

# MRB 2023

Finn Fonteijn, 1745799, [finn.fonteijn@student.hu.nl](mailto:finn.fonteijn@student.hu.nl)

Aug 2023

## Inhoudsopgave

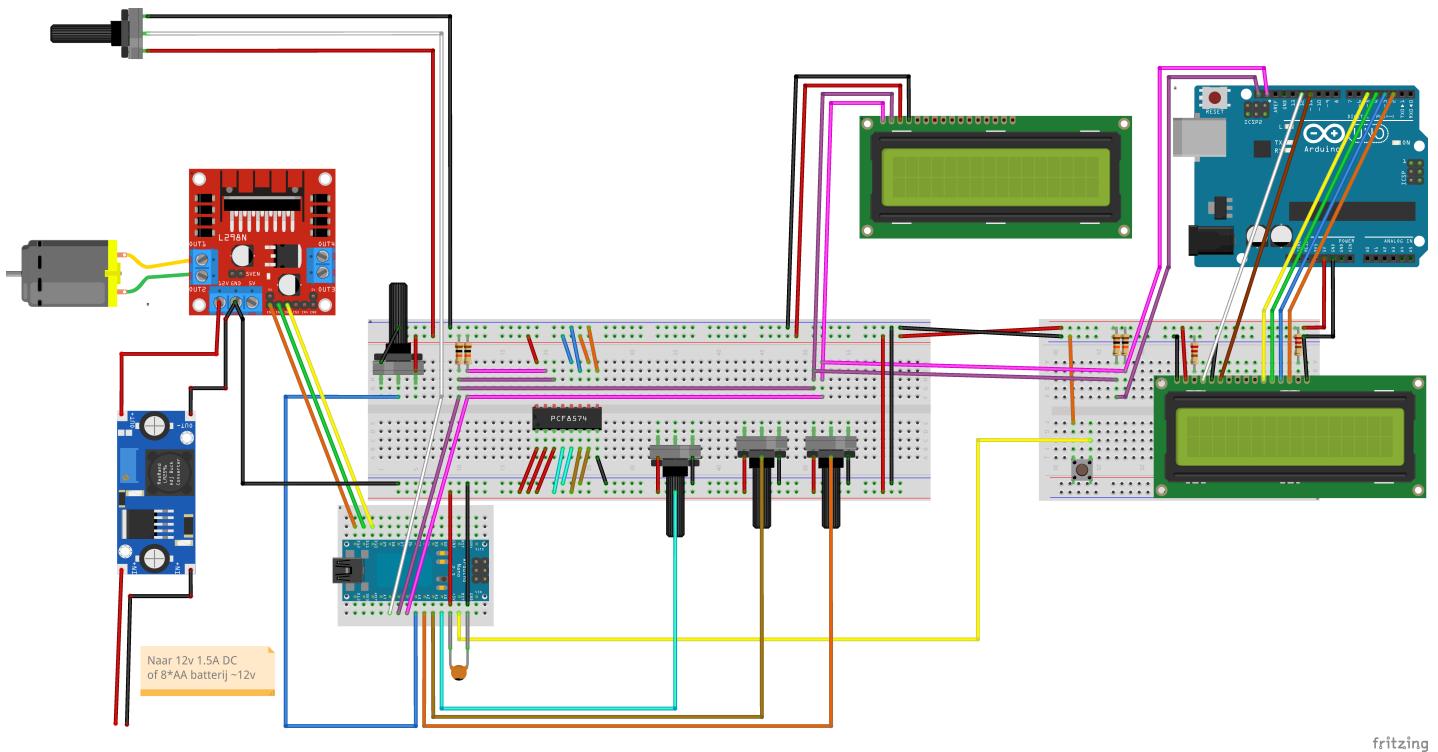
<b>1 Hardware</b>	<b>2</b>
1.1 Fritzing Hardware Diagram . . . . .	2
1.2 Gebruikte Componenten . . . . .	2
1.3 Inspiratie voor mijn Systeem . . . . .	3
1.4 Aerpendulum Constructie . . . . .	3
1.5 Microcontroller Keuze . . . . .	5
1.6 Potentiometers en Andere Sensoren . . . . .	6
1.7 PID-Tuning: Input . . . . .	7
1.8 Displays . . . . .	7
1.9 Motoren en propellors . . . . .	9
1.10 Motor Divers . . . . .	10
1.11 Stoomtoevoer . . . . .	11
<b>2 Software</b>	<b>13</b>
2.1 PID Performance . . . . .	13
2.2 Tunen van Lineaire en Niet-lineaire Uitdagingen . . . . .	13
2.3 Over het Gedrag van een Echt Pendulum . . . . .	13
2.4 Software Ontwerp Totaal . . . . .	13
2.5 AirPendulum . . . . .	14
2.6 LCD 1602 Slave and Master . . . . .	15
2.7 PotManager Class Architecture . . . . .	16
2.8 Digital filtering: . . . . .	16
<b>3 Appendix</b>	<b>19</b>
3.1 Extra Afbeeldingen en Diagrammen (Vergroot) . . . . .	19
3.2 Natuurkundige statica berekeningen voor benodigde kracht motor om staaf te bewegen. . . . .	28

## Videos

- 5 min 05 [Youtube Lange Video](https://www.youtube.com/watch?v=ir9XWVHvrGA) <https://www.youtube.com/watch?v=ir9XWVHvrGA>
- 60 sec [Youtube Korte Video](https://youtube.com/shorts/G_LV0VVTo9M) [https://youtube.com/shorts/G\\_LV0VVTo9M](https://youtube.com/shorts/G_LV0VVTo9M)

## 1 Hardware

## 1.1 Fritzing Hardware Diagram



Figuur 1: Fritzing Hardware Diagram

Kleur en positie van jumperwires op het breadboard zijn aangepast om de leesbaarheid en begrijpelijkheid te verbeteren. De afbeelding is ook vergroot te zien in de appendix en op GitHub (samen met het Fritzing.fzz-bestand).

