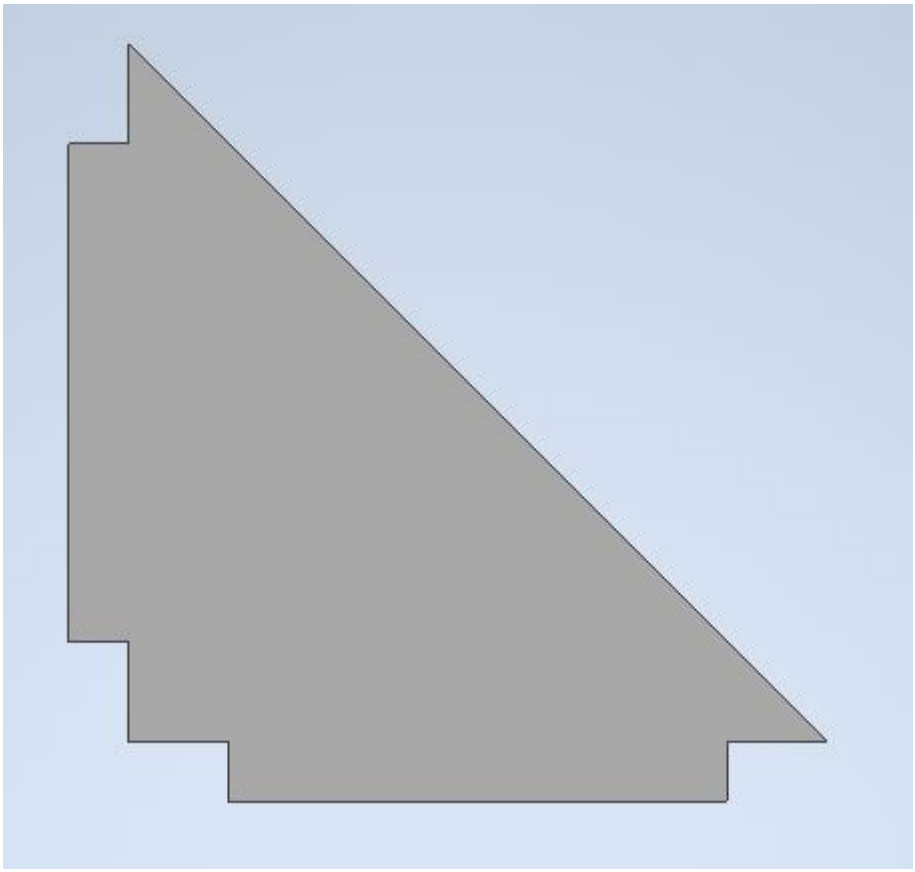
Level 4



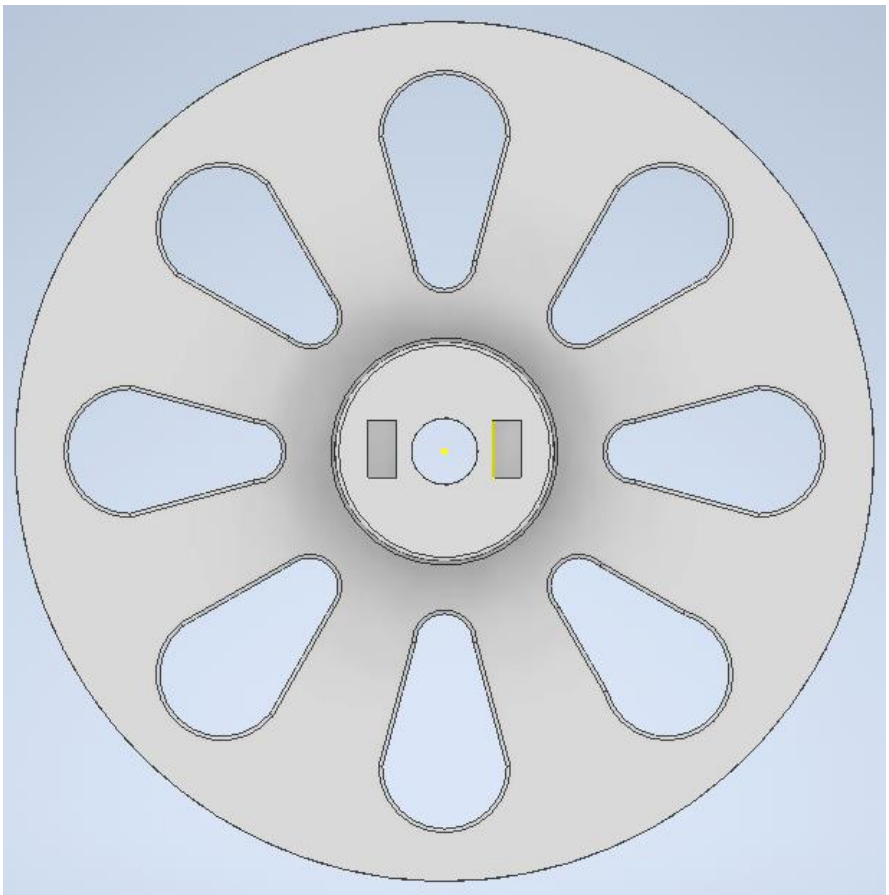
LCD Panel



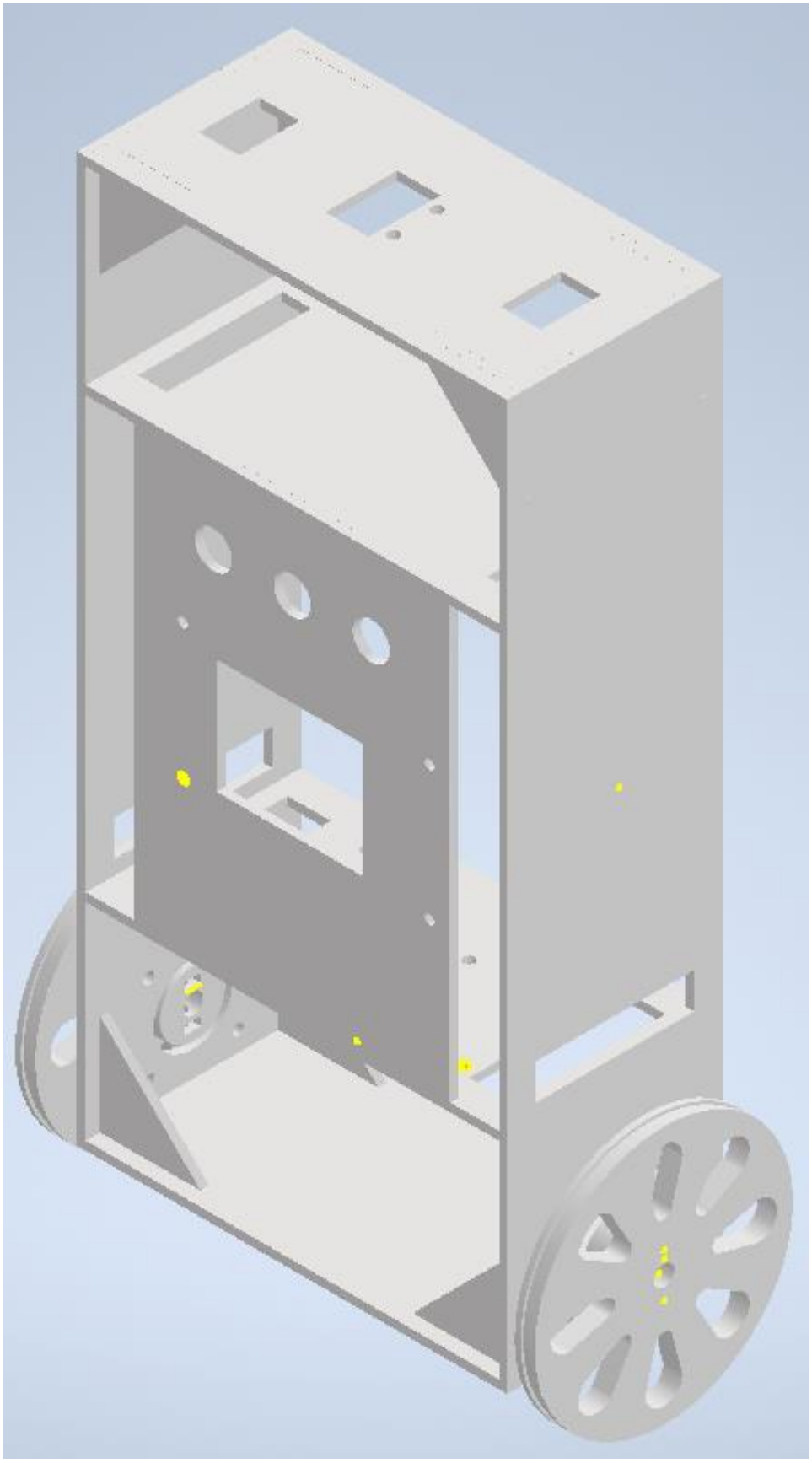
Support piece



Wiel



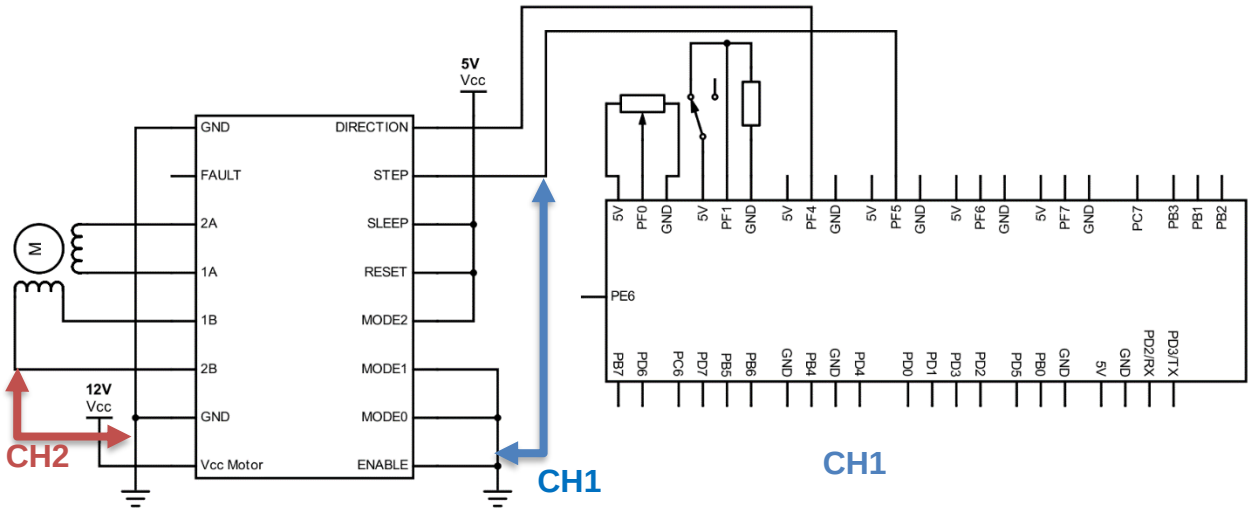
Assembly



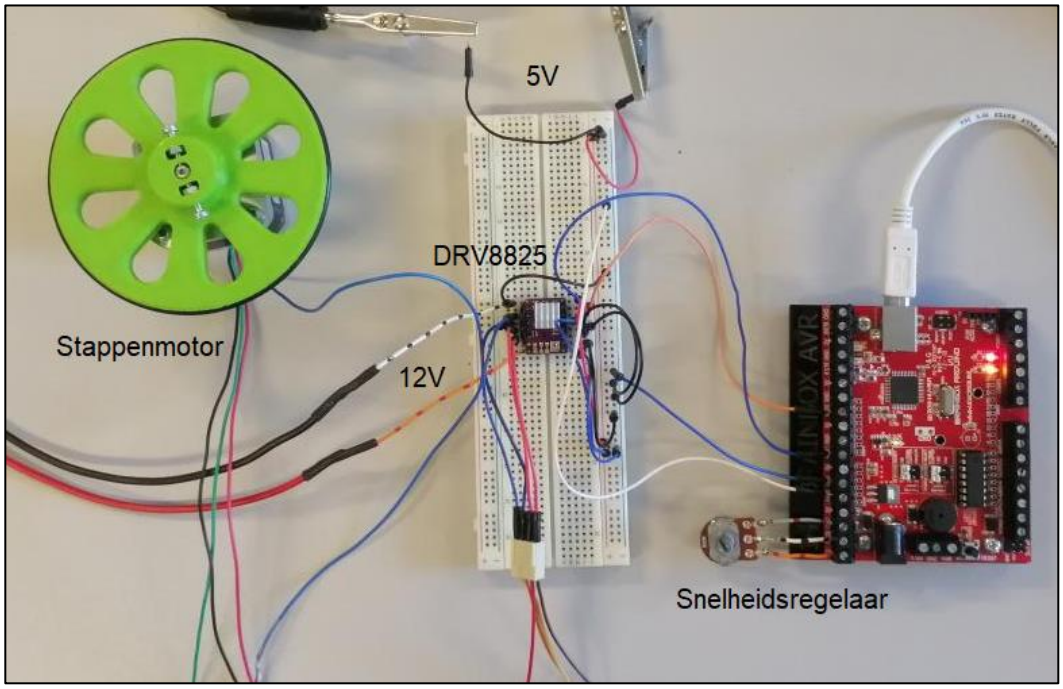
Labo Stappenmotor en Pololu DRV8825 driver

Doelstelling

Het doel van dit labo is om te begrijpen hoe ik een DRV8825 microstep driver moet aansturen via een ATMEGA32U4 en daarmee een stappenmotor laten draaien op verschillende modes. Want ik moet er voor de robot 2 tegelijkertijd kunnen aansturen en snel van snelheid en draairichting laten veranderen.



Schema

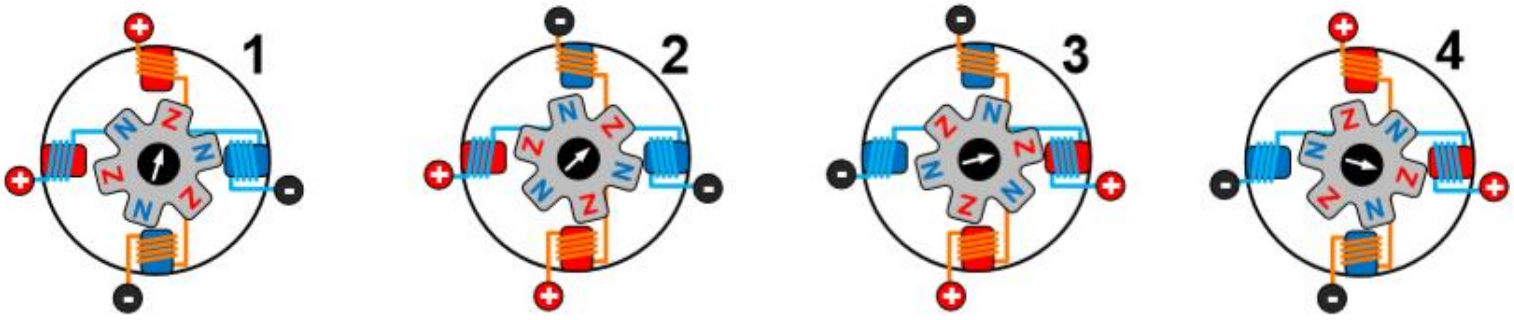


Het is een vrij simpel schema. Ik had mezelf de opdracht gegeven om een project te maken waarin ik een stappenmotor kon besturen. Ik zou de snelheid moeten kunnen regelen met een potentiometer en de draairichting kunnen veranderen met een soort schakelaar (mijn verwachtingen). Ik kan ook kiezen tussen Full step, Half step, 1/4 step, 1/8 step, 1/16 step en 1/32 step met de MODE pinnen (staat in datasheet) maar dat kon ik gewoon aanpassen door de draden op de breadboard te veranderen van 5V naar GND.

MODE0	MODE1	MODE2	Steppermode
0	0	0	Full Step
1	0	0	Half Step
0	1	0	1/4 Step
1	1	0	1/8 Step
0	0	1	1/16 Step
1	0	1	1/32 Step
X	1	1	1/32 Step

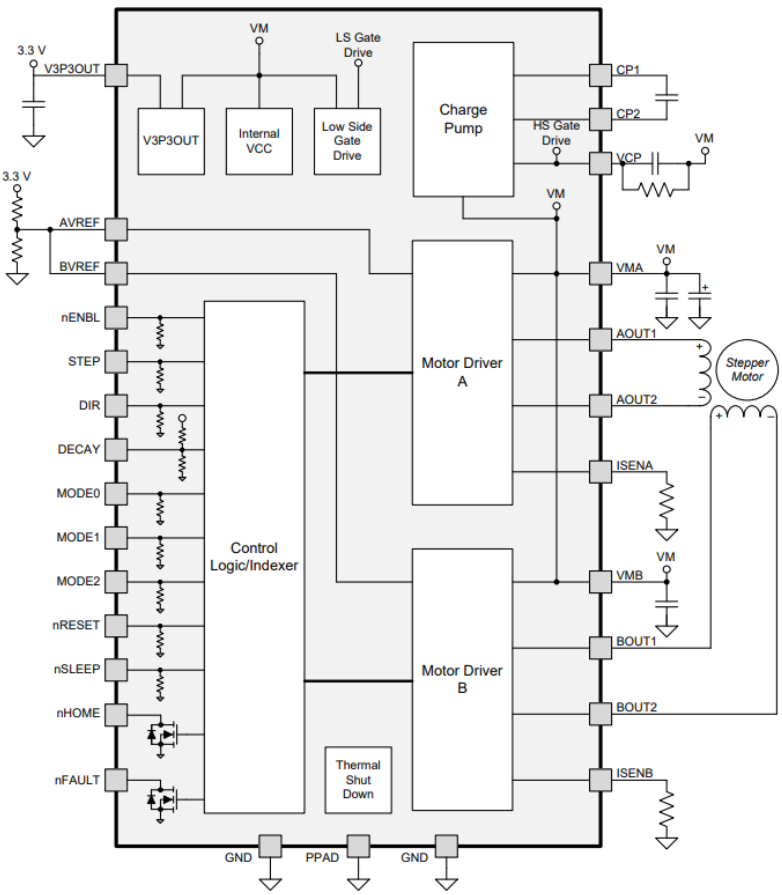
Stappenmotor

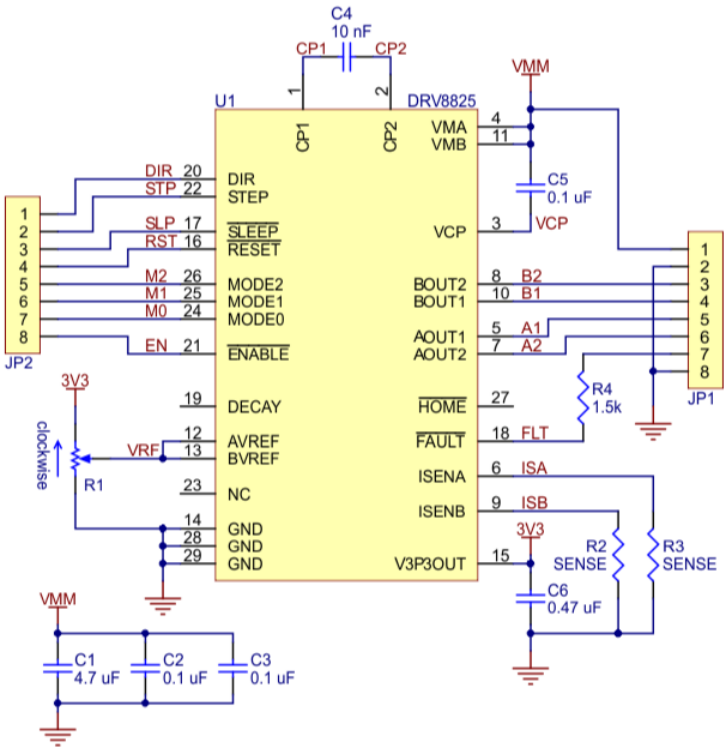
Stappenmotoren zijn een soort DC motoren die in stappen draaien i.p.v. in een constante cirkel. Er zijn 2 soorten stappenmotoren : een bipolaire en een unipolaire. Ik gebruik een bipolaire. In de foto hieronder zie je een voorbeeld van “fullstep” sturing. We zien 4 spoelen (stator) en een magneet met alternerende polen (rotor), de “+” betekent dat daar de stroom binnenkomt en de “-” betekent dat daar de stroom dan buitenkomt. De spoelen waar een “+” boven staat trekken dan de noordpool aan en de “-“ trekken de zuidpool aan. Door de 4 spoelen van de motor juist aan te sturen kunnen we de rotor in stappen laten draaien (zie foto hieronder). Natuurlijk is deze foto een simpele versie van dit, een echte stappenmotor heeft veel meer alternerende polen aan de rotor en veel meer spoelen aan de stator.