## 1.2 Gebruikte Componenten

### 1.2.1 Gemodelleerd in Fritzing:

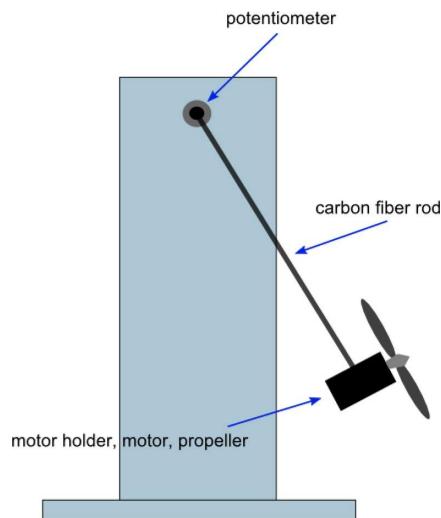
- DC-DC Adjustable Step-down Buck Converter XL4015 4A - with voltage meter
  - L298N DC-Motor Controller (HW-095)
  - NXP PCF8574 IC
  - 1602 LCD scherm (HD44780U) x2 1602 I2C backpack (PCF8574T based)
  - DC Motor
  - Arduino Nano *ik gebruik een Nano-clone met Atmega168 in plaats van 328; het verschil is minder ROM (16k vs 32k)*
  - Arduino Uno
  - 10k $\Omega$  Weerstand x4
  - 220 $\Omega$  Weerstand x2
  - 10k $\Omega$  Potentiometer B10k x5
  - Condensator 22nF x3
  - Drukknop
  - Jumperwires x Veel

### 1.2.2 Extra Benodigde Hardware

- Aluminium Hoek-profiel L-vorm, dikte 2 mm (intern 10x10 mm, extern 12x12 mm) van 100 cm
- Aluminium L-vormige plankdrager 10cm x 10cm (gat uitboren zodat de potentiometer erdoorheen past)
- Propellers die op de DC-motor passen
- Klem voor verf
- Extra lange kabels voor de lengte van het aluminium profiel
- Als contragewichten: M12 verbindingsmoer (40,3g) en/of kubieke centimeter wolfraam (19,25 gram)
- 8 AA-batterijen in houders, in serie
- 12V 1,5A DC-adapter (van oude router, jack verwijderen en jumperwires eraan vast solderen)
- Heel veel elektrische isolatietape

### 1.3 Inspiratie voor mijn Systeem

In tegenstelling tot de andere twee jaar dat ik dit vak heb geprobeerd te halen en deze opdracht heb geprobeerd te maken, wilde ik het dit keer een compleet vers begin geven. Dus geen balletje op een wipwap van K'NEX en een VL6180X. Hierdoor ben ik op zoek gegaan naar iets wat compleet anders werkte. Tijdens het googelen naar ideeën zag ik de afbeelding die de basis vormde voor dit idee:

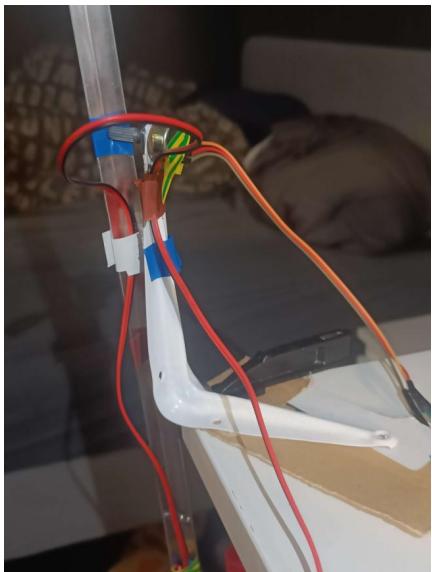


Figuur 2: Inspiratieafbeelding van een Aeropendulum

Het setpoint zou een hoek zijn ergens op deze rotatie, met het extra moeilijke uitdaging om het de pendulum (door de snelheid van de motor van richting te laten veranderen) precies tegenover de normale evenwichts situatie te kunnen balanceren.

### 1.4 Aeropendulum Constructie

De volgende ochtend, na de dag daarvoor online op Amazon/TinyTronics een aantal onderdelen qua elektronica te hebben besteld, heb ik bij de bouwmarkt geen ‘carbon fiber rod’ kunnen vinden. Wel vond ik een aluminium hoekprofiel van 100 cm, dat verwaarloosbare buiging heeft en maar 115 gram woog.



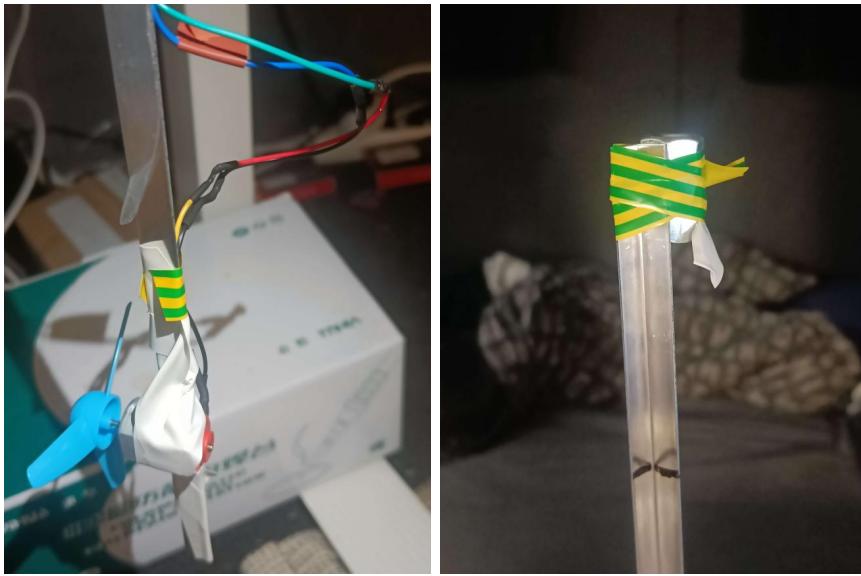
Dit lange profiel zou via een Rotary Encoder of Potentiometer, die als as zou dienen, worden verbonden aan deze boekenplankdrager die met een verfklem aan een tafel werd vastgemaakt. Hiervoor moest eerst het gaatje in de boekenplankdrager worden uitgeboord, zodat de potentiometer er doorheen paste en het moertje deze goed kon vastklemmen, zodat de potentiometer niet kon rondraaien. De spleet die in het draaigedeelte zit van de potentiometer klemt (als je wat isolatietape gebruikt) over het hoekprofiel heen. De DC-motor en bedrading werden met isolatietape vastgemaakt aan het profiel.

Voor de Rotary Encoder maakte dit niet uit, aangezien ze ongelimiteerd beide kanten konden opdraaien. Maar de oriëntatie van deze potentiometer ('WL' B10k ~300 graden totale draaiing) moet met de pootjes ongeveer horizontaal zijn, zodat de staaf maximaal zowel voorwaarts als achterwaarts zou kunnen draaien zonder dat die de uiteinden van de potmeter zou raken.

Na testen bleek dit systeem extreem moeilijk in beweging te krijgen. De motor aan of uit zorgen ervoor dat er nauwelijks beweging in kwam. Door tegengewichten te plaatsen aan de andere kant van het draaipunt. Hiervoor had ik een M12-bout van ~40g en een blokje wolfraam van ~20g; deze kunnen met tape worden bevestigd op verschillende afstanden tot het draaipunt om zo de balans van het systeem te veranderen. De locatie van het draaipunt veranderen. Dus in plaats van twee armen van 50 cm naar 70 cm en 30 cm of zoals de inspiratieafbeelding: één arm van 100 cm.