DRV8825

Hier zie je 2 duidelijke schema's die het blokschema en de externe componenten op de driver laten zien. Als we naar het blokschema kijken zien we ingangen die we zelf moeten aansluiten (zoals de sleep, modes en step pin) en welke pull up en pull down weerstanden hebben. We zien dat de data gaat naar een controller die dan bepaald wanneer en welke soort signalen moeten gestuurd worden naar de motor drivers. De soort signalen kunnen we zien verder in het verslag bij metingen. Deze driver wordt aangestuurd door pulsen te sturen naar de step pin. Met de driver kunnen we de motor dus heel simpel en op veel verschillende manieren aansturen, zoals microstep. Fullstep is simpel daar worden gewoon de pinnen verschillend hoog en laag gemaakt. Maar bij microstep word bij elke puls een volgende stap gezet in een blokvormig AC sinus signaal (duidelijker te zien bij de metingen). Het nut van microstepping is om de stappenmotor zo nauwkeurig mogelijk te laten draaien. Dit is nodig in bijvoorbeeld een 3D printer, een PCB freesmachine of een balancerende

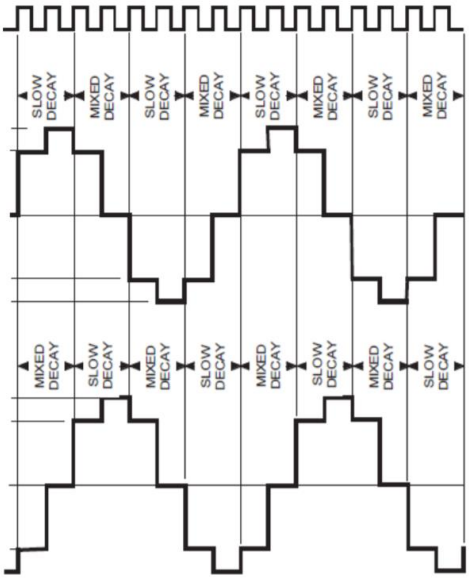




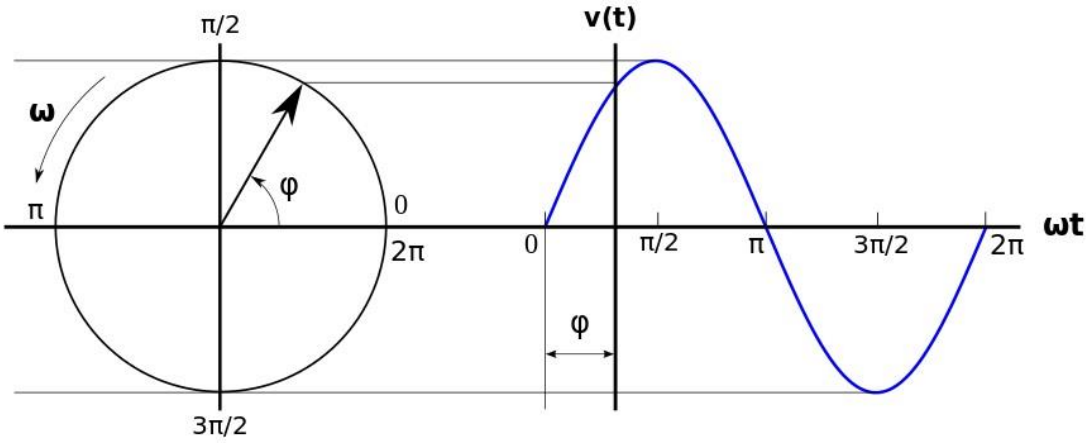
robot.

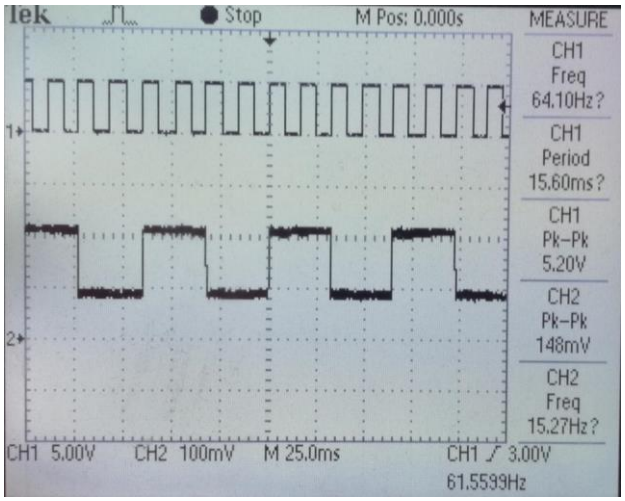
Metingen

Voor alle metingen op CH2 (de faseverschuivingen en stappen) moest ik een stroomprobe gebruiken. De reden hiervoor is omdat de DRV8825 de stroom moduleert voor microstepping. De probe die ik gebruik (zie foto, Pico TA018) heeft 2 modes: 1mV/10mA met max 20A en 1mV/100mA met max 60A. Dus ik gebruik de eerste mode en dat betekent dat de spanning van CH2 de stroom voorstelt (bvb 150mV = 1,5A). Maar je moet wel de gemeten spanning door 2 delen (dus 150mV = 0,75A) want net zoals bij een AC signaal (zoals op de onderste foto) is het nulpunt in de helft en omdat we Pk-Pk/Peak to Peak meten, meten we de positieve waarden en de negatieve waarden en tellen de absolute waarde op. Daarom moeten we het door 2 delen.

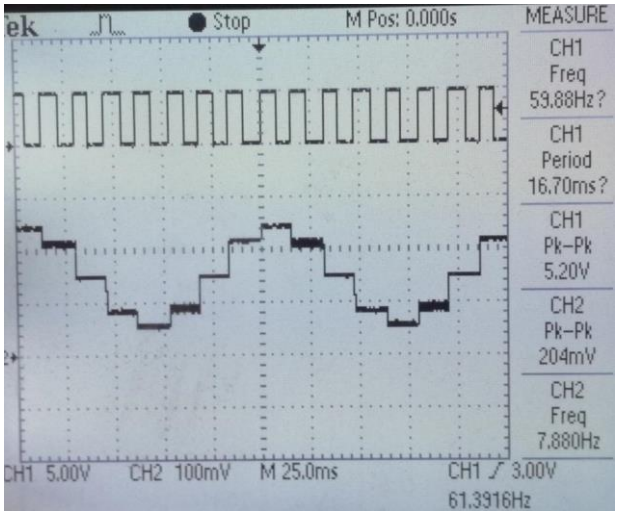


Op de scoopbeelden zal het ook lijken dat we maar 1 signaal naar de motor sturen. Maar eigenlijk stuur ik 2 signalen met een faseverschuiving van 90° , zoals we zien op de foto.

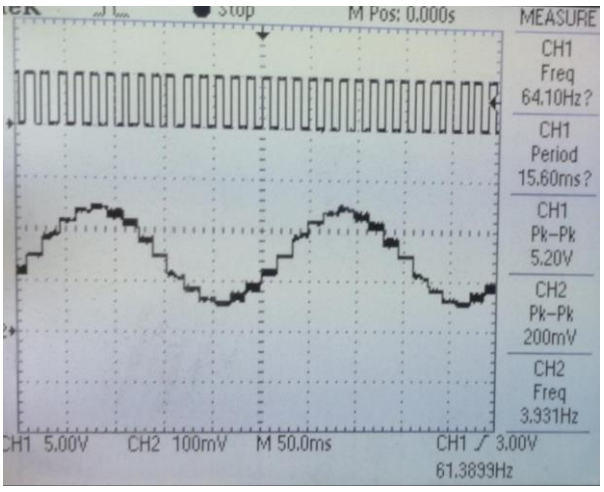




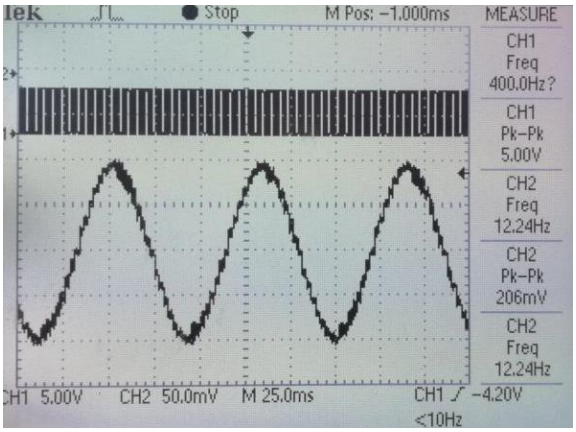
Dit is Full Step dus alle MODE pinnen zijn aan de GND verbonden (zie tabel bij schema). Door de pulsen zichtbaar op CH1 naar de driver te sturen, krijgen we de stappen zichtbaar op CH2. We kunnen ook zien dat de frequentie van CH1 gelijk is aan de frequentie van CH2 gedeeld door 4 (64Hz / 4 = 16Hz). Normaal wordt dit signaal 2 keer naar de motor gestuurd, maar ik kan er maar 1 tegelijkertijd meten. Het is ook duidelijk hoe elke stap start bij een rising edge van CH1. We zien ook de



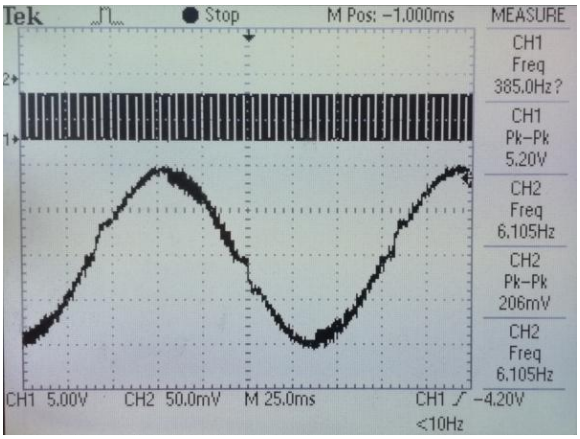
Dit is Half Step dus alle MODE pinnen zijn hetzelfde behalve MODE0, die is hoog (zie tabel bij schema). Hier zien we duidelijk dat het signaal een soort blokvormige sinusgolf wordt. Het is ook duidelijk hoe bij elke rising edge van CH1 een verandering in CH2 komt. Het is ook zo dat de frequentie van CH1 gelijk is aan de frequentie van CH2 gedeeld door 8 (60Hz / 8 = 7.5Hz). We zien ook de stroom door de motor weer met CH2 Pk-Pk.



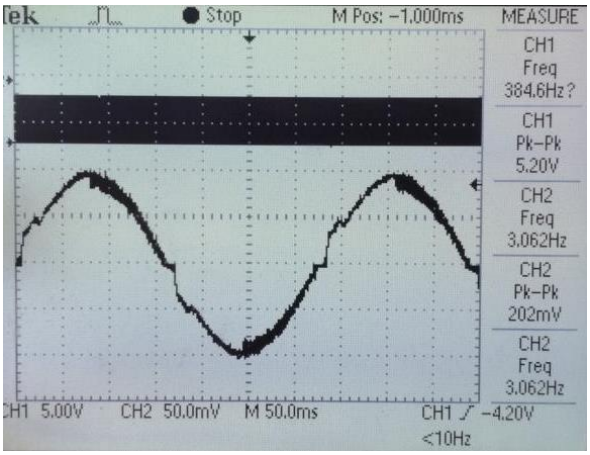
Dit is 1/4 Step dus de MODE pinnen zijn aangesloten in een 010 configuratie (zie tabel bij schema). We zien duidelijk dat het nog steeds een blokvormig sinus signaal is maar deze keer met kleinere blokken. Het is ook zo dat de frequentie van CH1 gelijk is aan de frequentie van CH2 gedeeld door 16 (64Hz / 16 = 4Hz). We zien ook de stroom door de motor weer met CH2 Pk-Pk.



Dit is 1/8 Step dus de MODE pinnen zijn aangesloten in een 110 configuratie (zie tabel bij schema). Hier is het duidelijk dat er nog steeds een blokvormig sinus signaal is, maar deze keer met nog zeer kleine blokken. We zien ook de stroom door de motor met de CH2 Pk-Pk meting. Het is ook zo dat de frequentie van CH1 gelijk is aan de frequentie van CH2 gedeeld door 32 (400Hz / 32 = 12.5Hz).



Dit is 1/16 Step dus de MODE pinnen zijn aangesloten in een 001 configuratie (zie tabel bij schema). We zien deze keer gewoon niet dat het nog blokvormig is want de blokvormige delen zijn gewoon te klein. We zien ook de stroom door de motor weer met de CH2 Pk-Pk meting. Het is ook zo dat de frequentie van CH1 gelijk is aan de frequentie van CH2 gedeeld door 64 (385Hz / 64 = 6.015Hz).



Dit is 1/32 Step dus de MODE pinnen zijn aangesloten in een 101, 011 of 111 configuratie (zie tabel bij schema). We zien ook hier niet dat het nog blokvormig is omdat de blokvormige delen gewoon te klein zijn. We zien ook de stroom door de motor weer met de CH2 Pk-Pk meting. De frequentie van CH1 is gelijk aan de frequentie van CH2 gedeeld door 128 (385Hz / 128 = 3.007Hz).

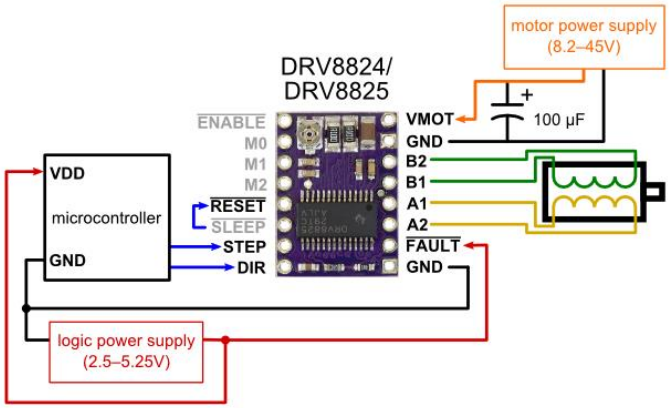
Software

```
unsigned int ADC0;        // Variable waar de ADC waarde in
wordt opgeslagen

void setup()
{
  DDRF |= 0b00110000;     // Maak PF5 en PF4 outputs en
verander de andere pinnen niet (door het masker)

  Serial.begin(9600);     // Bautrate van de seriële monitor
zetten op 9600

  unsigned int READ_ADC_INT_CHANNEL( unsigned char channel );
// Prototype functie voor ADC naar een INT
}

void loop()
{
  ADC0 = READ_ADC_INT_CHANNEL(0);      // Lees de potmeter
waarde op PF0 in en slaag het op in de variable ADC0
  ADC0 = map(ADC0, 0, 1023, 1, 1005);  // De minimum waarde
van de ADC meting is i.p.v. 0 nu 1 en de maximum waarde is 105
i.p.v. 1023

  Serial.println(ADC0);                // De potmeter waarde en
de stand van de schakelaar op de seriële monitor weergeven
  Serial.print("\t");
  Serial.println(digitalRead(22));

  if(ADC0 <= 1000)                      // Laat de motor alleen
draaien als er al een klein beetje aan de potmeter is gedraaid
  {
    PORTF ^= 0b00100000;               // Maak pulsen waarvan
je de frequentie van kan aanpassen met de potmeter
    delay(ADC0);                       // Er kan ook een for
loop gebruikt worden om te delay te regelen maar dit is
simpeler en korter
  }

  if(PINF & 0b00000010)                // Als PF1 hoog is/als
de schakelaar is overgehaald
  {
    PORTF |= 0b00010000;               // Maak alleen PF4 hoog
  }

  else
  {
    PORTF &= 0b11101111;               // Maak alleen PF4 laag
  }
}
```

```
unsigned int READ_ADC_INT_CHANNEL( unsigned char channel )
{
  DDRF &= ~(1<<channel); // config selected channel as input

  // Right adjust + choice of channel
  ADMUX = 0b01000000 +  channel;

  // Activate ADC - Stop conversion - prescaler 128 (for ADC
50K<..<200k) (16Mhz / 128 = 125000)
  ADCSRA = 0b10000111;

  // start conversion and wait for completion
  ADCSRA |= (1<<ADSC);
  while (ADCSRA & (1<<ADSC));

  // return A/D conversion result
  unsigned char ADCLBuff = ADCL; // first read ADCL!!
  return ( ((unsigned int) ADCH ) << 8 ) + ADCLBuff;
}
```

Besluit

Door dit labo heb ik uit eigen ervaring en testen geleerd hoe de DRV8825 driver werkt. Voorbeelden van dingen die ik geleerd heb zijn : de regelbare weerstand op de driver zorgt voor een stroomlimiet door de driver. In mijn schema heb ik niets gedaan met de FAULT pin, maar voor beveiliging ga ik die met een weerstand verbinden aan de SLEEP pin. Zodat als er een fout gebeurt, de driver in sleep modus gaat. Ten slotte zou ik ook nog condensatoren hebben moeten gebruiken aan de voeding voor de mot or om de spanning en stroompieken te filteren. Zoals op de foto hieronder.