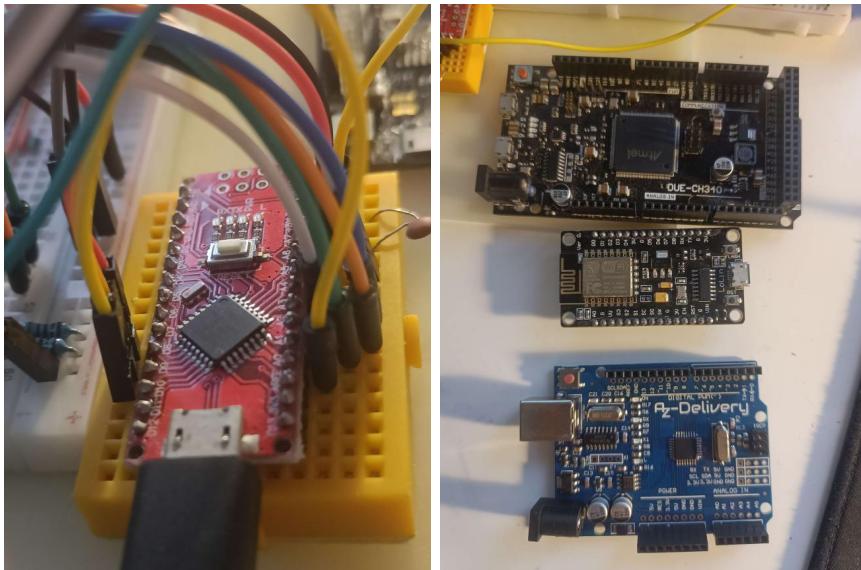
Hoewel we de langste tijd hebben geprobeerd te werken met de 50 cm armen, en de tegenGewichten (waarvan een aan de arm van de motor) zo geïnposeerd dat het zwaartepunt ongeveer hetzelfde was als het draaipunt, zodat er zo min mogelijk weerstand bestond om de opstelling te laten draaien. Dit bleek heel moeilijk te zijn voor het PID-systeem, omdat je door de oscillaties en het momentum dat voorbij 90 graden (profiel is horizontaal) kwam, de PID-controller (in combinatie met de geteste motoren) niet op tijd kon afremmen en hierdoor 'over de kop' ging waardoor de draad naar de motor in de knoop kwam.

Hierdoor hebben we in de laatste week voor een stabiel systeem gekozen, waarbij een arm 70 cm lang is met de motor aan het einde, en alleen de moer van 40g als tegenGewicht. Het nettoresultaat van dit systeem was (als het profiel horizontaal stond) een resulterende kracht van 23,5 gram die de motor naar beneden drukte. De statica/dynamica van deze situatie is verder uitgewerkt (door mijn huisgenoten die gelijk wilden krijgen) als bijlage achter in de Appendix.



Hierdoor kon bij maximale motorvermogen op ongeveer 7.5V (terug gerekend naar de graden van ADC-waarden) ten opzichte van het evenwicht een hoek van -36.2 graden ‘voorwaarts’ en 17.2 graden ‘achterwaarts’ bereikt worden. Dit is een totaal van ongeveer 53,4 graden aan setpoints die we kunnen bereiken. Door momentum kan de hoek wel horizontaal worden, waar het risico van over de kop gaan bestaat, hierdoor is er in de code een ‘emergency stop’ implementeerd die boven een bepaalde hoek het motorvermogen op 0 zet en 5 seconden delay aanhoud zodat het systeem weer tot rust komt.

## 1.5 Microcontroller Keuze



### 1.5.1 ESP8266

Oorspronkelijk begonnen met een ESP8266 NodeMCU omdat ik deze onlangs nog had gebruikt. Dankzij de ingebouwde netwerkverbinding zou het makkelijk zijn om de PID-waarden via een online dashboard te visualiseren. We zijn hier ver mee gekomen, maar het aanpassen van de PID-waarden bleek lastig omdat de ESP8266 maar één ADC heeft. Hierdoor kunnen er niet gemakkelijk meerdere potmeters worden aangesloten. (delen van deze code zijn nog steeds te vinden in de main branch.)

### 1.5.2 Arduino Due

Deze werkte goed totdat deze is doorgebrand. De snelheid was zeker een voordeel; ik meen dat we zonder `Serial.println` een loop-tijd van 7 ms hadden in vergelijking met de huidige 35 ms. Dat deze kapot ging bleek achteraf te komen door een defecte of intern kortgesloten PCF8574, waardoor er stroom(~3v 1.5A) op inputs van de Due werd gezet. Als gevolg hiervan wordt de Due nu alleen nog maar heel heet.

### 1.5.3 Arduino Uno

Deze had ook goed gewerkt, maar ik gebruikte er al een voor het tweede display en had maar een (korte) USB B-kabel.

### 1.5.4 Arduino Nano: Atmega168

De Arduino Nano-clone had vrijwel alle voordelen van de Uno, behalve dat ik hier meer exemplaren(mocht ik er meer laten doorbranden) van had en een lang genoeg USB-kabel, en werd hierdoor de keuze voor de rest van de ontwikkeling.

## 1.6 Potentiometers en Andere Sensoren



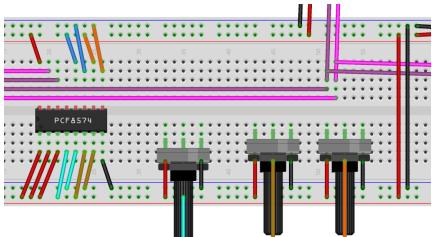
Voordat ik gebruikmaakte van de tegengewichten waren de potmeters niet echt geschikt voor dit systeem. Ze hadden intern te veel weerstand in termen van newtonmeter voor draaiing (niet in ohm) om door onze motor bewogen te worden, die op dat moment ook incorrect werd aangestuurd. Hierdoor heb ik eerst als als van de pendulum een rotary encoder gebruikt, die veel minder interne weerstand heeft. Dit werkte op zich goed, behalve dat de pendulum elke keer tijdens het instellen een kalibratieproces moest ondergaan zodat de rest van de code wist wat de 0-stand equilibrium zou zijn. Soms verloor de encoder ook zijn telling. Het grootste probleem was dat het PID-systeem hier niet goed genoeg kon functioneren. De twee rotary encoders die ik had, hadden slechts 24 posities, waardoor het voor het PID-systeem moeilijk was om precies te weten waar het zich bevond.

Ik had een ‘sub-angle’-systeem in ontwikkeling dat op basis van verandering in snelheid (tijdmeting tussen verschillende rotary encoder-veranderingen) probeerde om te zetten naar een geschatte positie tussen de twee rotary encoder-posities, de ‘sub-angle’. Maar dit was onbegonnen werk. Ook heb ik overwogen om een acceleraometer /gyroscope zoals de MPU6050 of GY-61 (die ik thuis had liggen) toe te voegen aan een van de eindes van het profiel, dit zou helpen met de nauwkeurigheid van de sub-angle, en de gyroscope zorgde er voor dat er niet gewacht moest worden op het 0-moment voor kalibreren. Dit allemaal toegevoeggen zou veel extra werk en complexiteit zijn dus ben ik hierna gaan kijken of ik niet meer koppel/kracht kon genereren om de potentiometer te laten werken. De potentiometer heeft ten opzichte van de 24 posities van de rotary encoder 1023 posities (4095 op de Due) en vereist geen kalibratie.

## 1.7 PID-Tuning: Input

Om de scalars—de  $K_p$ ,  $K_i$  en  $K_d$  waarden in de formule  $Kp \times \text{error} + Ki \times \text{integral} + Kd \times \text{derivative}$ —tijdens runtime aan te passen, heb ik drie extra potentiometers toegevoegd. Er is ook een potentiometer beschikbaar voor het instellen van de setpoint (de doelhoek) van het regelsysteem.

Tijdens het prototypen van scalaradjustment via websockets op de ESP8266 was ik al heel inefficient bezig met het tunen van het regelsysteem. Ik wist dus dat de waarden voor  $K_p$ ,  $K_i$  en  $K_d$  niet simpelweg een vaste combinatie zoals  $K_p=2$ ,  $K_i=3$  en  $K_d=0$  zouden zijn; er zou een significant verschil tussen deze parameters moeten zijn. Om deze aanpassingen tijdens runtime te kunnen maken, heb ik gebruikgemaakt van een NXP PCF8574AP, een 8-bit I/O Expander met i2c-interface, die me 8 extra inputs gaf. Daarvan hebben we er 6 gebruikt: twee voor elke potentiometer. Deze inputs gaven elke potentiometer 2 bits, wat resulteerde in de mogelijkheid om één van de 4 verschillende ‘multipliers’ te selecteren.



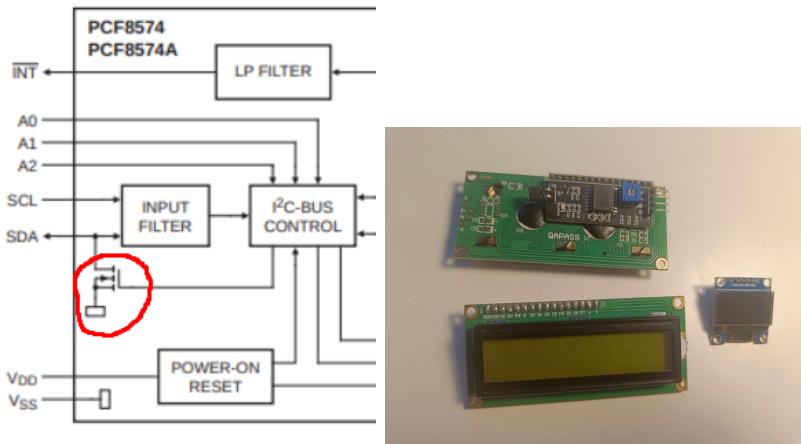
Bijvoorbeeld, in de Fritzing-afbeelding zijn er cyan, bruin en oranje jumperdraden aangesloten op de PCF8574; deze kleuren komen overeen met een van de drie potentiometers die er rechts naast staan. De posities van deze jumperdraden kunnen binair worden uitgedrukt als 00, 01, 10, of 11. Als de code gecompileerd zou zijn met `const int SensitivityMultipliers[] = {1,10,20,50};`, dan zou dit betekenen dat elke parameter ( $K_p$ ,  $K_i$ ,  $K_d$ ) onafhankelijk kan worden aangepast binnen bereiken van 0-1023, 0-10230, 0-20460, of 0-51150.

Deze setup zorgt ervoor dat ik zowel de positie van de potentiometers als de jumperdraden fysiek kan aanpassen, en dat deze instellingen bewaard blijven, zelfs bij stroomuitval of reset van de Arduino. Hierdoor zijn alle inputwaarden voor het PID-systeem (scalars, setpoint en de hoek van het systeem) non-volatile en direct gekoppeld aan een fysieke eigenschap. Dit maakt het mogelijk om relatief deterministisch gedrag te krijgen, vooral nuttig bij het debuggen van ‘underflow’ of ‘overflow’ bugs, mits de pendulum stil staat (d.w.z. de as is ontkoppeld van het profiel).

Alle code voor het instellen van deze 5 waardes is samengevoegd mijn eigen library `src/PotManager.h`.