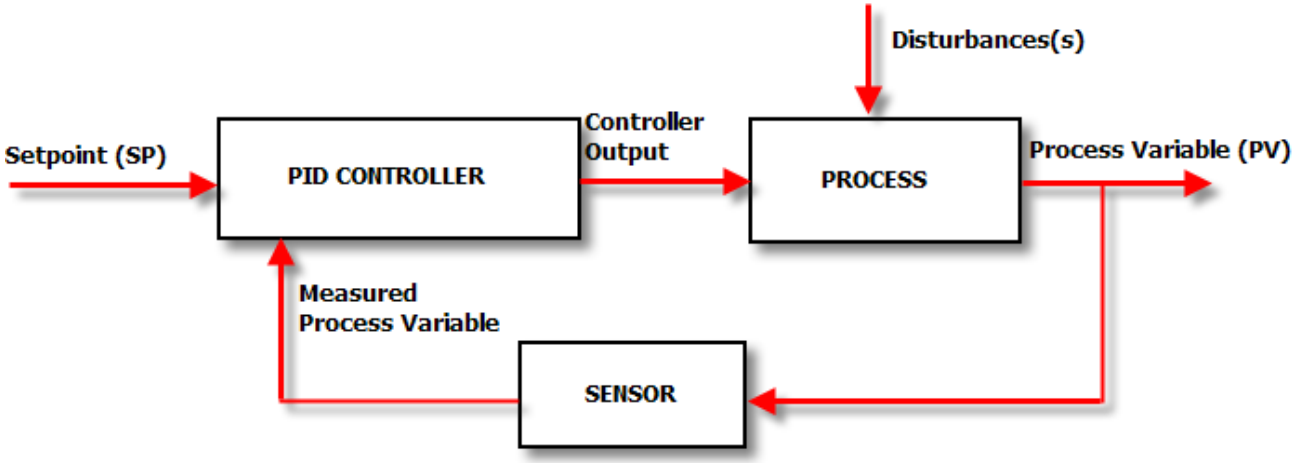
Labo PID regelaar en orde proces

Doelstelling

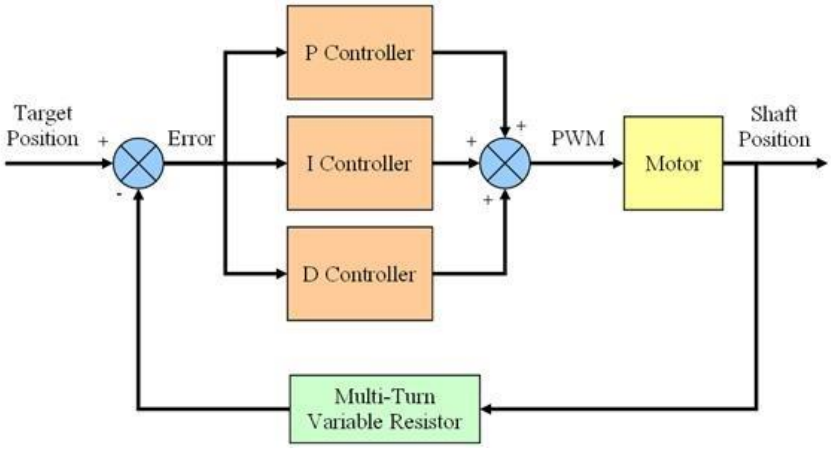
Het doel van dit labo is dat ik PID begrijp en het kan toepassen in de praktijk. Ik moest ook uitzoeken welk orde process mijn stappenmotor heeft bij een verandering van zijn toerental.

Theorie

PID staat voor **P**roportioneel, **I**ntegrerend en **D**ifferentiërend en het is een regelaar/regeltechniek. Hieronder zien we een blokschema over de werking van PID, het is duidelijk een soort feedback systeem. Dus we hebben een “setpoint” of de waarde die we willen bereiken en het “process variable” of de werkelijke waarde. Dus we sturen de waarde die we willen bereiken naar een controller (Ardiuno voor mij) en die gaat dan het process regelen zodat PV gelijk wordt aan SP, dan zal de sensor de werkelijke waarde meten en terugsturen naar de controller zodat de controller het process kan aanpassen om SP meer gelijk aan PV te krijgen. Het verschil van PV en SP noemen we de error en als de error 0 is zal de controller stoppen met regelen tot er terug een error is door bvb storingen, dan zal die terug beginnen bij het begin.



Aan dit ander blokschema zien we dat P, I en D een aparte functie en nut hebben in een opstelling. Om het simpel uit te leggen, P zal focussen op wat er op dat moment aan het gebeuren is en de controller zo laten reageren, het is de simpelste van de 3 en bepaalt eigenlijk hoe hard de controller zal reageren op de error. I zal focussen op het verleden of wat al gebeurd is, het integreert de error, als error negatief is dan wordt de totale waarde kleiner, daardoor wordt de reactiesnelheid beperkt en de stabiliteit beïnvloed. Ten laatste D, die zal focussen op de toekomst of wat nog moet komen door de snelheid van de verandering in de error waarden te weten en daarmee de controller laten weten dat de error snel 0 zal worden om overshooting te vermijden.



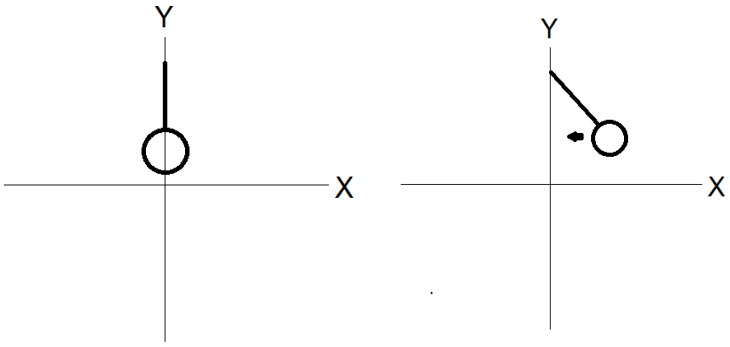
De formules voor PID zijn:

- P -> $\mathrm{Kp} \,.\, (\mathrm{Error})$
- I -> $\mathrm{Ki} \,.\, (\mathrm{Error}) \,.\, \mathrm{tijd} + \mathrm{OudeErrors}$
- D -> $\mathrm{Kd} \,.\, \frac{\mathrm{d(Error)}}{\mathrm{dtijd}}$

Met deze formules zie je ook duidelijk welk effect P, I en D hebben, P bepaalt hoe hard de controller reageert, I blijft de errors optellen per tijdseenheid om te finetunen en D ziet naar de verandering van de error per tijdseenheid.

We moeten ook niet altijd P, I en D tesamen gebruiken. Er zijn verschillende combinaties die goed werken bij bepaalde processen. Je kan alleen P gebruiken of PI of PD of natuurlijk PID.

Theoretische toepassing



Als voorbeeld gebruik ik een bal aan een touw dat je bovenaan vast hebt. Zoals wordt getoond op de foto. Als je de bal zou los laten dan zou die natuurlijk beginnen slingeren van links naar rechts, of oscilleren. Maar wat als we willen dat die perfect op de Y-as zou moeten stoppen. Dat kunnen we bereiken als we PID theoretisch zouden toepassen.

- Als we alleen P zouden gebruiken

Dit is wat er in het echt zou gebeuren, de bal zal terug verplaatsen maar niet direct kunnen stoppen op de Y-as waardoor er overshoot zal zijn, de bal te ver slingert en dan weer terug komt, tot de bal uiteindelijk stopt aan de Y-as. Maar dat is niet wat we willen en veel te traag. Het dus alleen voor heel ruwe functies die niet heel precies zijn

- Als we alleen I zouden gebruiken

Dan zou het heel erg traag dalen want het kijkt naar alle oude errors en voegt ze bij elkaar maar zal dus niet heel hard reageren op de error. De bal zou dan wel stoppen bij de Y-as maar het zal er een lange tijd over doen omdat I eigenlijk meer wordt gebruikt om heel kleine errors eruit te halen, zoals wanneer de bal een klein beetje afstand heeft van de Y-as waardoor P er niets aan doet.

- Als we alleen D zouden gebruiken

Het heeft niet echt nut om D alleen te gebruiken want net zoals als bij I is dit enkel om scherp te stellen. Maar i.p.v. kleine errors weg te halen zal het zorgen dat de bal weet hoe snel die naar de Y-as gaat en ervoor zorgen dat die zal beginnen afremmen als die dichtbij komt.

- Als we dan P I en D tezamen zouden gebruiken

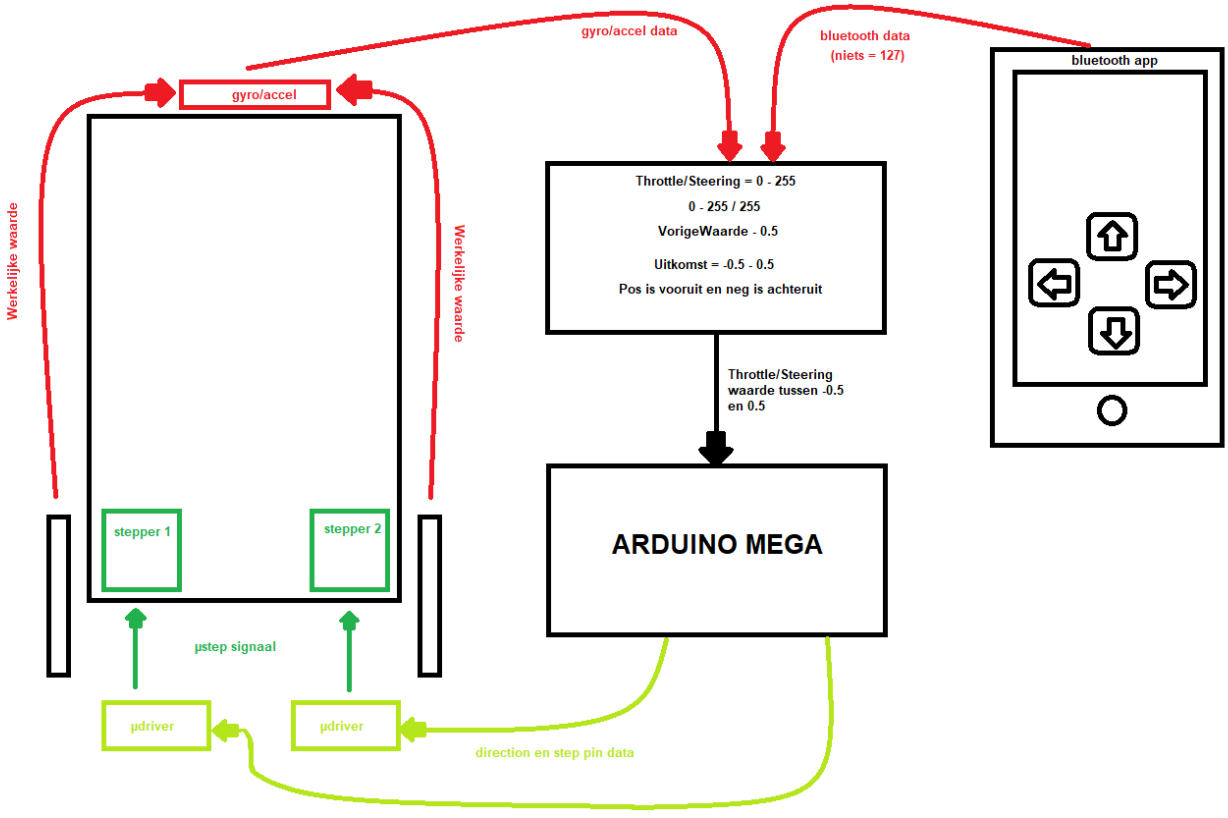
Dan zouden we krijgen wat we willen, we laten de bal los en die zal perfect op de Y-as stoppen (als de variables Kp Ki en Kd) goed afgesteld zijn natuurlijk.

Toepassing bij balancerende robot

Aan het vorige voorbeeld kunnen we duidelijk zien waarom we PID nodig gaan hebben voor een balancerende robot. Ik zal het nut van PID voor de robot uitleggen ook al is er niet veel verschil met het vorige voorbeeld. Het enige verschil is dat (volgens de oude versie van deze GIP) er 2 regelaars werden gebruikt, een PI regelaar voor de snelheid van de wielen en een PD regelaar voor de stabiliteit.

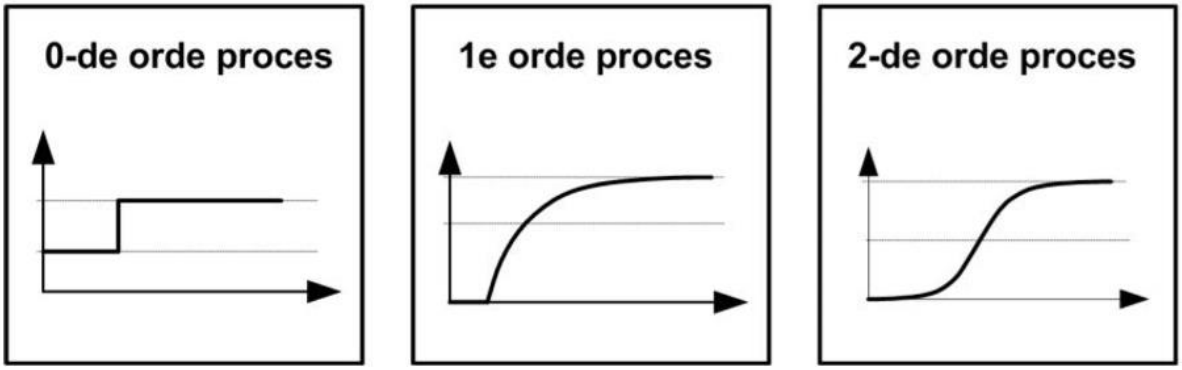
Ik moet niet meer uitleggen wat de P doet in beide regelaars want dat is vrij duidelijk. Maar de I en D zijn misschien niet zo duidelijk. De I bij de snelheid regeling is nodig want de Arduino moet de snelheid van de motor, heel snel en heel precies kunnen aanpassen. Dus door de I toe te passen zal die heel precies kunnen zijn. De D bij het stabiel houden moet gebruikt worden want de Arduino moet zien aankomen wanneer de robot bijna stabiel staat zodat de motoren de robot stabiel kunnen houden, en zo min mogelijk overshoot en oscillaties heeft.

Hier is ook dan een blokschema van de regelaar.

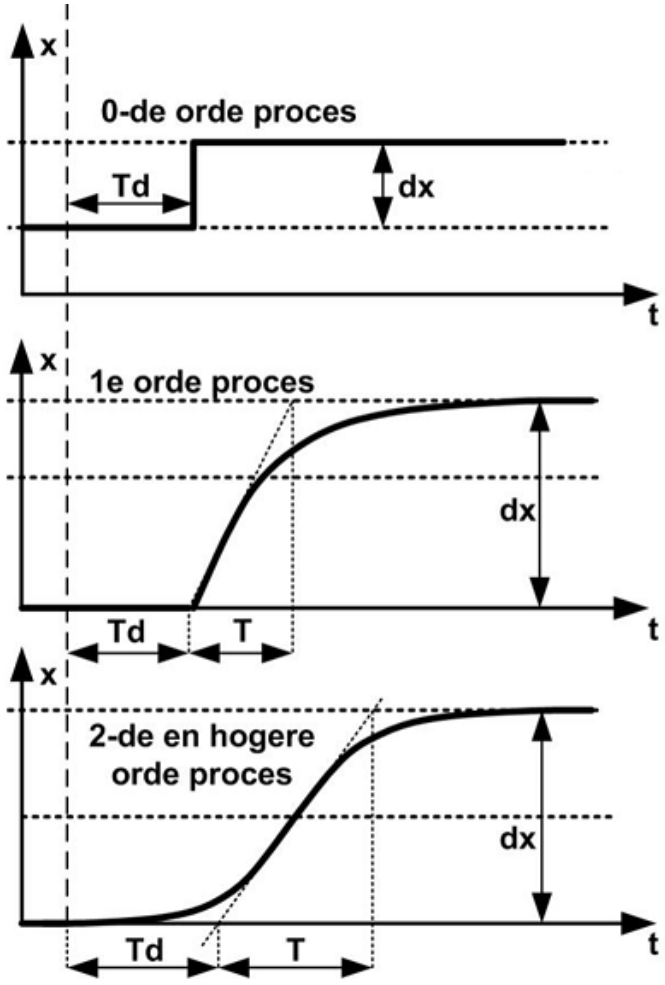


Orde process

Een orde process is simpelweg de manier hoe het process zal reageren op verandering door de controller. Dus bij mijn robot is het hoe de stappenmotoren zouden reageren op een verandering van hun toerental. Hieronder zie je de 3 verschillende orde processen.



Er is iets waar wel rekening mee moet gehouden worden : de dode tijd of Td. Td begint wanneer de controller een nieuw signaal stuurt naar het process en het eindigt wanneer PV begint met veranderen door dat signaal. Het andere waar rekening mee moet gehouden worden is : T. T begint wanneer Td eindigt en eindigt wanneer PV klaar is met veranderen. Die 2 parameters bepalen hoe gemakkelijk het is om het process aan de passen.



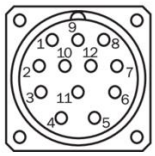
Vuistregel :

- T/Td = >10, goed regelbaar
- T/Td = 6, matig regelbaar
- T/Td = <3, moeilijk regelbaar

Orde process bepalen

Ik kreeg de taak om uit te zoeken hoe de stappenmotoren die ik gebruik, zouden reageren op snelheidsveranderingen. Dat moest ik dan doen met een Rotary encoder (de srs50-hxa0-k21). Dat kon ik doen door de rotor van de motor aan de rotor van de encoder te verbinden en de motor te laten draaien. Als je de encoder voedt met +12V en je draait aan de rotor dan krijg je een sinus en cosinus signaal aan de witte en roze kabel waarvan de frequentie recht evenredig is aan de snelheid van de rotor.