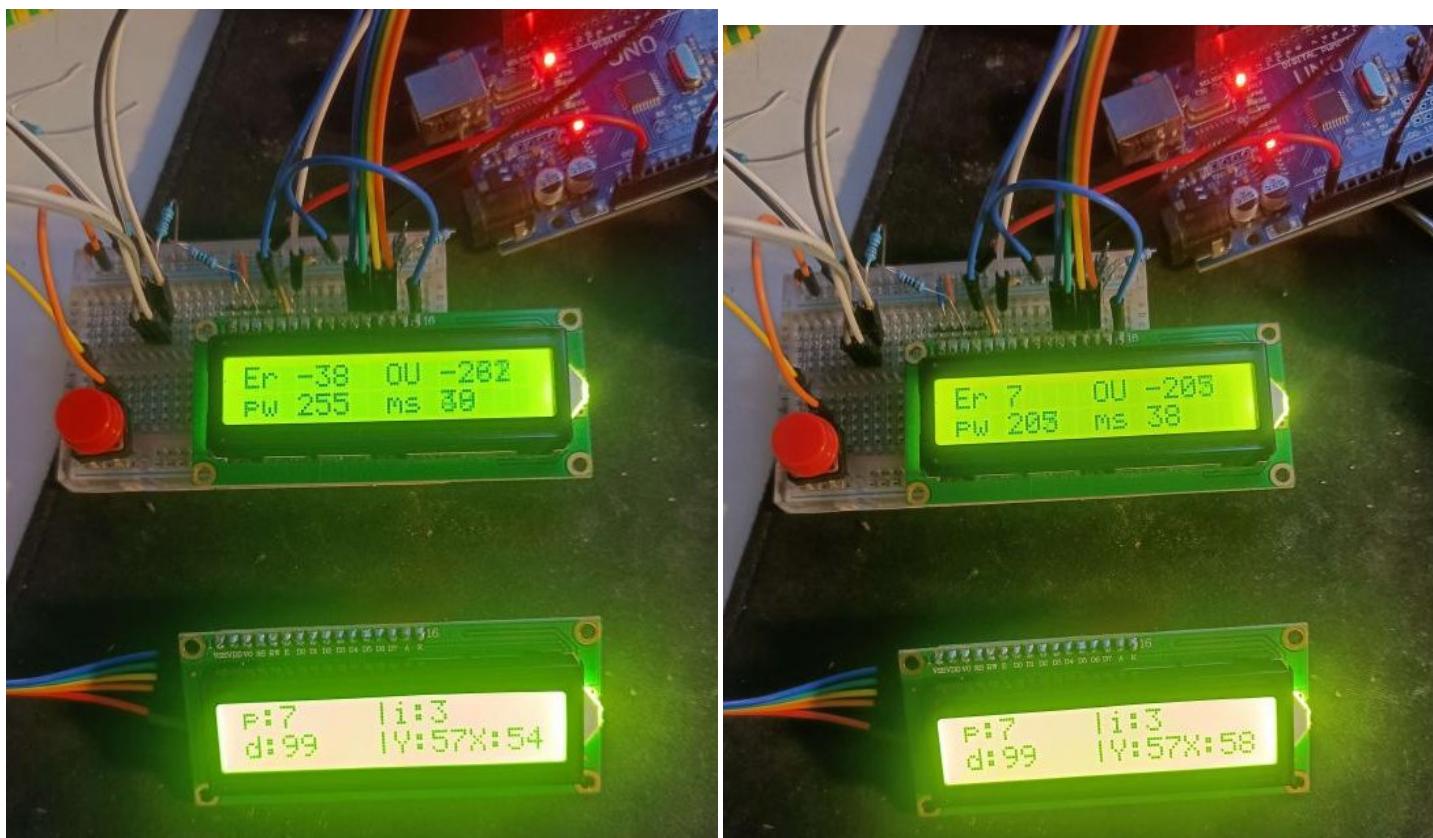
## 1.8 Displays

Aanvankelijk was het mijn bedoeling om via de ESP8266 invoerwaarden te visualiseren op een webpagina met grafieken die waarden over tijd weergeven. Echter, vanwege veranderingen in de hardwarekeuze, ben ik uiteindelijk overgeschakeld op twee 1602 LCD-displays. Dit was ook noodzakelijk omdat mijn SSD1306 OLED-display kapot was. Voor een van de twee 1602-displays heb ik een I2C-‘backpack’ gesoldeerd die een interne PCF8574-chip bevat. Daardoor kon het display via I2C worden aangestuurd, een interface die ik al gebruikte voor mijn eigen I/O-uitbreiding, en dus perfect uit kwam.



Ondanks dat de PCF8574A-chips interne pull-upweerstanden hebben, ondervond ik problemen bij het aansluiten van een tweede apparaat op dezelfde I2C-bus. Daarom heb ik zowel de SDA- als de SCL-lijnen met een 10k ohm weerstand naar de 5V getrokken. Daarmee werd een stabiele communicatie met beide apparaten mogelijk.

Het 1602-display biedt 32 karakters, wat voldoende was om de invoerparameters voor het PID-systeem weer te geven. Om echter ook inzicht te krijgen in de interne werking en outputwaarden van het systeem, was een tweede 1602-display noodzakelijk. Om niet tegelijkertijd de libraries `#include <LiquidCrystal_I2C.h>` en `<LiquidCrystal.h>` te hoeven gebruiken, heb ik ervoor gekozen om het tweede display ook via I2C aan te sturen. Dit werd gerealiseerd met behulp van een tweede Arduino Uno die als I2C-slave functioneert. Deze Arduino toont waarden die het via de I2C-bus ontvangt op het display.



De zelfgeschreven code voor deze I2C-slave is te vinden in `LCD1602Slave/LCD1602Slave.ino`. De master kan dit aansturen met `LCD1602SlaveController.h`, een meer gestroomlijnde library die alleen de benodigde `char*` of `ints` doorstuurt om te worden weergegeven op het tweede display. Net als de eerste I2C-verbinding, is ook deze tweede verbinding uitgerust met twee 10k pull-upweerstanden. Op ditzelfde breadboard is een push button knop die de `RESET` pin van de Arduino naar Ground trekt, dit was handig als de arduino vast liep terwijl het

motorvermogen aan stond op dat moment.

De getoonde waardes op de displays bestaat uit: (en de range van mogelijk waardes die ze kunnen laten zien, zowel als het corresponderende interne datatype)

#### 1.8.1 Display PotManager:

- p:0-65,535: Dit is de Kp waarde voor het PID regelsysteem, uint16\_t.
- i:0-65,535: Dit vertegenwoordigt de Ki waarde, uint16\_t.
- d:0-65,535: Hier wordt de Kd waarde getoond, uint16\_t
- Y:0-99: Dit is de actuele positie van de pendulum. Hoewel het display slechts waarden tussen 0 en 99 toont, wordt intern een nauwkeuriger 10-bits meting (0-1023) gebruikt.
- X:0-99: Dit is het ingestelde setpoint voor het PID-systeem. Net als bij de pendulumpositie wordt intern een 10-bits waarde (0-1023) gebruikt voor meer nauwkeurigheid.

#### 1.8.2 Display i2cSlave:

- Err: -999-9999: Dit is het verschil tussen de gelezen ADC-waarden van het setpoint en de positie van de pendulum. Deze waarde is ongeveer  $(X-Y) \times 10.2$  en wordt als een signed int opgeslagen.
- Out: -999-9999: Dit is het resultaat van de PID-berekening, oftewel de gewenste output van het regelsysteem. Intern wordt dit opgeslagen als een signed long.
- pw: 0-255: Dit geeft de duty cycle van het PWM-signal aan de motor aan, uitgedrukt in een 8-bits getal.
- ms: 0-9999: Dit toont de tijd in milliseconden die elke systeemlus duurt. Intern wordt dit opgeslagen als een long.

Door deze waarden op de displays te tonen, krijk ik een gedetailleerd en real-time overzicht van het systeem, wat waardevol is voor zowel diagnose als het tunen van het PID systeem.

### 1.9 Motoren en propellers

Voor dit project had ik de keuze uit vier verschillende motoren en drie soorten propellers. Hier volgt een gedetailleerde beschrijving van mijn bevindingen en uiteindelijke keuzes.

#### 1.9.1 Soorten Motoren:

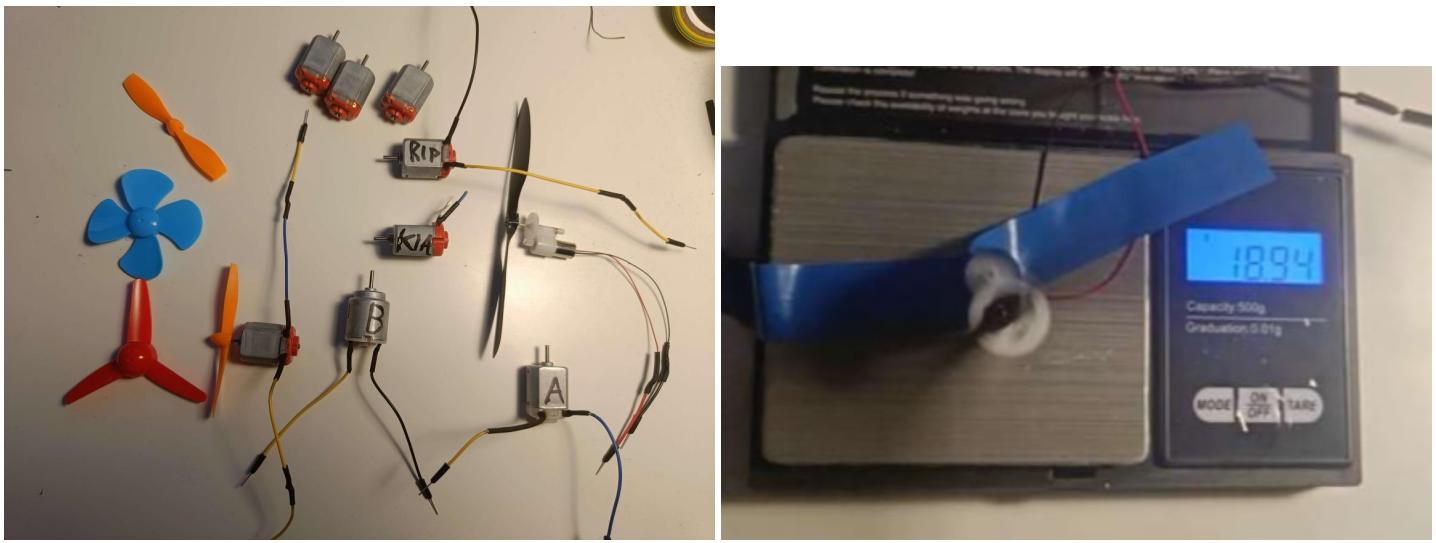
- **Motor A:** Gewicht is 14g.
- **Motor B:** Gewicht is 15g.
- **Motor R** (met een rode achterkant): Gewicht is 17g. Ik had 10 stuks van deze motor.

#### 1.9.2 Selectern van Propeller:

Ik heb de propellers getest met Motor R bij een spanning van 5.5V. Door de motor omgekeerd aan een weegschaal te bevestigen, kon ik de stuwwerkt als een negatief getal meten:

- **2-blads:** -1.8g
- **3-blads:** -2.3g
- **4-blads:** -1.9g

Op basis van deze resultaten(niet opgemeten maar de 3-blad propeller leek het ook beter te doen op de andere 2 motor types) koos ik voor de 3-blads propeller vanwege zijn hogere stuwwerkt in deze RPM-range.



### 1.9.3 Thrust metingen:

- **Motor A :** Produceerde -2.9g met de 3-blads propeller.
- **Motor B:** Produceerde -1.1g (onbruikbaar vanwege excessieve vibratie en lage snelheid).
- **Motor R:** Motor R: -2.3g met de 3-blads propeller.

### 1.9.4 Speciale Overweging - 8520 Drone Motor:

Ik heb ook een 8520 drone-motor gekocht om de draaiweerstand van de potentimeters te overwinnen. Deze motor weegt slechts 8g (inclusief versnellingsbak en een 15cm propeller) en produceert meer dan 20g stuwwerk. Hoewel indrukwekkend, bleek deze motor niet geschikt voor mijn PID-systeem. Het reageerde pas bij een PWM duty cycle van ongeveer 50% en vertoonde daarna een niet-lineaire respons. De andere motoren begonnen mescal met heel langzaam te draaien vanaf ~15% duty cycle.

### 1.9.5 Duurzaamheid en Eindkeuze:

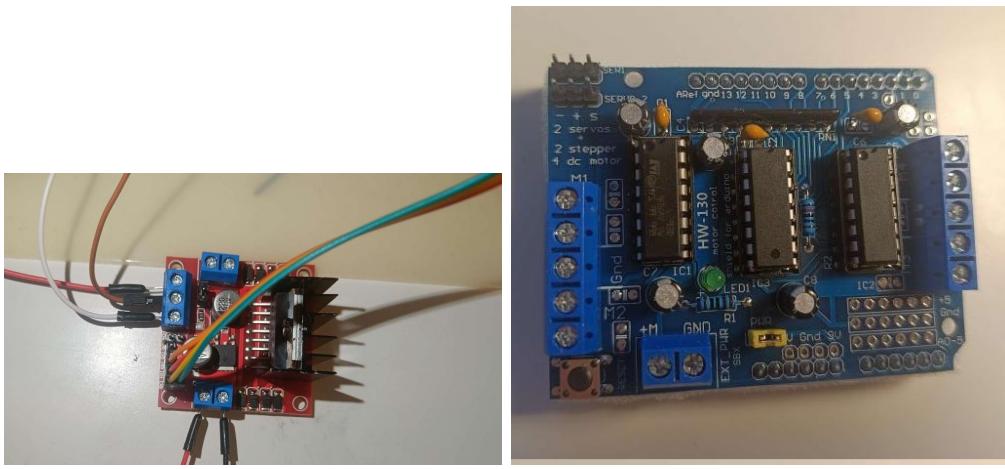
Bij langdurige en frequente richtingsveranderingen tijdens PID-tests kunnen motoren beschadigd raken, vooral hun koolborstels. Hoewel Motor A lichter is en meer stuwwerk biedt, heb ik gekozen voor Motor R vanwege de beschikbaarheid en vervangbaarheid. Gedurende de tests heb ik vier Motor R-eenheden gebruikt, waarvan er twee uiteindelijk defect raakten en één nu stroef loopt.

## 1.10 Motor Divers

Ik heb de enige twee DC motordrivers aangeschaft op TinyTronics dit waren de L298N en de L293D.

De L298N vereist een spanning van 5-35V DC voor de motoren en kan maximaal 2A leveren voor elk van zijn twee uitgangen. Daarentegen heeft de L293D een spanning nodig van 5-12V DC en kan slechts maximaal 600mA per uitgang leveren. Omdat onze besturingskaart twee van deze L293D-drivers bevatte, in combinatie met een shiftregister, kon het in totaal drie motoren aansturen.

Vanwege de beperkte stroomcapaciteit van de L293D koos ik voor de L298N, die een maximale output van 2A per uitgang kan leveren. Een uitdaging met de L298N was de merkbare spanningsval van ongeveer 2V. Onze oorspronkelijke voedingsbron van een 5V 2A USB-voeding bleek onvoldoende, waardoor we moesten overstappen op een voeding met een hogere spanning.