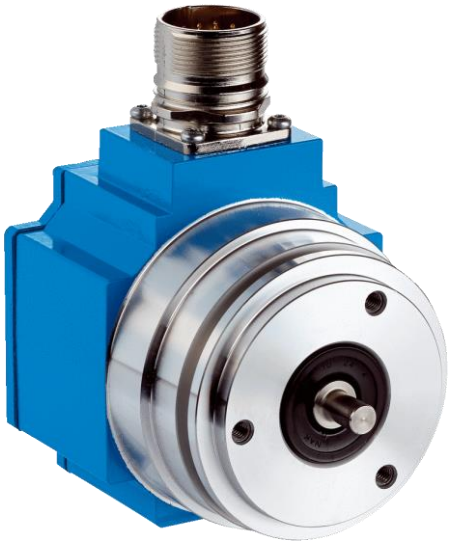
PIN	Signal	Colour of Wires	Explanation
1	REFCOS	black	Process data channel
2	Data +	grey or yellow	RS-485-parameter channel
3	N. C.	–	N. C.
4	N. C.	–	N. C.
5	SIN	white	Process data channel
6	REFSIN	brown	Process data channel
7	Data –	green or purple	RS-485-parameter channel
8	COS	pink	Process data channel
9	N. C.	–	N. C.
10	GND	blue	Ground connection
11	N. C.	–	N. C.
12	Us	red	7 ... 12 V Supply voltage



View of the plug-in face

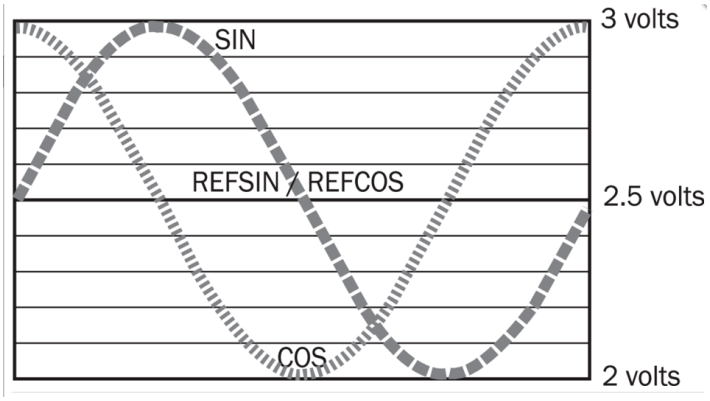
Screen connection on connector housing

N. C. = Not connected



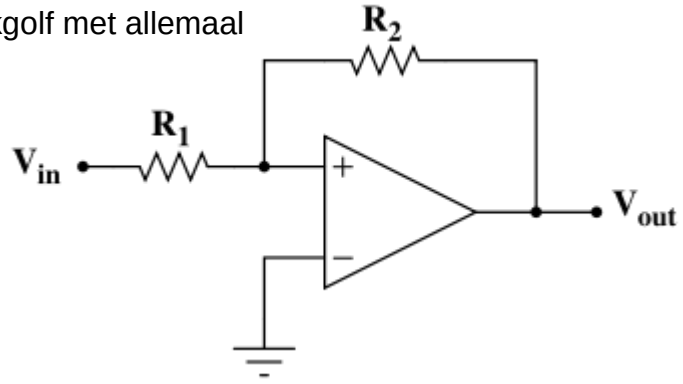
Er was wel een probleem. Het was onmogelijk om het via een oscilloscoop te doen want het signaal veranderde te snel, dus ik moest het doen via een logic analyzer. Maar daarop volgde nog een probleem. De logic analyzer kon het sinusvormig signaal niet meten. Dus ik moest met een schmitt trigger er een blokgolf van maken.

Omdat het signaal verandert tussen 2 en 3 volt, zoals op de foto zichtbaar is, besloot ik de hysteresis op 2.25 en 2.75V te maken.

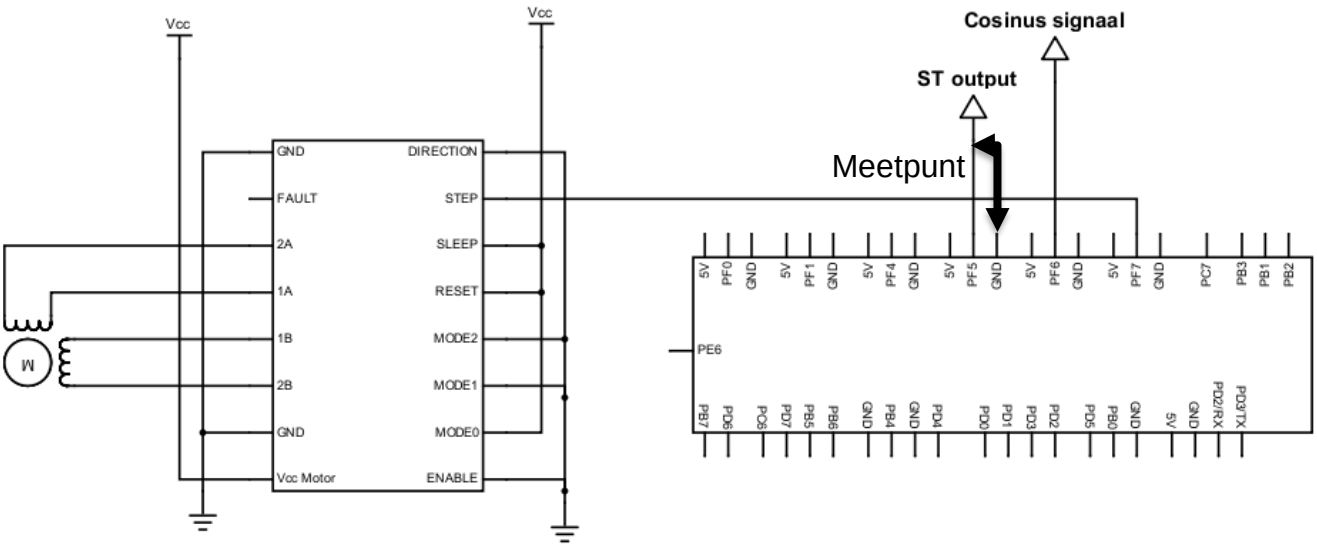


Eerst ging ik proberen een ST te maken met een opamp in deze configuratie. Maar ik had nog nooit geleerd hoe ik de weerstanden moest berekenen, dus probeerde ik het te leren via het internet. Maar dat lukte niet want elke bron zei iets anders. Dus probeerde ik calculators die het werk voor jou doen te gebruiken, maar ze gaven allemaal andere waardes die ook niet werkten. Dus op het einde heb ik dan een werkende schmitt trigger gemaakt door die te programmeren in Arduino.

Maar dat heeft ook niet geholpen want het probleem nu is dat er te veel slip was door de fitting tussen de rotor van de motor en encoder. Dus ook al liet ik de motor op een constante snelheid draaien, kreeg ik een blokgolf met allemaal verschillende frequenties.



Schema



Software

```
unsigned char Ucos = 0; // Variable die de spanning van het cosinus signaal
weergeeft

#define ST_Input 6
#define ST_Output 20

void setup()
{
  DDRF &= 0b10111111; // Maak PF6 laag (input)
  DDRF |= 0b10100000; // Maak PF7 en PF5 hoog (output)
  Serial.begin(9600);

  unsigned char Schmitt_Trigger(unsigned char LowH, unsigned char HighH);
// LowH = lage hysteresis en HighH = hoge hysteresis
  unsigned int READ_ADC_INT_CHANNEL( unsigned char channel );   //
Prototype functie voor ADC naar een INT
}

void loop()
{
  Ucos = map(READ_ADC_INT_CHANNEL(ST_Input), 0, 1023, 0, 100);  // Met een
waarde van 0 tot 100 kan ik gemakkelijk de hysteresis berekenen met procent

  Schmitt_Trigger(45,55); // Zet de hysteresis op 45% en 55% van 5V

  PORTF |= 0b00100000;  // Maak alleen PF5 hoog
  _delay_ms(1);
  PORTF &= 0b11011111;  // Maak alleen PF5 laag
  _delay_ms(1);

  Serial.print(Ucos);
```

```
  Serial.print(" ");
  Serial.print(digitalRead(ST_Output));
}

unsigned char Schmitt_Trigger(unsigned char LowH, unsigned char HighH)
{
  if(Ucos <= LowH)  // 45% van 5V is 2.25 en dat is de lage hysteresis die
ik wil
  {
    PORTF &= 0b01111111;   // Als de spanning van de RE gelijk ligt of
onder 2.25V ligt dan word de output van de ST laag
  }

  if(Ucos >= HighH)  // 55% van 5V is 2.75 en dat is de lage hysteresis die
ik wil
  {
    PORTF |= 0b10000000;  // Als de spanning van de RE gelijk ligt of boven
2.75V ligt dan word de output van de ST hoog
  }
}

unsigned int READ_ADC_INT_CHANNEL( unsigned char channel )
{
  DDRF &= ~(1<<channel); // config selected channel as input

  // Right adjust + choice of channel
  ADMUX = 0b01000000 +  channel;

  // Activate ADC - Stop conversion - prescaler 128 (for ADC 50K<..<200k)
(16Mhz / 128 = 125000)
  ADCSRA = 0b10000111;

  // start conversion and wait for completion
  ADCSRA |= (1<<ADSC);
  while (ADCSRA & (1<<ADSC));

  // return A/D conversion result
  unsigned char ADCLBuff = ADCL; // first read ADCL!!
  return ( ((unsigned int) ADCH ) << 8 ) + ADCLBuff;
}
```

Besluit

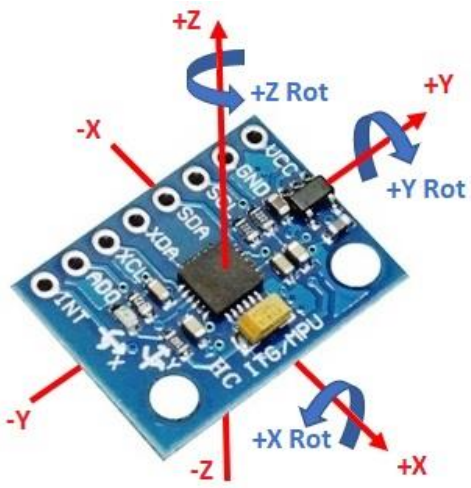
Door dit labo begrijp ik nu hoe PID werkt en hoe ik het moet toepassen. Het is wel spijtig dat ik niet heb kunnen uitzoeken welke soort orde process de motor heeft bij verandering van snelheid, zelfs na een goede fitting te vinden die geen slip had (Ducttape) had ik nog steeds geen goede metingen. Dus ik moest het op een andere manier aanpakken. Ik ging het doen met een toerentalteller schijf (foto) en een IR sensor. (Dit is op het einde niet door kunnen gaan door de lock down).

Labo MPU-6050 gyrosensor en accelerometer

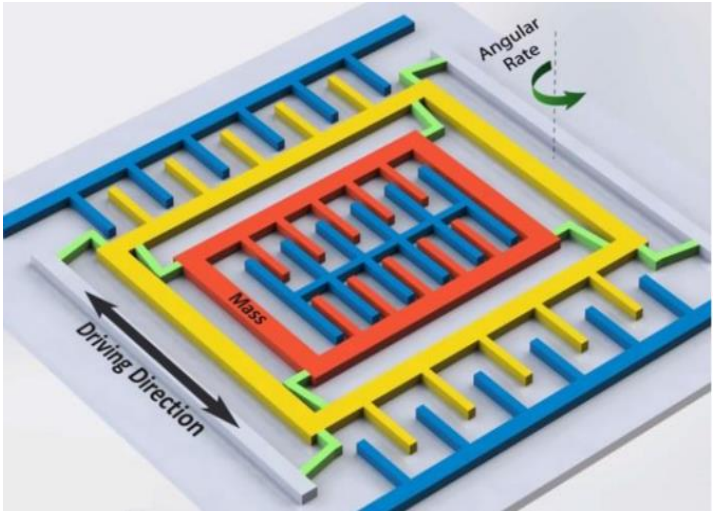
Doelstelling

Het doel van dit labo is om te begrijpen hoe een gyrosensor en accelerometer werken en de waarden van de sensoren (gyrosensor, accelerometer en temperatuursensor) kunnen uitlezen via I2C en die weer te geven als grafiek.

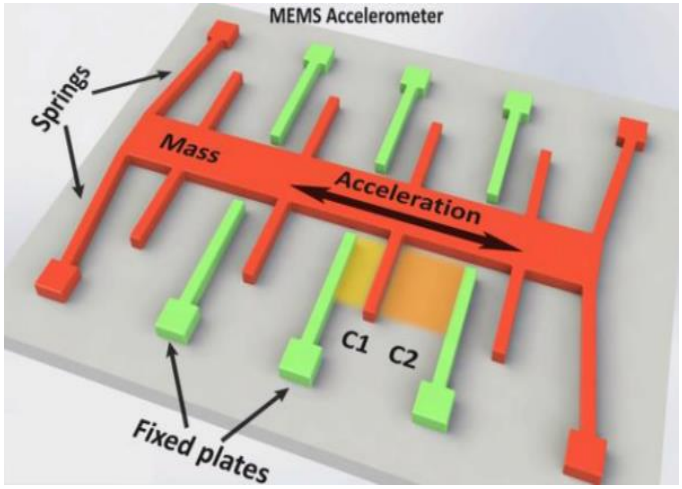
Theorie



Hier zien we de 3 assen waarop de gyrosensor en accelerometer zullen meten. De blauwe pijlen stellen de meetpunten van de gyrosensor voor en de rode pijlen stellen de meetpunten van de accelerometer voor. De gyrosensor meet draaiingen t.o.v. de zwaartekracht. De accelerometer meet lineaire bewegingsverschillen. Dit zal later wel duidelijker worden.

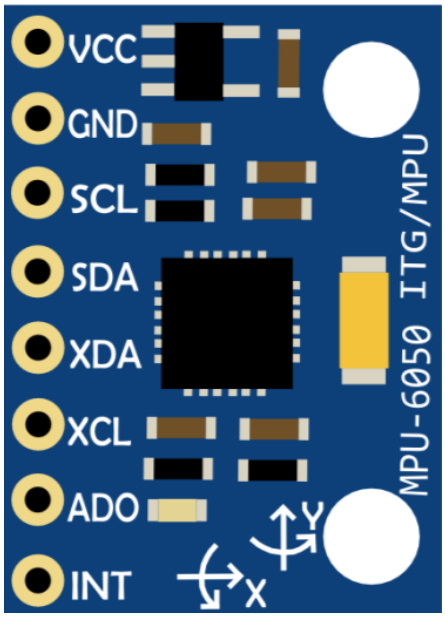


Dit is een foto die de werking van een gyrosensor laat zien. Het gele deel is een constant oscillerende kern. Wanneer er dan een draaibeweging voorkomt dan verschuift de binnenste (oranje) plaat. Er wordt constant de capaciteit tussen de 2 binnenste (oranje en blauwe) platen gemeten en die zal veranderen als er een draaibeweging voorkomt.



Dit is een foto die de werking van een accelerometer laat zien. De accelerometer bestaat uit (groene) vaste plaatjes en (oranje) bewegende plaatsjes. Er wordt ook constant de capaciteit tussen de 2 plaatjes gemeten en als er dan een beweging voorkomt dan zal de capaciteit veranderen.

MPU6050



Hier zie je de pin lay-out van MPU6050:

- Vcc is waar je +5V moet aansluiten.
- GND is de ground.
- SCL is de “Serial Clock” voor I2C, hier komt het kloksignaal van de master binnen.
- SDA is de “Serial Data” voor I2C, hier komt de data voor de master er uit.
- XDA is “Auxiliary Serial Data” en wordt gebruikt om met andere I2C modules te communiceren.
- XCL is “Auxiliary Serial Clock” idem als XDA.
- AD0 is voor het slave address aan te passen als je 2 MPU6050s wilt aansluiten aan 1 master.
- INT is de interrupt pin wanneer er nieuwe leesbare data is voor de master

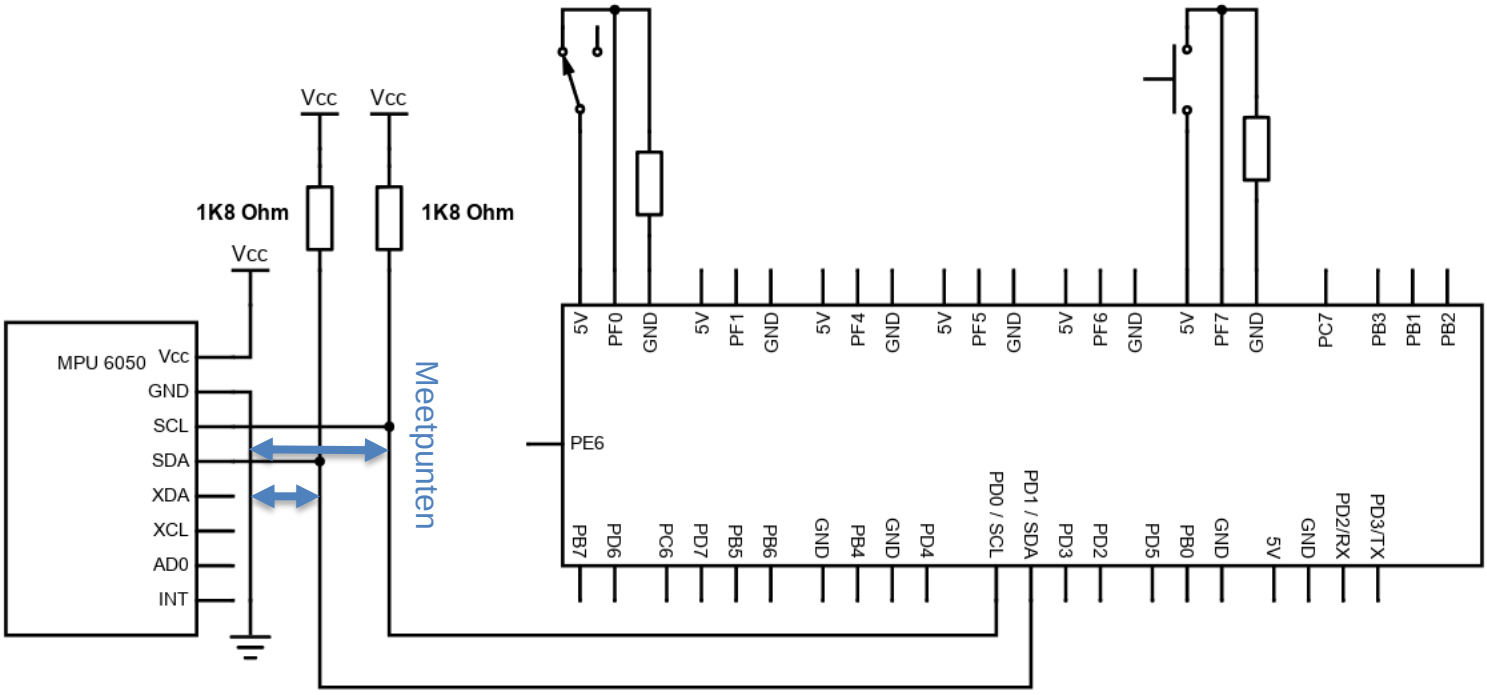
In mijn situatie ga ik alleen de eerste 4 pinnen (Vcc, GND, SCL en SDA) gebruiken.

Het slave address van de MPU6050 is standaard 0x68 = 0110 100X (0x69 als AD0 hoog is), de laatste bit is dan de R/W bit. Dus als de MPU6050 in “read mode” moet zijn is het address 0xD0 en als de MPU6050 in “write mode” moet zijn zal het address 0xD1 zijn. (Slave address is te vinden in de datasheet)

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]							

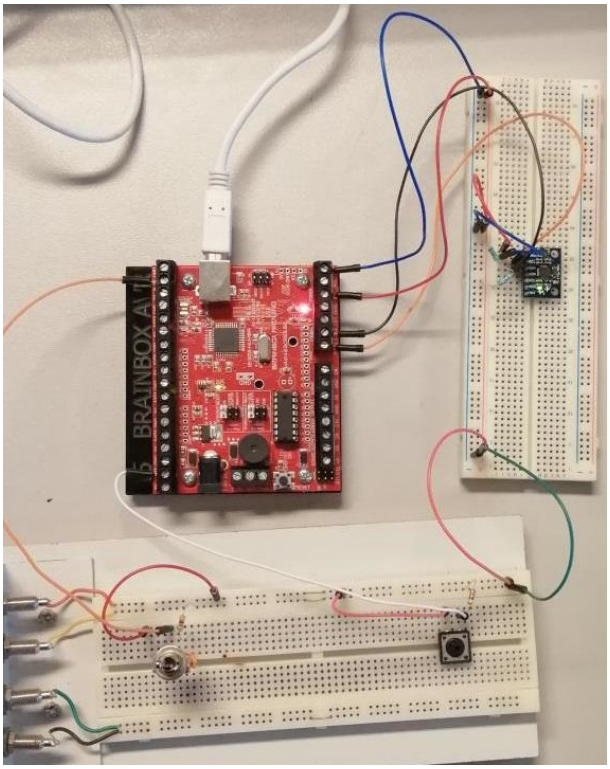
Hier is een foto van het index register van de MPU6050. Ik gebruik elk register behalve het bovenste want ik gebruik de INT pin niet. Elke as bij een sensor heeft 2 aparte indexen die een byte groot zijn, de 8 MSB (H) en de 8 LSB (L), dus de totale index voor 1 as is 16 bits lang.

Schema

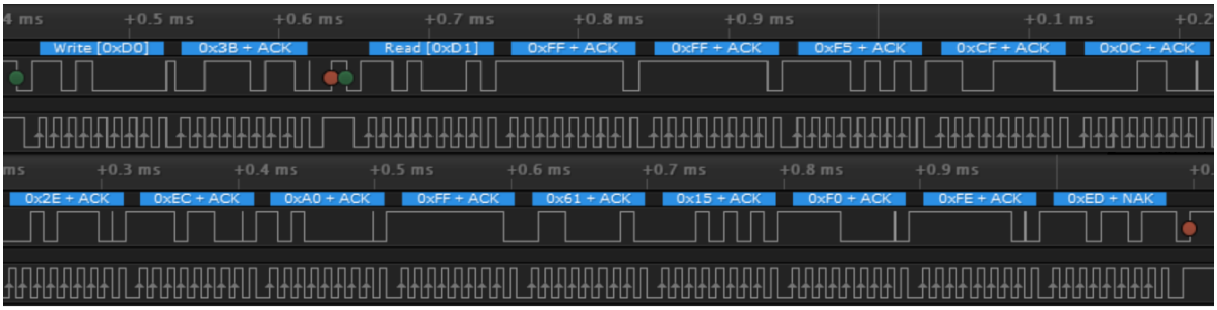


Beschrijving

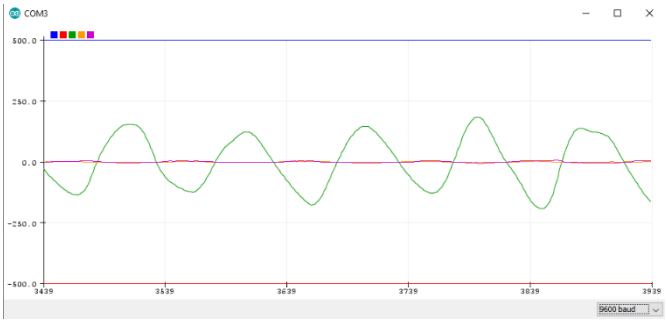
Ik gebruikt de 3 sensoren in dit labo. Ik gebruik de accelerometer om hoeken te meten (zoals een waterpas), de gyrosensor meet de hoeksnelheid en de temperatuursensor meet de kamertemperatuur. Er zijn ook 2 schakelaars, een drukknop en een bistabiele schakelaar. De drukknop pauzeert de meting als je er op drukt en de schakelaar bepaalt wanneer de metingen beginnen bij de start van het programma.

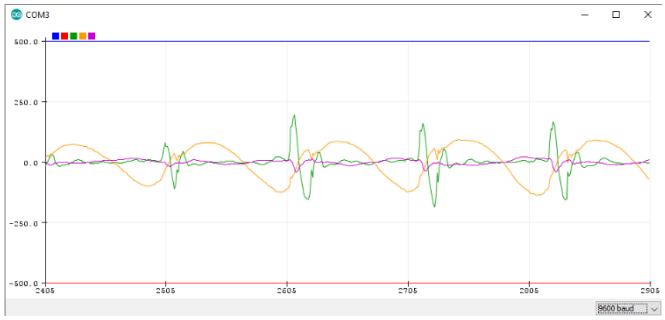


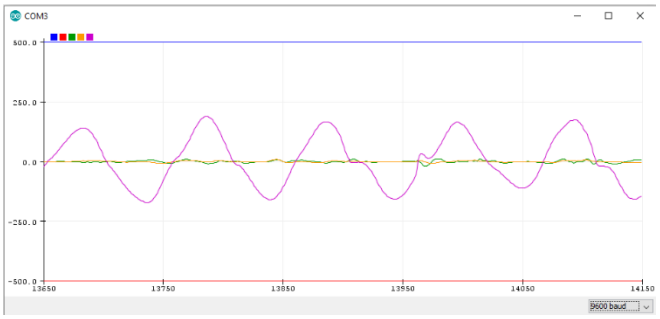
Metingen



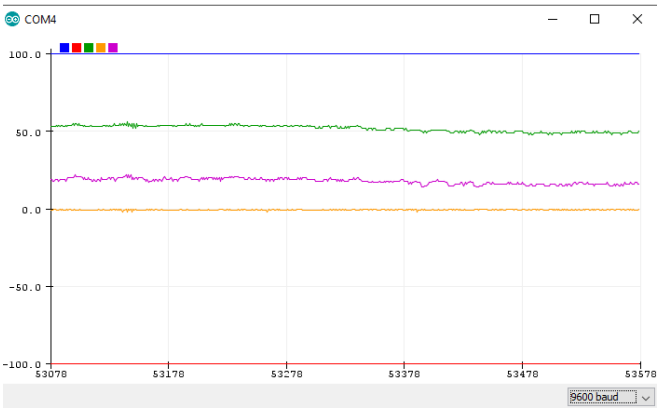
Hier is een foto van de 14 bytes die worden gestuurd naar de brainbox. Eerst wordt 0xD0 en 0x3B gestuurd, dat is het write address en de eerste index van de MPU en zal elke keer als ik data inlees hetzelfde zijn. Daarna is er een stop (rode en groene bol) en 0xD1, dat is om te veranderen naar read mode en zal ook elke keer hetzelfde zijn. Nu komen de 14 bytes. De eerste 6 bytes zijn van de accelerometer (2 per as), de 2 hierna zijn voor de temp. sensor en de laatste 6 zijn die van de gyro sensor (2 per as).



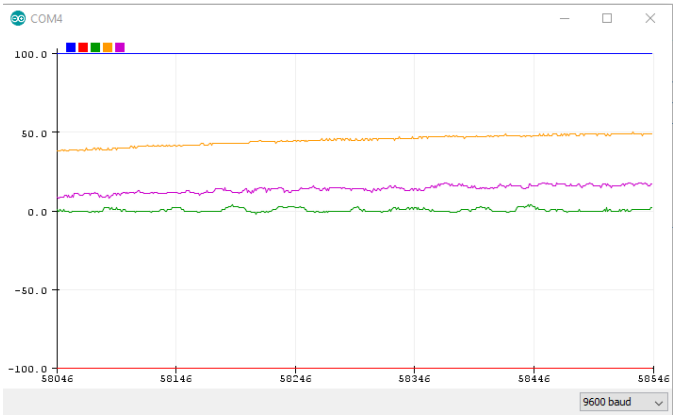




Hier zijn ook nog metingen van de seriële plotter van arduino. De eerste 3 zijn de X Y en Z as metingen van de accelerometer en de laatste 2 zijn de X en Y as van de gyrosensor (Z as maakt niet uit met mijn programma). Bij de eerste 3 doe ik hetzelfde, ik beweeg het breadboard op en neer op de as van de meting. Dus eerst gaat het naar boven (dan stijgt de gemeten waarde) en daarna terug naar beneden (de meting daalt en wordt negatief want ik ga de tegenovergestelde richting op).



Bij deze 2 metingen (X is groene en Y is oranje) heb ik het breadboard gekanteld tot +-50° en het zo even laten staan. Dit zou bvb gebruikt kunnen worden om te meten hoeveel graden een object is gekanteld.



Wat niet perfect is aan deze metingen is dat ik de accelerometer gebruik voor het aantal graden en de gyrosensor voor versnellingen te meten. Mijn reden hiervoor zijn de formules die ik heb gebruikt. Er zijn formules nodig om de rauwe waardes die van de MPU6050 komen om te vormen naar versnelling. De juiste formules vinden was helemaal niet simpel en koste veel tijd. De enige werkende formules die ik vond zijn degene die ik nu gebruik. Ook al is het een onorthodoxe manier om de data uit te lezen, het werkt nog steeds heel goed dus zie ik er geen probleem mee. Het enige probleem dat ik zie is dat de gyrosensor beter zal zijn in hoeken meten dan de accelerometer en vice versa.

Software

```
#include <Wire.h>                                                    // I2C
lib

boolean Hold;                                                        //
Variable om de metingen te laten stoppen en in hold mode gaan
float GyX, GyY, GyZ;                                                 //
Variable om de rauwe accelerometer data in op te slaan
float AcX, AcY, AcZ;                                                 //
Variable om de rauwe gyro sensor data in op te slaan
float Temp;                                                          //
Variable om de rauwe temperatuur sensor data in op te slaan
int GyX_Int, GyY_Int, GyZ_Int;                                       //
Variable om de omgerekende afgeronde accelerometer data in op de slaan
int AcX_Int, AcY_Int, AcZ_Int;                                       //
Variable om de omgerekende afgeronde gyro sensor data in op de slaan
int Temp_Int;                                                        //
Variable om de omgerekende afgeronde temperatuur sensor data in op de slaan
float GyX_Offset, GyY_Offset, GyZ_Offset;                            //
Variable om de offset van de accelerometer data in op de slaan
float AcX_Offset, AcY_Offset, AcZ_Offset;                            //
Variable om de offset van de gyro sensor data in op de slaan

void Read_Data();                                                    //
Function waar de 14 bytes worden ingelezen van de MPU6050

void setup()
{
  while(PINF & 0b10000000)                                           // Zolang
PF7 hoog blijft doe dan niets
  {
    Serial.println("Haal rode schakelaar over om te starten.");      // Print
instructies op de monitor
  }

  Serial.begin(9600);                                                // Baut
rate van de seriele monitor is 9600
```

```
  Wire.begin();                                                    // Start
de I2C connectie

  // Start de MPU6050
  Wire.beginTransmission(0x68);                                    // Start
de communicatie met de MPU-6050 (write)
  Wire.write(0x6B);                                                //
Selecteer de index
  Wire.write(0);                                                   // Stuur
data naar geselcteerde index
  Wire.endTransmission();                                          // Eindig
de transmissie

  // Zet de accelerometer zijn Full Scale Range op 8g (FS_SEL = 2)
  Wire.beginTransmission(0x68);                                    // Start
de communicatie met de MPU-6050 (write)
  Wire.write(0x1C);                                                //
Selecteer de index
  Wire.write(0x10);                                                // Stuur
data naar geselcteerde index
  Wire.endTransmission();                                          // Eindig
de transmissie

  // Zet de gyrosensor zijn Full Scale Range op 500 (FS_SEL = 1)
  Wire.beginTransmission(0x68);                                    // Start
de communicatie met de MPU-6050 (write)
  Wire.write(0x1B);                                                //
Selecteer de index
  Wire.write(0x08);                                                // Stuur
data naar geselcteerde index
  Wire.endTransmission();                                          // Eindig
de transmissie

  // Berekend offset
  for (int Repeat = 0; Repeat < 2000; Repeat ++)                   // Lees
2000 keer de 14 bytes in in
  {
    Read_Data();                                                   // Lees
de 14 bytes in

    GyX_Offset += GyX;                                             // Tel de
gemete waarde van GyX elke keer bij elkaar op / integreer de data
    GyY_Offset += GyY;                                             // Tel de
gemete waarde van GyY elke keer bij elkaar op / integreer de data
    GyZ_Offset += GyZ;                                             // Tel de
gemete waarde van GyZ elke keer bij elkaar op / integreer de data

    AcX_Offset += AcX;                                             // Tel de
gemete waarde van AcX elke keer bij elkaar op / integreer de data
    AcY_Offset += AcY;                                             // Tel de
gemete waarde van AcY elke keer bij elkaar op / integreer de data
    AcZ_Offset += AcZ;                                             // Tel de
gemete waarde van AcZ elke keer bij elkaar op / integreer de data

    Serial.println("BEREKENEN, MPU NIET BEWEGEN!");                // Print
instructies op monitor
  }

  GyX_Offset /= 2000;                                              // Deel
de totale GyX_Offset waarde door 2000 voor het gemiddelde te krijgen
```

```
  GyY_Offset /= 2000;                                                    // Deel
de totale GyY_Offset waarde door 2000 voor het gemiddelde te krijgen
  GyZ_Offset /= 2000;                                                    // Deel
de totale GyZ_Offset waarde door 2000 voor het gemiddelde te krijgen

  AcX_Offset /= 2000;                                                    // Deel
de totale AcX_Offset waarde door 2000 voor het gemiddelde te krijgen
  AcY_Offset /= 2000;                                                    // Deel
de totale AcY_Offset waarde door 2000 voor het gemiddelde te krijgen
  AcZ_Offset /= 2000;                                                    // Deel
de totale AcZ_Offset waarde door 2000 voor het gemiddelde te krijgen
}


void Read_Data()
{
  Wire.beginTransmission(0x68);                                          // Start
de communicatie met de MPU-6050 (write)
  Wire.write(0x3B);                                                      //
Selecteer de index (0x3B is waar de eerste van de 14 bytes in staat)
  Wire.endTransmission();                                                // Eindig
de transmissie
  Wire.requestFrom(0x68, 14);                                            // Vraag
de 14 opvolgende bytes van de MPU6050, 14 want 6 van de gyro, 6 van de
accelero en 1 voor temp (read) (van 0x3B tot 0x48)

  while(Wire.available() < 14); {}                                       //
Wachten tot alle 14 bytes beschikbaar zijn

  // Inlezen
  AcX = -(Wire.read() << 8 | Wire.read());                               // Eerst
worden de 8 LSB ingelezen en daarna de 8 MSB, eerst worden de LSBs
toegevoegt en dan de MSBs die 8 plaatsen naar links zijn geschift
  AcY = -(Wire.read() << 8 | Wire.read());                               // Eerst
worden de 8 LSB ingelezen en daarna de 8 MSB, eerst worden de LSBs
toegevoegt en dan de MSBs die 8 plaatsen naar links zijn geschift
  AcZ = -(Wire.read() << 8 | Wire.read());                               // Eerst
worden de 8 LSB ingelezen en daarna de 8 MSB, eerst worden de LSBs
toegevoegt en dan de MSBs die 8 plaatsen naar links zijn geschift

  Temp = Wire.read() << 8 | Wire.read();                                 // Eerst
worden de 8 LSB ingelezen en daarna de 8 MSB, eerst worden de LSBs
toegevoegt en dan de MSBs die 8 plaatsen naar links zijn geschift

  GyY = -(Wire.read() << 8 | Wire.read());                               // Eerst
worden de 8 LSB ingelezen en daarna de 8 MSB, eerst worden de LSBs
toegevoegt en dan de MSBs die 8 plaatsen naar links zijn geschift
  GyX = Wire.read() << 8 | Wire.read();                                  // Eerst
worden de 8 LSB ingelezen en daarna de 8 MSB, eerst worden de LSBs
toegevoegt en dan de MSBs die 8 plaatsen naar links zijn geschift
  GyZ = Wire.read() << 8 | Wire.read();                                  // Eerst
worden de 8 LSB ingelezen en daarna de 8 MSB, eerst worden de LSBs
toegevoegt en dan de MSBs die 8 plaatsen naar links zijn geschift
}



void loop()
{
  Read_Data();                                                           // Lees
de 14 bytes in
```

```
  // Offset aftrekken
  GyX -= GyX_Offset;                                                    // Trek
de offset af van de gemete waarde voor een meer accurate gemete waarde
  GyY -= GyY_Offset;                                                    // Trek
de offset af van de gemete waarde voor een meer accurate gemete waarde
  GyZ -= GyZ_Offset;                                                    // Trek
de offset af van de gemete waarde voor een meer accurate gemete waarde

  AcX -= AcX_Offset;                                                    // Trek
de offset af van de gemete waarde voor een meer accurate gemete waarde
  AcY -= AcY_Offset;                                                    // Trek
de offset af van de gemete waarde voor een meer accurate gemete waarde
  AcZ -= AcZ_Offset;                                                    // Trek
de offset af van de gemete waarde voor een meer accurate gemete waarde

  // Berekeningen
  GyX /= 65.5;                                                          // Dit
zet de output om naar °/s want als de sensor draait aan 1°/S dan zal de
output waarde 65.5 zijn (datasheet p12)
  GyY /= 65.5;                                                          // Dit
zet de output om naar °/s want als de sensor draait aan 1°/S dan zal de
output waarde 65.5 zijn (datasheet p12)
  GyZ /= 65.5;                                                          // Dit
zet de output om naar °/s want als de sensor draait aan 1°/S dan zal de
output waarde 65.5 zijn (datasheet p12)


  AcX /= 45.51111;                                                      // Dit
zet de output om naar ° want als de sensor 90°zou zijn gekanteld dan zal de
output waarde 4096 zijn(4096/90 = 45.51) (datasheet p13)
  AcY /= 45.51111;                                                      // Dit
zet de output om naar ° want als de sensor 90°zou zijn gekanteld dan zal de
output waarde 4096 zijn(4096/90 = 45.51) (datasheet p13)
  AcZ /= 45.51111;                                                      // Dit
zet de output om naar ° want als de sensor 90°zou zijn gekanteld dan zal de
output waarde 4096 zijn(4096/90 = 45.51) (datasheet p13)

  Temp /= 340;                                                          // Dit
zet de output om naar °C
  Temp += 36.53;                                                        // Dit
zet de output om naar °C

  // Afronden
  AcX_Int  = (int)AcX;                                                  // Zet
AcX om naar een int zodat het wordt afgerond tot een eenheid zonder
kommagetallen
  AcY_Int = (int)AcY;                                                   // Zet
AcY om naar een int zodat het wordt afgerond tot een eenheid zonder
kommagetallen
  AcZ_Int = (int)AcZ;                                                   // Zet
AcZ om naar een int zodat het wordt afgerond tot een eenheid zonder
kommagetallen
  Temp_Int = (int)Temp;                                                 // Zet
Temp om naar een int zodat het wordt afgerond tot een eenheid zonder
kommagetallen
  GyX_Int = (int)GyX;                                                   // Zet
GyX om naar een int zodat het wordt afgerond tot een eenheid zonder
kommagetallen
```

```
  GyY_Int = (int)GyY;                                          // Zet
GyY om naar een int zodat het wordt afgerond tot een eenheid zonder
kommagetallen
  GyZ_Int = (int)GyZ;                                          // Zet
GyZ om naar een int zodat het wordt afgerond tot een eenheid zonder
kommagetallen

  // Data printen
  Serial.print("AcX = ");
  Serial.print(AcX_Int);
  Serial.print("  ");
  Serial.print("AcY = ");
  Serial.print(AcY_Int);
  Serial.print("  ");
  Serial.print("AcZ = ");
  Serial.print(AcZ_Int);
  Serial.print("     ");

  Serial.print("Temp = ");
  Serial.print(Temp_Int);
  Serial.print("     ");

  Serial.print("GyX = ");
  Serial.print(GyX_Int);
  Serial.print("  ");
  Serial.print("GyY = ");
  Serial.print(GyY_Int);
  Serial.print("  ");
  Serial.print("GyZ = ");
  Serial.println(GyZ_Int);

  if(PINF & 0b00000001)                                        // Als je
drukt op de drukknop
  {
    delay(250);                                                // Delay
voor niet direct in nieuwe loop te springen
    Hold = HIGH;                                               // Maak
de Hold varible hoog
    Serial.println("Hold");                                    // Print
hold op de seriele monitor

    while(Hold == 1)                                           // Zolang
Hold hoog is
    {
      if(PINF & 0b00000001)                                    // En als
je dan terug op de drukknop drukt
      {
        Hold = LOW;                                            // Maak
dan Hold laag
      }
    }

    delay(250);                                                // Delay
voor niet direct in nieuwe loop te springen
  }
}
```