Wat betreft de controle voor de driver heeft de L298N drie belangrijke pinnen voor elke motoruitgang: ENable, IN\_1, en IN\_2. Door een PWM-signalen naar de EN-pin te sturen, kun je de snelheid van de motor regelen. De IN\_1 en IN\_2 pinnen bepalen vervolgens de draairichting. Ik had achteraf een defecte chip of PCB waarbij de IN\_2 input van Output A niet was aangesloten intern waardoor de PWM-besturing slechts in één richting werkte voor motor A. Dit probleem werd (na enige tijd en veel frustratie) opgelost door simpelweg de andere motoruitgang te gebruiken.

Tijdens dit proces hebben we ook de L293D motorshield getest, maar deze kon zoals verwacht niet voldoende stroom leveren. Ik heb ook overwogen om MOSFETs en condensatoren te gebruiken om zo zelf mijn motor te driven, maar deze optie bleek inefficiënt en beperkt in functionaliteit omdat we ook PNP-transistors nodig zouden hebben voor een volledige H-bridge, die nodig is om de motor twee kanten op te kunnen laten draaien.

## 1.11 Stoomtoevoer

### 1.11.1 Verhogen Spanning

Vanwege de merkbare spanningsval van ongeveer 2V bij de L298N heb ik nu een minimale voedingsspanning van 7V nodig. Aanvankelijk gebruikten ik een serie van acht AA-batterijen(13.5v-10.5v). Hoewel dit werkte, liepen de batterijen snel leeg vanwege de hoge stroom die de motor verbruikte. Deze exacte DC motor-stroomwaarde kon ik nooit echt meten omdat mijn multimeter een maximale capaciteit van 500mA heeft en ik geen geschikte shuntweerstand(<10 Ohm) had.

Om dit probleem op te lossen, schakelden we over naar een step-down buck-converter (XL4015) met een capaciteit van 4A en een ingebouwd display. Dit stelde ons in staat de in- en uitgangsspanningen te monitoren en aan te passen met een potentiometer. Om een stabiele uitgangsspanning van 5,5V te behouden, was nu een ingangsspanning van 9V vereist. Na het verbruiken van 16 AA-batterijen, vonden we een oude 12V, 1,5A DC-adapter van een router. We verwijderden de stekker en soldeerden er jumperdraden aan.

### 1.11.2 Elektrickery

Tijdens het gehele PID proces zijn we diverse problemen tegengekomen, waaronder het onstabiel functioneren van de Arduino-boards en tegelijkertijd knipperen van mijn computermonitor wanneer ik de arduino reset, waarschijnlijk wegen raare groundloops en het feit dat die op dezelfde elektrische groep was aangesloten. Om deze problemen te verhelpen, hebben we eerst ontkoppelingscondensatoren van 100 pF toegevoegd, bedoeld voor hogere frequenties. Dit bleek niet effectief, waarna we voor lagere frequenties condensatoren van 22nF toevoegden, die wel leken te helpen met stabiliteit.

Ten slotte hebben we ervoor gekozen om het gehele PID-systeem — inclusief displays en invoerapparaten — te laten functioneren zonder USB- of pc-verbinding om de storingsgevoeligheid te minimaliseren. Deze aanpassing bleek succesvol.



### 1.11.3 USB-Stroomproblemen

Tijdens het instellen van de PID-waarden ging onze PCF8574-chip defect. Dit resulteerde in een kortsleuteling die de Arduino (aangesloten via USB op mijn computer) ertoe aanzette om tijdelijk een stroom van ~5.15V en 1.4A te trekken. Mijn computer reageerde hierop door alle USB-apparaten tijdelijk los te koppelen om overbelasting te voorkomen, zoals bleek uit de kernel-logbestanden.

```
Aug 22 20:19:04.564401 fenrir kernel: usb usb2-port1: over-current condition
Aug 22 20:19:04.587431 fenrir kernel: usb usb1-port1: over-current condition
Aug 22 20:19:04.587641 fenrir kernel: usb 1-1: USB disconnect, device number 2
Aug 22 20:19:04.587672 fenrir kernel: usb 2-1: USB disconnect, device number 7
```

Dit probleem was moeilijk te diagnosticeren totdat we een USB-spanning- en stroommeter gingen gebruiken, gekoppeld aan een powerbank om te voorkomen dat de Arduino uitviel. We hebben toen de bedrading een voor een losgemaakt om het probleem te isoleren. Na het oplossen van dit issue hebben we de schaalfactoren voor de PID-regeling handmatig tijdens de compilatietaid ingevuld, zodat deze niet meer aanpasbaar waren tijdens de uitvoering.

De combinatie van hardwareaanpassingen en de loskoppeling van de PID-opstelling van de USB- en pc-verbindingen heeft bijgedragen aan een aanzienlijk stabieler en betrouwbaarder systeem.

## 2 Software

### 2.1 PID Performance

Met de PID-waarden van  $K_p=10$  (bereik 6-12),  $K_i=4$  (bereik 1-5) en  $K_d=90$  (bereik 85-98) kan het systeem snel (binnen circa 1 seconde) en nauwkeurig een hoek tussen 36.2 en -17.2 graden ten opzichte van het evenwicht (0.0 graden) bereiken. Dit gebeurt met minimale overshoot en bounce/oscillatie. De grootste uitdaging voor het PID-systeem—vooral de ‘D’-term—zit in de transitie van het ene uiteinde van het bereik naar het andere (van 36 naar -17 graden). Dit komt omdat het systeem dan moet compenseren voor zowel het gewicht en het momentum als voor de P- en I-termen, die in deze fase geen corrigerende werking hebben.

Hoewel mijn oorspronkelijke plan om een pendulum op 180 graden van zijn evenwicht te balanceren niet is geslaagd, geloof ik dat dit met de huidige hardware onmogelijk is zonder aanpassingen, zoals bijvoorbeeld een tweede motor. Desondanks functioneert het PID-systeem nog steeds goed met dezelfde scalars, zelfs als ik de Savitzky-Golay-filtering uitschakel.

### 2.2 Tunen van Lineaire en Niet-lineaire Uitdagingen

Het afstellen was een tijdrovend proces, voornamelijk vanwege aanhoudende hardwareproblemen en bugs. De Inputs en opzet van het systeem om mee te tunen van het systeem heel goed werkte, werd bemoeilijkt nadat de PCF8574 defect raakte. Het complete systeem bleek bij het zwaartepunt en draaipunt (~50 cm) extreem instabiel te zijn. De motor was simpelweg niet krachtig genoeg om het natuurlijke momentum effectief te compenseren.

Traditionele methoden voor PID-tuning, zoals de b.v. Ziegler–Nichols-methode, bleken niet behulpzaam. Deze methoden beginnen vaak met  $K_i$  en  $K_d$  op nul en zijn ontworpen om een oscillatie in het systeem te creëren via de  $K_p$ -term, om zo de ultieme versterking  $K_u$  te vinden. In de context van ons pendulum was dit problematisch. Zonder een significante demping ( $K_d$ ) aan het begin, was het vrijwel onmogelijk om de output te laten oscilleren zonder dat de pendulum over de kop ging of de limieten van de potentiometer bereikte. De huidige waardes zijn het resultaat van uren lang trial en error, als er niks in die tijd kapot ging.