Besluit

Door dit labo heb ik uit eigen ervaring en testen geleerd hoe de MPU6050 werkt. Door dit labo heb ik ook veel geleerd over I2C en ermee werken in Arduino en ook C code, maar om een of andere reden willen de waardes niet correct gemeten worden in C code (dit heb ik ondervonden in mijn I2C project voor EAC-ICT). De reden hiervoor weet ik echter niet. Mijn hypothese is dat het iets te maken heeft met de verschillen in libraries van C code en Arduino, maar dat kan ook niet kloppen.

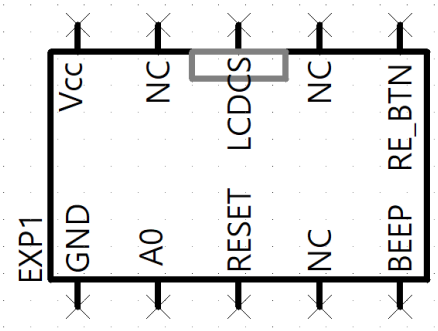
Labo Werking grafische LCD, driver en library

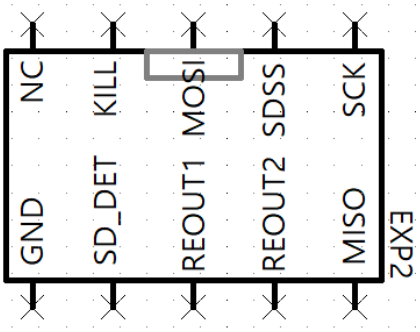
Doelstelling

Het doel van dit labo is : uitzoeken hoe we de grafische LCD kunnen aansturen. We gebruiken een bordje met de naam “MKS MINI12865 V2.0”. Het bestaat uit de LCD, zijn driver, een Rotary encoder, een buzzer, een SD-kaartlezer en een stopknop. Omdat ik de SD-kaartlezer en de knop niet nodig heb, worden ze niet besproken in dit labo.

Theorie

Eerst zal ik de pin-out van de printplaat uitleggen en dan kan ik daar verder op opbouwen. Op de PCB zitten twee 10 pin connectoren genaamd EXP1 en EXP2. Ik bespreek alleen de pinnen die ik nodig heb. Dus alles met SD en KILL valt weg.





- Vcc, hier sluit je de voeding aan die tot max +12V mag zijn door een ingebouwde spanningsomvormer.
- GND, het is de ground van de hele PCB.
- A0, deze pin laat de driver weten of de gestuurde data een instructie is of pixeldata (instructies wanneer laag).
- LCDCS, bepaalt wanneer de slave inkomende data mag inlezen (mag inlezen wanneer laag).
- RESET, wordt gebruikt om een hardware reset te doen aan het begin van het programma om alle instellingen te resetten.
- BEEP, deze pin is voor de buzzer, die moet gewoon hoog gemaakt worden en de buzzer zal piepen zolang de pin hoog blijft.
- RE_BTN, de output pin van het drukknop gedeelte van de Rotary encoder.
- REOUT1/2, de 2 output pinnen van de Rotary encoder.
- MOSI, de Master Out Slave In pin voor SPI. (Wordt nog meer uitgelegd)
- MISO, de Master In Slave Out pin voor SPI.
- SCK, de Serial Clock pin voor SPI0

De LCD heet 12864B11 en heeft een resolutie van 128x64. De driver van de LCD is de ST7567. Met instructies die nog zullen worden uitgelegd, kunnen we met die driver elke aparte pixel van de LCD aansturen.

Maar voor we de pixels kunnen aansturen moeten er eerst een aantal regels vastgelegd worden. Eerst moet de initialisatie gebeuren. Hierbij gaan we de RESET pin gebruiken. Die moet heel kort laag en terug hoog gemaakt worden. Daardoor worden alle interne registers klaar gemaakt voor gebruik en gereset.

Na de reset moeten ook een aantal registers geconfigureerd worden. De registers zijn zichtbaar in de (verkorte) tabel hieronder vanuit de datasheet. Omdat het instructies zijn, moet je als je 1 van deze registers wil configureren, A0 laag maken. Instructie 2, 3 en 4 worden in het begin niet geconfigureerd want ze bepalen de coördinaat van de pixel die je wil aan of uit zetten.

INSTRUCTION	A0	R/W	COMMAND BYTE								DESCRIPTION
			D7	D6	D5	D4	D3	D2	D1	D0	
(1) Display ON/OFF	0	0	1	0	1	0	1	1	1	D	D=1, display ON D=0, display OFF
(2) Set Start Line	0	0	0	1	S5	S4	S3	S2	S1	S0	Set display start line
(3) Set Page Address	0	0	1	0	1	1	Y3	Y2	Y1	Y0	Set page address
(4) Set Column Address	0	0	0	0	0	1	X7	X6	X5	X4	Set column address (MSB)
	0	0	0	0	0	0	X3	X2	X1	X0	Set column address (LSB)
(8) SEG Direction	0	0	1	0	1	0	0	0	0	MX	Set scan direction of SEG MX=1, reverse direction MX=0, normal direction
(9) Inverse Display	0	0	1	0	1	0	0	1	1	INV	INV =1, inverse display INV =0, normal display
(10) All Pixel ON	0	0	1	0	1	0	0	1	0	AP	AP=1, set all pixel ON AP=0, normal display
(11) Bias Select	0	0	1	0	1	0	0	0	1	BS	Select bias setting 0=1/9; 1=1/7

Dit is hoe ik de drivers zou moeten configureren :

- Display OFF dus 0b1010 1110.
- SEG Direction normal dus 0b1010 0000.
- Inverse Display OFF dus 0b1010 0110.
- All Pixels normal dus 0b1010 0100.
- Bias is 1/9 dus 0b1010 0010

Het scherm bestaat uit 9 rijen / “pages” / Y coördinaten en 132 kolommen / columns / X coördinaten. Elke column bestaat uit 8 bits. Deze pixels worden allemaal individueel aangestuurd door het display data RAM / DDRAM van de driver. Dus als je een pixel zou willen aansturen zou je eerst de “page” en “column” moeten bepalen van de pixels die je wil aansturen en ook met een 8-bit getal bepalen welke pixels aan en welke uit zullen zijn (zichtbaar op de foto hieronder).

Waar ook rekening mee moet gehouden worden is het Display register. Dat register bepaalt wanneer de opgeslagen data in het DDRAM moet gedisplayd worden op de LCD, het zorgt voor de verbinding tussen de 2. Dus bij de initialisatie van het programma moet display OFF zijn (0b1010 1110). Daarna bepaal je de positie (de page en column) en het 8-bit getal dat bepaalt welke pixels aan en welke pixels uit zullen zijn. Tenslotte maak je dan terug een verbinding met de LCD en DDRAM door display ON te zetten (0b1010 1111) en zal de ingeschreven data gedisplayd worden.

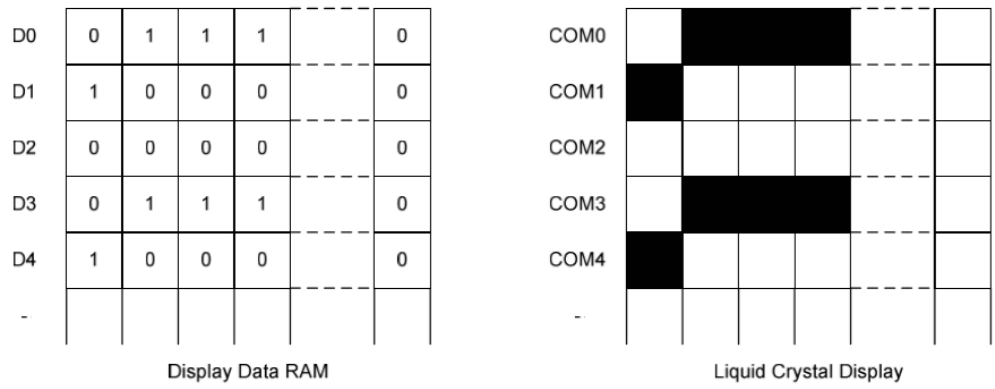
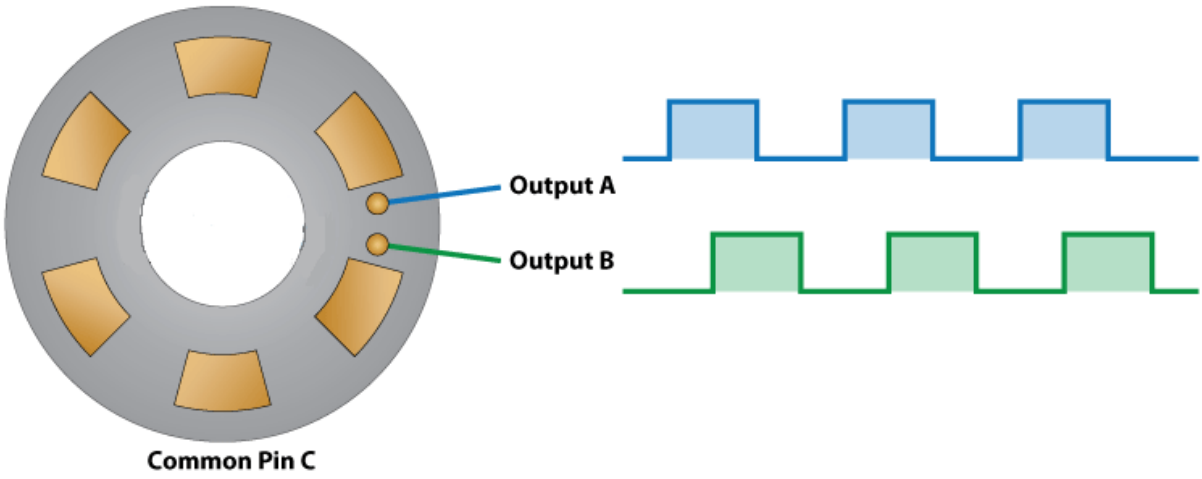


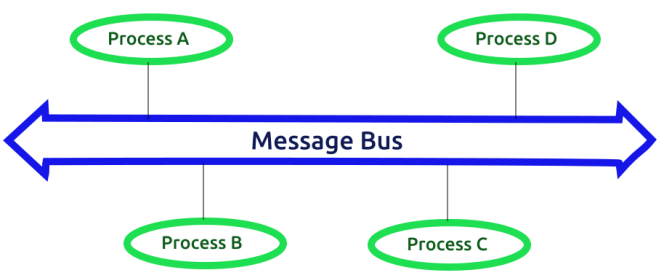
Fig 7. DDRAM Mapping Mode (Default Setting)

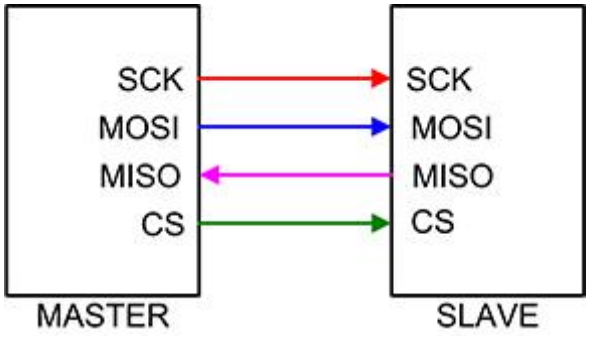
Fig 8. DDRAM Format



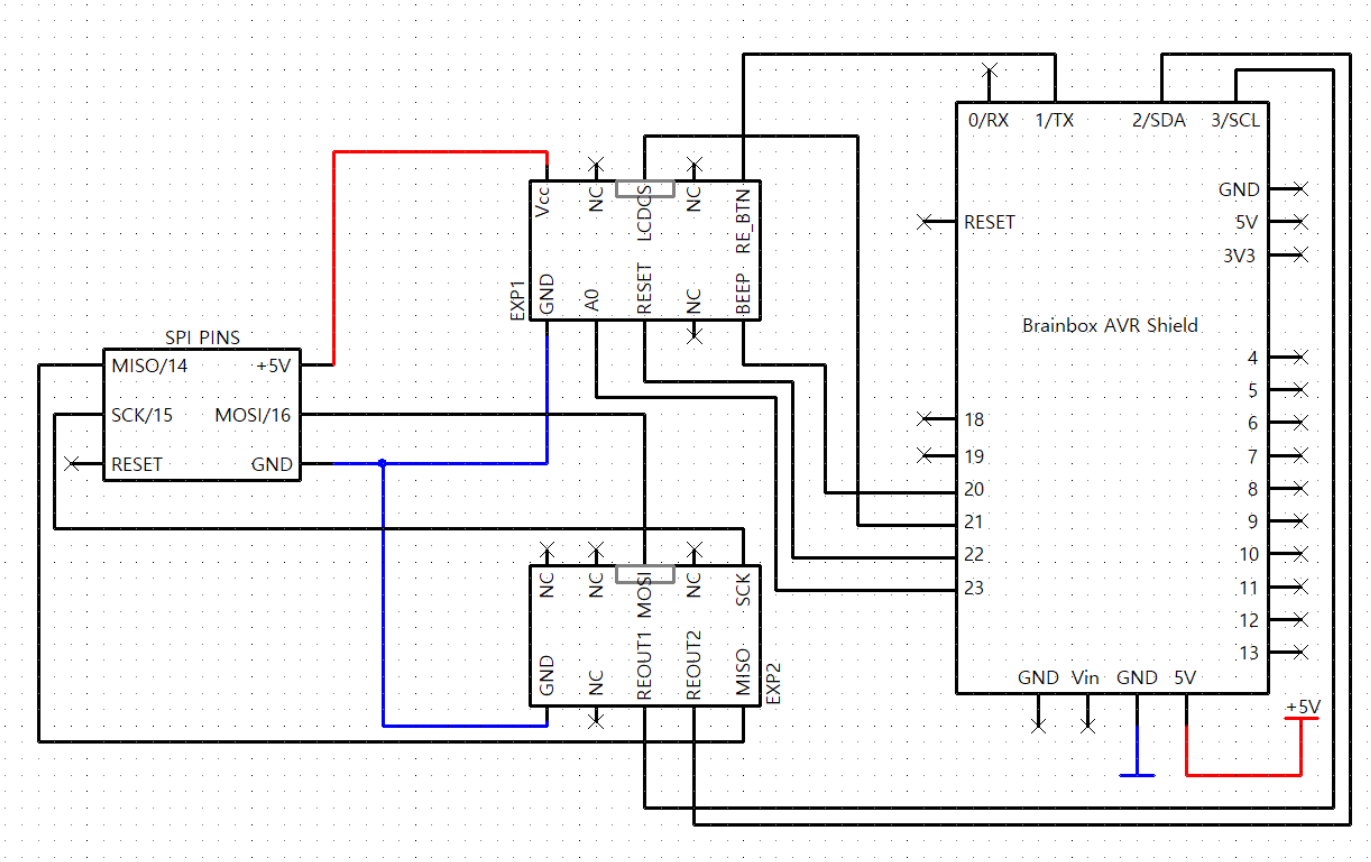
Dit is hoe de binnenkant van een Rotary encoder er uit ziet. Maar eerst zal ik het nut van een Rotary encoder uitleggen. Het is net zoals een potentiometer, maar er is geen einde aan het draaien naar links/rechts. Hierdoor is het perfect voor projecten waarbij bijvoorbeeld door een menu moet gescrold worden. Een Rotary encoder heeft 2 uitgangen genaamd A en B maar ook een uitgang voor de knop en de GND. Wanneer er aan de Rotary encoder wordt gedraaid zal er een plaatje dat verbonden is aan de GND ook meedraaien (de vorm van het plaatje is een beetje zoals de toerentalteller plaat in het PID verslag). Dat plaatje zal contact maken met 2 aparte plaatjes die verbonden zijn aan A en B. Doordat het grote plaatje bijvoorbeeld eerst A aanraakt en daarna B kunnen we zien welke uitgang eerst hoog wordt en daaruit de draairichting bepalen.

SPI is een vorm van seriële communicatie via een bus, zoals I2C. Het bestaat uit 4 of 3 draden (de brainbox heeft CS niet). De SCK is de seriële klok die de datasnelheid bepaalt net zoals bij I2C. De MOSI is de draad waar de master data gaat sturen naar de slave. De MISO is de draad waar de slave data gaat terugsturen naar de master. Ten sloten de CS is al uitgelegd bij de pin-out van de GLCD.





Schema



Beschrijving

Het is een simpel programma. Het gebruikt de 3 componenten die in de doelstelling staan. Wanneer je aan de rotary encoder draait in wijzerzin dan zal er een waarde (zichtbaar op de seriële monitor) vergroten en als je tegenwijzerzin draait dan zal die verminderen. Als je dan op de knop (op de rotary encoder) zou drukken, dan zullen alle pixels op het scherm aan gaan en zal de buzzer ook even biepen. Als je voor de 2de keer drukt dan gaan de pixels terug uit en biept de buzzer ook natuurlijk.

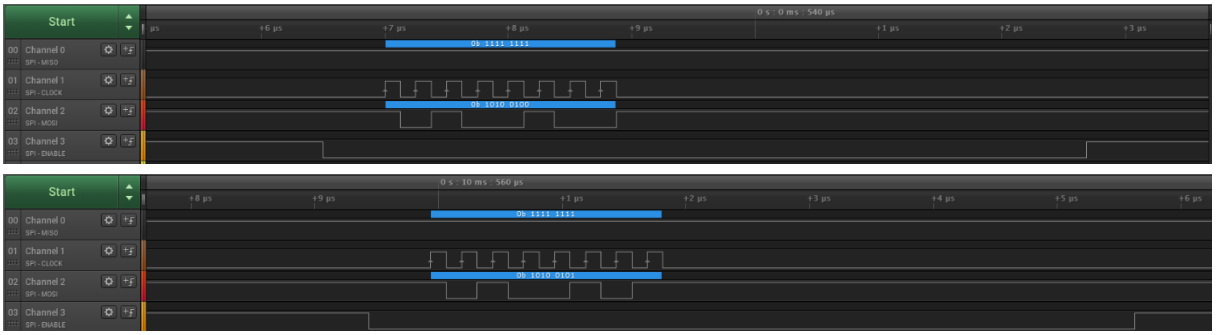
Code

```
#include <SPI.h>

#define PD0 3
#define PD1 2
#define PD3 1

// A0 -> PF0
// RESET -> PF1
// CS -> PF4
// BEEP -> PF5
// RO_1 -> PD0/SCL
// RO_2 -> PD1/SDA
// RO_Button -> PD3/TX

// Als 1 eerst laag is dan is het clockwise
// Functions voor wanneer er op de RO wordt gedrukt/gedraaid
void RO_1();
void RO_2();
void RO_Button();

volatile boolean RO_1_Eerst, RO_2_Eerst, PixelsON = false;  // Variable om
te bepalen welke uitgang als eerste laag werd en of alle pixels al aan zijn
of niet
boolean Tester1, Tester2;   // Variable die bepalen wanneer er terug een
interrupt mag zijn
long Teller = 0;    // Teller om te zien hoeveel keer je al hebt gedraaid
aan de RO


void setup()
{
  attachInterrupt(digitalPinToInterrupt(PD0), RO_1, FALLING);        //
Interrupt bij INT0 wanneer de pin van hoog naar laag gaat
  attachInterrupt(digitalPinToInterrupt(PD1), RO_2, FALLING);        //
Interrupt bij INT1 wanneer de pin van hoog naar laag gaat
  attachInterrupt(digitalPinToInterrupt(PD3), RO_Button, FALLING);   //
Interrupt bij INT3 wanneer de pin van hoog naar laag gaat

  Serial.begin(9600);
  SPI.begin();

  PORTF &= 0b11111101;  // Reset laag
  delay(1);
  PORTF |= 0b00000010;  // Reset hoog
  delay(1);

  PORTF &= 0b11111110;  // A0 laag
  PORTF |= 0b00010000;  // CS hoog

  SPI.beginTransaction(SPISettings(4000000, MSBFIRST, SPI_MODE0));
}

void loop()
{
  if(RO_1_Eerst)
  {
    detachInterrupt(digitalPinToInterrupt(PD1));   // Stop interrupts aan
RO_2
    Teller++;
```

```
    while((Tester1 == false) | (Tester2 == false))
    {
      Tester1 = digitalRead(PD0);
      Tester2 = digitalRead(PD1);
    }

    RO_1_Eerst = false;
    RO_2_Eerst = false;
    attachInterrupt(digitalPinToInterrupt(PD1), RO_2, FALLING);   // Start
interrupts terug aan RO_2
  }

  if(RO_2_Eerst)
  {
    detachInterrupt(digitalPinToInterrupt(PD0));  // Stop interrupts aan
RO_1
    Teller--;

    while((Tester1 == false) | (Tester2 == false))
    {
      Tester1 = digitalRead(PD0);
      Tester2 = digitalRead(PD1);
    }

    RO_1_Eerst = false;
    RO_2_Eerst = false;
    attachInterrupt(digitalPinToInterrupt(PD0), RO_1, FALLING);   // Start
interrupts terug aan RO_1
  }

  Serial.println(Teller);
}

void RO_1()          // Interrupt functie wanneer 1 als eerste laag wordt
{
  RO_1_Eerst = true;
  RO_2_Eerst = false;
}

void RO_2()          // Interrupt functie wanneer 2 als eerste laag wordt
{
  RO_2_Eerst = true;
  RO_1_Eerst = false;
}

void RO_Button()     // Interrupt functie wanneer er op de knop wordt
gedrukt
{
  PORTF |= 0b00100000;  // BEEP hoog
  delay(1);             // BEEP hoog houden voor 1ms
  PORTF &= 0b11011111;  // BEEP laag

  if(PixelsON == false)   // Zijn alle pixels uit
  {
    PORTF &= 0b11101111;        // Chip select laag
    SPI.transfer(0b10100101);   // Zet alle pixels aan
    PORTF |= 0b00010000;        // Chip select hoog
    PixelsON = true;
  }
```

```
  else    // Zijn alle pixels aan
  {
    PORTF &= 0b11101111;          // Chip select laag
    SPI.transfer(0b10100100);     // Zet alle pixels uit
    PORTF |= 0b00010000;          // Chip select hoog
    PixelsON = false;
  }
}
```

Metingen



Hier zijn de 2 metingen van de data die naar de driver wordt gestuurd. We zien dat op de 1ste foto via de MOSI lijn 0b1010 0100 wordt gestuurd, dus alle pixels zijn uit dan. Op de 2de foto zien we dat via de MOSI lijn 0b1010 0101 wordt gestuurd, dus alle pixels zijn dan aan. Er wordt niets op de MISO lijn gestuurd want er moet niets terug naar de master worden gestuurd.

In deze 2 foto's zien we duidelijk de 2 mogelijke uitkomsten uit de 2 uitgangen van de Rotary encoder. Bij de 1ste foto wordt RO_1 of A eerst laag dus ik draaide in wijzerzin. In de 2de foto wordt RO_2 of B eerst laag dus draaide ik in tegenwijzerzin.

U8G2 Library

De U8G2 library is een library voor meerdere GLCDs waarmee je op een simpele manier letters, figuren etc. op de LCD kunt displayen. De library kan een groot aantal LCDs aansturen maar dat maakt ons niet zo veel uit zolang het onze GLCD kan aansturen, wat de library ook kan.

Er zijn 3 modes waarmee de LCD kan worden aangestuurd. Single page, double page en full frame mode. Single page mode zorgt ervoor dat er 1 “page” wordt opgeslagen in het RAM van de brainbox. Double page mode doet hetzelfde als single page mode maar dan met 2 “pages” i.p.v. 1. Tenslotte doet full frame mode ook hetzelfde maar dan met een volledige kopie van de display. Dat is de snelste van de 3 modes, maar een nadeel ervan is dat die ook het meeste RAM van de brainbox inpakt.

Besluit

We hebben 2 manieren gevonden om de GLCD aan te sturen. Via SPI of via de library. Het is duidelijk dat we gaan met de library. De reden hiervoor zijn simpel : het is heel gemakkelijk te gebruiken, het maakt de code zeker een stuk simpeler en minder groot en het kan ons veel tijd besparen. Ik moet nog alleen later beslissen welke van de 3 modes ik ga gebruiken. De reden hiervoor is dat ik niet weet hoeveel procent van het RAM over gaat blijven na de “main” code voor de robot.

Labo Energieverbruik

Doelstelling

Het doel van dit labo is berekenen en meten hoeveel Watt mijn robot in totaal zal verbruiken. Dit doe ik door de vermogens te berekenen van de HC06, Arduino Mega, MPU6050, DRV8825/stappenmotoren. Tenslotte worden al die waardes opgeteld tot 1 eindwaarde.

Arduino Mega

Volgens de officiële site van Arduino zelf zou die een max stroom kan trekken van 200mA maar trekt meestal rond de 75mA met een spanning van 5V. De theoretische waarde die ik dan kies is 75mA.

Mijn gemeten waarde was 77mA

HC06

Volgens de datasheet van de module zal die een stroom van 30-40mA (dus 35mA) trekken tijdens het “paren”. Voor de rest zal die een stroom van maar 8mA trekken. De voedingspanning is 5V.

Mijn gemeten waarde was een onstabiele stroom tussen 37mA en 43mA tijdens het paren dus ik zal gaan voor een stroom van 40mA en ik mat 8mA na het paren.

MPU6050

Volgens de datasheet zal deze component ongeveer 20mA trekken met een spanning van 5V.

Mijn gemeten waarde was 38mA.

DRV8825 + Stappenmotoren

Bij een bronspanning van 10.8V (de 3 batterijen) zullen de motoren ongeveer 250mA per stuk trekken. Dus in totaal 500mA. Maar bij stroompieken kan die 250mA stijgen tot wel 800mA. Dat verhoogt de totale stroom naar 1.6A

Mijn gemeten waarde was ongeveer 220mA per motor. Maar ik heb geen gemeten waarde voor de stroompieken.

GLCD

De backlight van de GLCD trekt ongeveer 25mA tot 35mA met een voedingspanning van 3.3V. Dus 30mA is mijn theoretische waarde.

Mijn gemeten waarde was 28mA.

Tabel

	Voedingspanning	Theoretische Stroom	Gemeten Stroom	Theoretisch Vermogen	Gemeten Vermogen
Arduino Mega	5V	75mA	77mA	0,385W	0,375W
HC06	5V	35mA	40mA	0,2W	0,175W
MPU6050	5V	20mA	38mA	0,19W	0,1W
DRV8825	10,8V	500mA en 1,6A	440mA en 1,6A	4,752W en 17,28W	5,4W en 17,28W
GLCD	3,3V	30mA	28mA	0,0924W	0,099W
				5,62W - 18,14W	6,149W - 18,03W

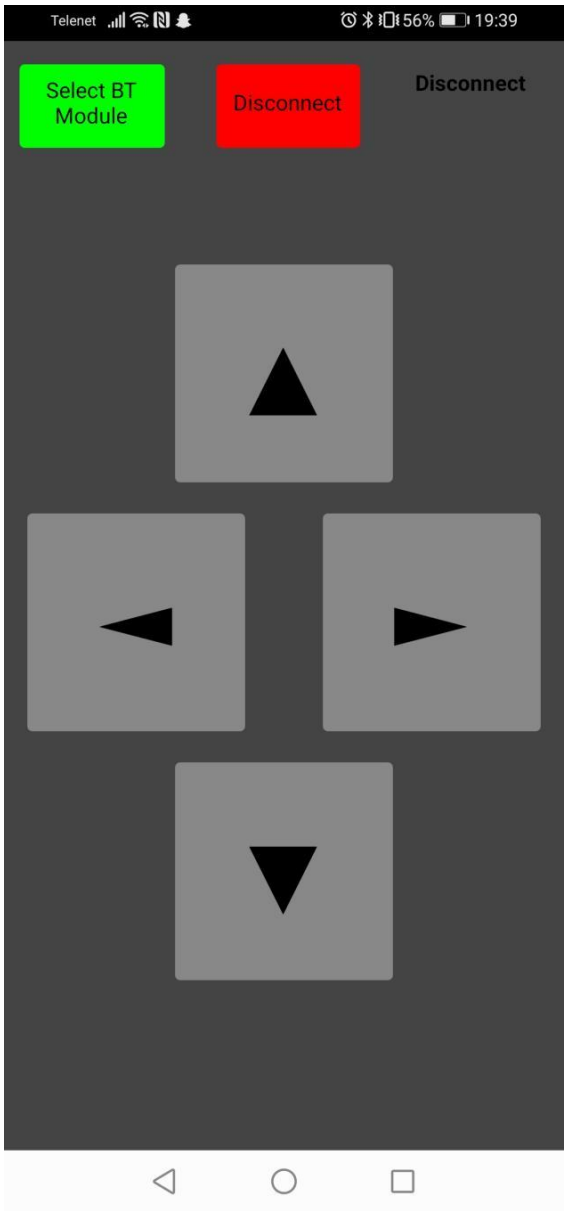
Besluit

De voeding voor de robot bestaat uit 2 batterijen. Een 9V batterij en 3 herlaadbare 18650 batterijen (10.8V). Die batterijen hebben een capaciteit van 5000mAh dus die kunnen de stroompieken zeker aan. Met die capaciteit kunnen ze de motoren voeden voor een 187 minuten (5000/1600)! De 9V batterij heeft een capaciteit van 500mAh. Als de rest van de schakeling dan nog 183mA trekt dan zal de batterij de schakeling 224 minuten kunnen voeden! Deze resultaten zijn zeker voldoende.

Programma

Applicatie

Dit is de app. De app heeft 2 functies : de robot besturen via de 4 knoppen en de hoeveelheid graden de robot is gekanteld tonen. De app ziet er simpel uit, en dat komt grotendeels omdat ik het gyro sensor deel nog moet toevoegen. Maar de app doet wel het belangrijkste, de robot besturen.



De reden voor de variable waardes in de code zijn dat er formules worden gebruikt om een waarde van -0.5 tot 0.5 te krijgen. Met 0.5 vooruit en -0.5 achteruit. Dus neem bijvoorbeeld, als er niets wordt gedrukt is de waarde 127. Want (in de code) wordt 127 gedeeld door 255 en verkleint met 0.5. Dus de uitkomst is 0.

Software

```
// B-ROBOT  SELF BALANCE 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)
// Updated: 20/05/2016
// Version: 2.0
// 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 <Arduino.h>
#include <U8g2lib.h>

#ifdef U8X8_HAVE_HW_SPI
#include <SPI.h>
#endif
#ifdef U8X8_HAVE_HW_I2C
#include <Wire.h>
#endif

#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 175
#define MAX_STEERING 150
```

Hier worden de nodige libraries toegevoegd en sommige parameters bepaald. Zoals baudrate, PID waardes, max waardes.

```
#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 0.16 // 0.16
#define KD 68 // 63
#define KP_THROTTLE 0.07 // 0.07
#define KI_THROTTLE 0.04 // 0.04

// Control gains for raiseup (the raiseup movement requiere special control
parameters)
#define KP_RAISEUP 0.16
//#define KD_RAISEUP 68
#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 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

#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

// Pinout of the GLCD
#define RE2 2 // Rotary encoder output 2
#define RE1 3 // Rotary encoder output 1
#define RE_S 18 // Rotary encoder switch
#define MISO 50 // SPI Master in Slave out pin
#define MOSI 51 // SPI Master out Slave in pin
#define SCK 52  // SPI Serial Clock pin
#define CS 54 // GLCD driver Chip Select pin
#define BEEP 55 // Buzzer pin
#define RESET 56 // GLCD driver reset pin
#define A0 57 // GLCD data control pin

#define S1 14
#define S2 15
#define S3 16
```

```
U8G2_ST7565_JLX12864_F_4W_HW_SPI u8g2(U8G2_R2, /* cs=*/ 54, /* a0=*/ 57, /*
reset=*/ 56);


// MPU control/status vars
boolean 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;
int angle_adjusted_int;

// 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 de code eenvoudiger maken, door gebruik van define regels.
Variabelen waar de data van de gyrosensor en accelerometer in worden opgeslagen.
Variabelen voor de data van de app in op te slagen.
Variabelen voor de data voor de motoren aan te sturen.

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

boolean Calibration = 0;

boolean Scroll = 0;
unsigned char Pagina = 0;
unsigned char Snelheid = 50;
unsigned char SnelheidXas = 25;
unsigned char Gevoeligheid = 50;
unsigned char GevoeligheidXas = 25;

unsigned char IntroPagina = 1;

// 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);
}
```

Beginnen met data van de MPU te lezen.

```
// Quick calculation to obtein 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);
}
```

```
// PD controller implementation(Proportional, derivative). DT is in
miliseconds
float stabilityPDControl(float DT, float input, float setPoint,  float Kp,
int 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)
```

Dit is de PD regeling functie voor de stabiliteit van de robot.

```
  //    And the second using the setpoint to make it a bit more agressive
setPoint-setPoint(t-1)
  output = Kp * error + (Kd * (setPoint - setPointOld) - Kd * (input -
PID_errorOld2)) / DT;
  PID_errorOld2 = PID_errorOld;
  PID_errorOld = input;  // error for Kd is only the input component
  setPointOld = setPoint;
  return (output);
}

// PI controller implementation (Proportional, integral). DT is in
miliseconds
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);

  output = Kp * error + Ki * PID_errorSum * DT * 0.001; // DT is in
miliseconds...
  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" "\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
```

Dit is de PI regeling functie voor de snelheid van de robot.

De interrupt functies om elke microseconde een puls naar de motor driver te sturen.

```
  digitalWrite(11, HIGH); // STEP MOTOR 1
  delay_1us();
  digitalWrite(11, LOW);
}
// 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
  digitalWrite(7, HIGH); // STEP MOTOR 2
  delay_1us();
  digitalWrite(7, LOW);
}



/* 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 (USART0_RX_vect) // interrupt routine that executes every time a new
byte is received
{
 UCSR0B &= ~(1 << RXCIE0); // disable RX interrupt into the variable "
ByteReceived " - the RXint flaq is automatically cleared now
 throttleHC = UDR0; // Fetch the received byte value
 throttleHC = throttleHC/255; // Constrain throttleHC between 0 - 0.5
 while ((UCSR0A & (1 << RXC0)) == 0) {}; // wait until byte 2 is received

 // Do nothing until data have been received and is ready to be read from
UDR
 steeringHC = UDR0; // receive byte 2
 steeringHC = steeringHC/255; // Constrain steeringHC between 0 - 0.5
 UCSR0B |= (1 << RXCIE0); //Re-enable RX interrupt
}
```

Interrupt functie voor de inkomende data van de app te lezen.

```
// 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
```

Hier wordt de draaizin van de motoren ingesteld en de juiste data gestuurd om motor 1 aan te sturen.

```
#endif

  if (speed == 0)
  {
    timer_period = ZERO_SPEED;
    dir_M1 = 0;
  }
  else if (speed > 0)
  {
    timer_period = 2000000 / speed; // 2Mhz timer
    dir_M1 = 1;
    digitalWrite(12, LOW); // DIR Motor 1 (Forward)
  }
  else
  {
    timer_period = 2000000 / -speed;
    dir_M1 = -1;
    digitalWrite(12, HIGH); // Dir Motor 1
  }
  if (timer_period > 65535)   // Check for minimun speed (maximun 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;

#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
```

Deze functie zorgt ervoor dat motor 2 zijn max snelheid niet overschrijd.

```
    dir_M2 = 1;
    digitalWrite(8, HIGH);   // Dir Motor2 (Forward)
  }
  else
  {
    timer_period = 2000000 / -speed;
    dir_M2 = -1;
    digitalWrite(8, LOW);  // 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;
}



void Beep(char keuze)
{
  if(keuze == 1)
  {
    digitalWrite(BEEP, HIGH); // Beep when robot is ready
    delay(50);
    digitalWrite(BEEP, LOW);
    delay(50);
    digitalWrite(BEEP, HIGH);
    delay(50);
    digitalWrite(BEEP, LOW);
    delay(50);
    digitalWrite(BEEP, HIGH);
    delay(50);
    digitalWrite(BEEP, LOW);
    delay(50);
  }

  if(keuze == 2)
  {
    digitalWrite(BEEP, HIGH);
    delay(50);
    digitalWrite(BEEP, LOW);
  }
}

// INITIALIZATION
void setup()
{
  // STEPPER PINS ON JJROBOTS BROBOT
BRAIN BOARD
  pinMode(4, OUTPUT); // ENABLE MOTORS
  pinMode(7, OUTPUT); // STEP MOTOR 1
  pinMode(8, OUTPUT); // DIR MOTOR 1
  pinMode(11, OUTPUT); // STEP MOTOR 2
  pinMode(12, OUTPUT); // DIR MOTOR 2
  pinMode(S1, INPUT); // S1 (Black)
  pinMode(S2, INPUT); // S2 (Red)
  pinMode(S3, INPUT); // S3 (Black)
  pinMode(55, OUTPUT); // BEEP
```

Functie voor de buzzer te laten beepen.

Alle nodige pins input en output maken. De buzzer en 2 motoren ook stop zetten bij initialisatie.

```
  digitalWrite(4, HIGH);  // Disbale motors
  digitalWrite(55, LOW);  // Disbale buzzer

  //initialize RS232 comms on TX & RX pin to communicate with HC06
Bluetooth module
  UCSR0B = (1 << RXEN0 ) | (1 << TXEN0 ); // Enable uart1 for transmit and
receive
  UCSR0C = (1 << UCSZ10 ) | (1 << UCSZ11 );
  UBRR0H = (BAUD_PRESCALE >> 8);  // Set baud rate register
  UBRR0L = BAUD_PRESCALE; // Set baud rate register
  UCSR0B |= (1 << RXCIE0);  // 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;

  u8g2.begin();
  u8g2.setContrast(200);

  u8g2.clearBuffer();
  u8g2.setFont(u8g2_font_ncenB08_tr);
  u8g2.drawStr(10,25,"Balancerende Robot");
  u8g2.drawStr(25,45,"Jelte Boumans");
  u8g2.sendBuffer();
  delay(3000);

  while(IntroPagina != 6)
  {
    if((digitalRead(S1)) && (IntroPagina != 1))
    {
      IntroPagina--; // Previous page
      Beep(2);
      delay(150);
    }

    if(digitalRead(S2))
    {
      IntroPagina = 6;  // Skip
      Beep(2);
      delay(150);
    }

    if(digitalRead(S3))
    {
      IntroPagina++;  // Next page
      Beep(2);
      delay(150);
    }

    if(IntroPagina < 6)
    {
      u8g2.clearBuffer();
      u8g2.setFont(u8g2_font_unifont_t_symbols);
      u8g2.drawGlyph(1,10, 0x2190);
      u8g2.drawGlyph(119,10, 0x2192);
      u8g2.setFont(u8g2_font_profont11_tr);
```

Instellingen voor Bluetooth en I2C.

Introductie die op de GLCD wordt geprint.

De handleiding waar alles in staat dat je nodig hebt om de robot te gebruiken. Je kan pagina per pagina lezen, of als je het al weet de handleiding gewoon overslaan.

```
    u8g2.drawStr(53,8,"SKIP");
  }

  switch(IntroPagina)
  {
    case 1:
      u8g2.setFont(u8g2_font_ncenB08_tr);
      u8g2.drawStr(5,22,"Laat eerst de robot op");
      u8g2.drawStr(7,32,"zijn rug liggen tot je");
      u8g2.drawStr(5,42,"de robot 3 keer achter");
      u8g2.drawStr(9,52,"elkaar hoort biepen.");
      u8g2.setFont(u8g2_font_profont11_tr);
      u8g2.drawStr(119,62,"1");
    break;

    case 2:
      u8g2.setFont(u8g2_font_ncenB08_tr);
      u8g2.drawStr(5,22,"Als dit langer dan 15s");
      u8g2.drawStr(7,32,"duurt, reset de robot.");
      u8g2.drawStr(16,42,"Na de biep zit je");
      u8g2.drawStr(13,52,"in de instellingen.");
      u8g2.setFont(u8g2_font_profont11_tr);
      u8g2.drawStr(119,62,"2");
    break;

    case 3:
      u8g2.setFont(u8g2_font_ncenB08_tr);
      u8g2.drawStr(16,22,"Daarna kan je de");
      u8g2.drawStr(12,32,"motoren aanzetten,");
      u8g2.drawStr(12,42,"en de robot rechop");
      u8g2.drawStr(11,52,"zetten om te rijden.");
      u8g2.setFont(u8g2_font_profont11_tr);
      u8g2.drawStr(119,62,"3");
    break;

    case 4:
      u8g2.setFont(u8g2_font_ncenB08_tr);
      u8g2.drawStr(7,22,"Je hebt ook de keuze");
      u8g2.drawStr(10,32,"om de robot met de");
      u8g2.drawStr(5,42,"app te besturen via de");
      u8g2.drawStr(12,53,"bluetooth module.");
      u8g2.setFont(u8g2_font_profont11_tr);
      u8g2.drawStr(119,62,"4");
    break;

    case 5:
      u8g2.setFont(u8g2_font_ncenB08_tr);
      u8g2.drawStr(15,22,"Tenslotte als je");
      u8g2.drawStr(15,32,"de robot terug op");
      u8g2.drawStr(18,42,"zijn rug legt, zie");
      u8g2.drawStr(13,52,"je de instellingen.");
      u8g2.setFont(u8g2_font_profont11_tr);
      u8g2.drawStr(119,62,"5");
    break;

    case 6:
      u8g2.clearBuffer();
      u8g2.setFont(u8g2_font_ncenB08_tr);
      u8g2.drawStr(15,35,"Wacht even AUB...");
      u8g2.sendBuffer();
    break;
```

```
    }

    u8g2.sendBuffer();
  }

  #if DEBUG > 0
    delay(9000);
  #else
    delay(2000);
  #endif

  //mpu.initialize();
  // Manual MPU initialization... accel=2G, gyro=2000°/s, filter=20Hz BW,
output=200Hz
  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(4);   // 0=1khz 1=500hz, 2=333hz, 3=250hz 4=200hz
  mpu.setSleepEnabled(false);

  delay(500);
  devStatus = mpu.dmpInitialize();
  if (devStatus == 0) {
    // turn on the DMP, now that it's ready
    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)
  }

  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;

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

  delay(200);

  // Enable stepper drivers and TIMER interrupts
```

Instellingen van de MPU handmatig goed zetten. Data op hoogste frequentie sturen.

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

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



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

  throttle = (throttleHC - 0.5) * max_throttle;  //eerste 0.5 moet throttle
data van HC-06 zijn
  // We add some exponential on steering to smooth the center band
  steering = steeringHC - 0.5;                   //eerste 0.5 moet steering
data van HC-06 zijn
  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();

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

    if((angle_adjusted <= -89) && (angle_adjusted >= -92)) // If the gyro
has finished calibrating and the program has been running for a while (this
would often false trigger without the millis())
    {
      if((Calibration == false) && (millis() >= 13000))
      {
        Calibration = true;
        u8g2.clearBuffer();
        u8g2.sendBuffer();
        Beep(1);
```

De ingekomen data van de app omrekenen naar data voor de motoren aan te sturen.

Als het FIFO pakket groter is dan zijn max/18, dan moet het gereset worden en opnieuw verstuurd worden.

Laat de buzzer beepen als de robot klaar is met calibreren.

Daarna worden er de nodige berekeningen gedaan om de robot recht te houden met de ingelezen data.

```
      }
    }

    // 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 < 76) && (angle_adjusted > -76)) // Is robot ready
(upright?)
    {
      // NORMAL MODE
      digitalWrite(4, LOW);  // Motors enable
      setMotorSpeedM1(motor1);
      setMotorSpeedM2(motor2);

      // 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
```

Als de robot tussen -70° en 70° ligt dan mogen de motoren draaien.

```
      {
        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;
      if(Calibration) Menu();
    }

  } // End of new IMU data
}



void Menu()
{
  if(Pagina == 0)
  {
    u8g2.clearBuffer();
    PaginaInstel();
    u8g2.setFont(u8g2_font_unifont_t_symbols);
    if(Scroll == 0) u8g2.drawGlyph(15,36, 0x2192);
    if(Scroll == 1) u8g2.drawGlyph(15,60, 0x2192);
    u8g2.sendBuffer();

    if((digitalRead(S1) == 1) || (digitalRead(S3) == 1))
    {
      Scroll = !Scroll;
      Beep(2);
      delay(300);
    }

    if((digitalRead(S2) == 1) && (Scroll == 0))
    {
      Pagina = 1;
      Scroll = 0;
      Beep(2);
      delay(300);
    }

    if((digitalRead(S2) == 1) && (Scroll == 1))
    {
      Pagina = 2;
      Scroll = 0;
      Beep(2);
      delay(300);
    }
  }
```

Hier is een menu dat je toegang tot krijgt wanneer de robot op zijn rug ligt. Je kan er in de max_throttle en de Kd van de PD regelaar aanpassen.

```
if(Pagina == 1)
{
  u8g2.clearBuffer();
  PaginaSnel();
  u8g2.setFont(u8g2_font_unifont_t_symbols);
  if(Scroll == 0) u8g2.drawGlyph(15,36, 0x2192);
  if(Scroll == 1) u8g2.drawGlyph(15,60, 0x2192);
  u8g2.setCursor(50,34);
  u8g2.print("%");
  u8g2.setFont(u8g2_font_profont12_tf);
  u8g2.setCursor(30,34);
  u8g2.print(Snelheid);
  u8g2.drawBox(64,25,SnelheidXas,8);
  u8g2.sendBuffer();

  if((digitalRead(S1) == 1) || (digitalRead(S3) == 1))
  {
    Scroll = !Scroll;
    Beep(2);
    delay(300);
  }

  if((digitalRead(S2) == 1) && (Scroll == 0))
  {
    if(Snelheid == 100) Snelheid = 0;
    Snelheid = Snelheid + 10;
    constrain(Snelheid, 0, 100);

    if(max_throttle == 225) max_throttle = 125;
    else max_throttle = max_throttle + 10;
    constrain(max_throttle, 125, 225);

    if(SnelheidXas == 50) SnelheidXas = 0;
    SnelheidXas = SnelheidXas + 5;
    constrain(SnelheidXas, 0, 50);

    Beep(2);

    delay(300);
  }

  if((digitalRead(S2) == 1) && (Scroll == 1))
  {
    Pagina = 0;
    Scroll = 0;
    Beep(2);
    delay(300);
  }
}


if(Pagina == 2)
{
  u8g2.clearBuffer();
  PaginaGevoelig();
  u8g2.setFont(u8g2_font_unifont_t_symbols);
  if(Scroll == 0) u8g2.drawGlyph(15,36, 0x2192);
  if(Scroll == 1) u8g2.drawGlyph(15,60, 0x2192);
  u8g2.setCursor(50,34);
  u8g2.print("%");
```

```
    u8g2.setFont(u8g2_font_profont12_tf);
    u8g2.setCursor(30,34);
    u8g2.print(Gevoeligheid);
    u8g2.drawBox(64,25,GevoeligheidXas,8);
    u8g2.sendBuffer();

    if((digitalRead(S1) == 1) || (digitalRead(S3) == 1))
    {
      Scroll = !Scroll;
      Beep(2);
      delay(300);
    }

    if((digitalRead(S2) == 1) && (Scroll == 0))
    {
      if(Gevoeligheid == 100) Gevoeligheid = 0;
      Gevoeligheid = Gevoeligheid + 10;
      constrain(Gevoeligheid, 0, 100);

      if(Kd == 88) Kd = 48;
      else Kd = Kd + 4;
      constrain(Kd, 48, 88);

      if(GevoeligheidXas == 50) GevoeligheidXas = 0;
      GevoeligheidXas = GevoeligheidXas + 5;
      constrain(GevoeligheidXas, 0, 50);

      Beep(2);

      delay(300);
    }

    if((digitalRead(S2) == 1) && (Scroll == 1))
    {
      Pagina = 0;
      Scroll = 0;
      Beep(2);
      delay(300);
    }
  }
}

void PaginaInstel()
{
  u8g2.setFont(u8g2_font_ncenB08_tr);
  u8g2.drawStr(32,10,"Intellingen");
  u8g2.drawLine(32,13, 93,13);
  u8g2.drawFrame(5,20, 118,20);
  u8g2.drawFrame(5,44, 118,20);
  u8g2.drawStr(50,33,"Snelheid");
  u8g2.drawStr(40,58,"Gevoeligheid");
}

void PaginaSnel()
{
  u8g2.setFont(u8g2_font_ncenB08_tr);
  u8g2.drawStr(38,10,"Snelheid");
  u8g2.drawLine(38,13, 86,13);
  u8g2.drawFrame(5,20, 118,20);
  u8g2.drawFrame(5,44, 118,20);
  u8g2.drawStr(50,58,"Terug");
```

```
}

void PaginaGevoelig()
{
  u8g2.setFont(u8g2_font_ncenB08_tr);
  u8g2.drawStr(27,10,"Gevoeligheid");
  u8g2.drawLine(27,13, 100,13);
  u8g2.drawFrame(5,20, 118,20);
  u8g2.drawFrame(5,44, 118,20);
  u8g2.drawStr(50,58,"Terug");
}
```

Besluit

Ik heb een balancerende robot die bestuurbaar is via een app gemaakt. Bij het maken van deze robot moesten er parameters heel precies afgesteld worden. Want de robot mocht niet te snel draaien/rijden en ook niet te gevoelig willen balanceren. De behuizing moest ook veel over nagedacht worden, hoe hoger de robot hoe makkelijker die is te balanceren. Maar als je een grote behuizing hebt moet die ook stevig zijn gebouwd. Ik had niet alles met de behuizing goed gedaan, ik had sommige delen niet groot genoeg gemaakt voor wat er allemaal ingestoken moesten worden. Maar dat veranderd niet veel aan de conclusie. De robot kan gebruikt worden door het school op hun opendeurdagen, want kinderen zullen zeker geïnteresseerd zijn door een robot die ze met een app kunnen besturen.

Ik vond dit project heel leuk om te maken, zoals eerder is vermeld ben ik geïnteresseerd in robots sinds ik een kind was. Om dan nu zelf zo een robot te mogen maken is geweldig voor mij. Ik heb zeer veel door deze GIP geleerd: zoals het concept van PID en meer over gesloten processen, maar ook over gyrosensoren en accelerometers. Het feit dat ik de software moest aanpassen om te werken voor mijn robot gaf me soms veel stress want de code is vaak vrij complex, dus heel moeilijk om dingen in te veranderen en toe te voegen. Maar door de quarantaine voor COVID-19 had ik extra veel tijd om aan die code te werken en heb ik de robot ten slotte nog kunnen late werken. Wat zeker het moeilijkste deel was, was de PID regelaar afstellen want ik had zoiets nog nooit gedaan en de robot gebruikt niet een eenvoudige regelaar. Maar ik ben toch heel tevreden over het resultaat en wat ik allemaal heb bijgeleerd.

Bronnenlijst

In deze bronnenlijst verdeel ik mijn bronnen in 2 soorten : veelgebruikte bronnen, dat zijn de bronnen die ik over heel het jaar of een lange periode heb gebruikt. En bronnen die ik minder heb gebruikt of in een kleine periode.

Veelgebruikte bronnen

E2CR8 Brainbox AVR. (Geraadpleegd in eerste helft van het jaar 2019)

https://e2cre8.be/?page_id=21

E2CR8 Apps maken 3de graad. https://e2cre8.be/?page_id=2422

Arduino Mega 2560 (Geraadpleegd in tweede helft van het jaar (2020))

https://www.arduino.cc/en/Guide/ArduinoMega2560

Arduino Forum https://forum.arduino.cc/

Google afbeeldingen, Wikipedia, YouTube, Farnell.

Minder gebruikte bronnen

Julio J. THE B-ROBOT EVO 2 (the self balancing robot). (Geraadpleegd op 24 februari 2020) https://www.jjrobots.com/much-more-than-a-self-balancing-robot/

Video's over de MPU6050. (Geraadpleegd op 11 november 2019)

https://www.youtube.com/watch?v=4BoIE8YQwM8

https://www.youtube.com/watch?v=eqZgxR6eRjo

Pololu DRV8825 eigenschappen + datasheet. (Geraadpleegd op 14 september 2019) https://www.pololu.com/product/2133

Video's over PID. (Geraadpleegd op 19 september 2019)

https://www.youtube.com/watch?v=0vqWyramGy8

https://www.youtube.com/watch?v=JFTJ2SS4xyA