Pas nadat ik de pendulum zo veel mogelijk liet gedragen als een eenvoudig lineair systeem—door het minimaliseren van niet-lineaire effecten—werd(zwaam maken aan een kant, draaipunt verschuiven van 50cm naar 70cm etc.) werd het tunen enigszins beheersbaar. Dit brengt ons bij het fundamentele probleem van het tunen van een systeem dat in essentie niet-lineair is.

### 2.3 Over het Gedrag van een Echt Pendulum

Een pendulum met echte massa en momentum gedraagt zich bij grote hoekafwijkingen niet als een simpel lineair systeem. De relatief makkelijke formule  $m \cdot g \cdot \theta$  is dan niet meer voldoende om het gedrag te beschrijven. Dit komt omdat niet-lineaire effecten, zoals de zwaartekracht en de aerodynamische krachten van de propeller, een steeds grotere rol gaan spelen. Vooral wanneer er een motor of propeller aan het uiteinde van de pendulumstaaf is gemonteerd, wordt het gedrag van het systeem sterk beïnvloed door de dynamische interactie tussen de aerodynamische krachten en de mechanische eigenschappen van het pendulum. Dit maakt het des te uitdagender om het systeem effectief te regelen met een eenvoudige PID-regelaar.

Dus, het echte obstakel in het tunen was het inherent niet-lineaire gedrag van de pendulum, wat traditionele lineaire PID-tuning methoden inadequaat maakte voor deze toepassing. Het vereiste aanpassingen en improvisaties om het systeem in een zo lineair mogelijke staat te brengen voor effectieve regeling.

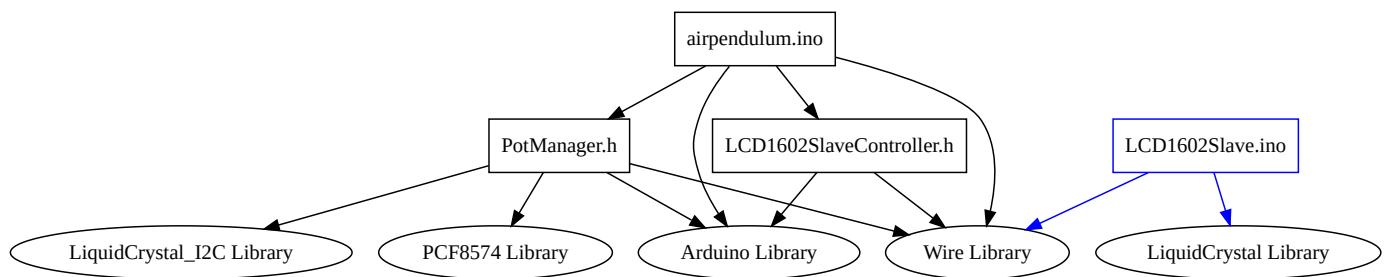
### 2.4 Software Ontwerp Totaal

Het onderstaande diagram illustreert de relaties tussen de requirements van elk bestand. Hierin zijn de externe arduino libraries `Wire`, `LiquidCrystal` en `LiquidCrystal_I2C` opgenomen om hun belang in de structuur van het systeem aan te geven. Voor de rest bestaat de software uit 3 delen:

1. LCD1602 Slave Controller De bestanden `LCD1602SlaveController.cpp` en `LCD1602SlaveController.h` vormen het LCD1602 Slave Controller gedeelte van het systeem. Deze hebben een afhankelijkheid van de

Wire-bibliotheek voor I2C-communicatie. Deze module fungeert als een tussenpersoon voor de aansturing van een LCD1602 display via de I2C-interface.

- Het bestand *LCD1602Slave.ino* wordt beschouwd als een afzonderlijk apparaat dat alleen is verbonden via de I2C-interface. Daarom is het in een andere kleur opgenomen in de diagram. Deze slave fungeert als een zelfstandige eenheid die communiceert met de hoofdeenheid via I2C.
2. PotManager De bestanden PotManager.cpp en PotManager.h beheren de potentiometerinvoer. Ze zijn afhankelijk van meerdere bibliotheken: Wire voor I2C-communicatie, LiquidCrystal\_I2C voor de interface met het LCD-display, PCF8574 voor extra I/O-mogelijkheden en de standaard Arduino-bibliotheek voor algemene functionaliteiten. Deze module zorgt voor de afhandeling en eventuele schaling van analoge invoer van verschillende potentiometers.
  3. AirPendulum Het bestand airpendulum.ino is het hoofdscript van het project. Dit script heeft afhankelijkheden van zowel LCD1602SlaveController.h als PotManager.h en maakt ook gebruik van de Wire en Arduino bibliotheken voor diverse taken. Dit is het centrale punt waar alle modules samenkommen en waar de hoofdlogica van het project is geïmplementeerd.



Figuur 3: Relaties Tussen de verschillende onderdelen van de software

## 2.5 AirPendulum

Het bestand airpendulum.ino is het hoofdscript van het project. Dit script heeft afhankelijkheden van zowel LCD1602SlaveController.h als PotManager.h en maakt ook gebruik van de Wire en Arduino bibliotheken voor diverse taken. Dit is het centrale punt waar alle modules samenkommen en waar de PID Controller is geïmplementeerd.

### 2.5.1 PID loop

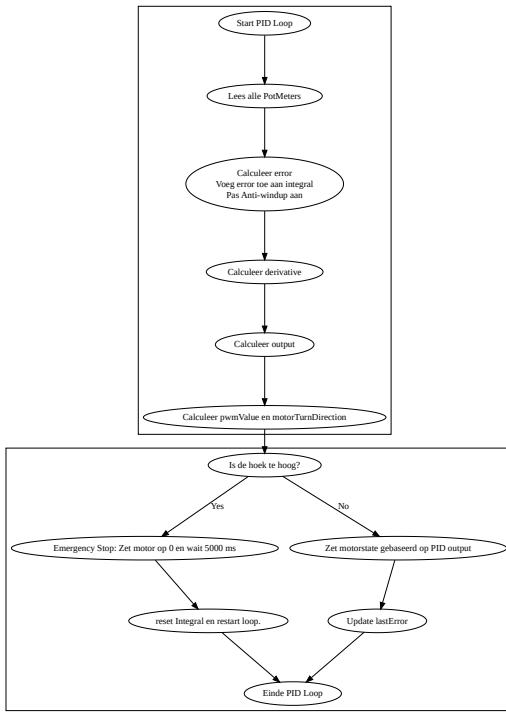
Bestaat uit :

- Proportionele Term ( $K_p * \text{error}$ ): Deze term probeert de huidige fout te verminderen door een correctie te maken die evenredig is met de huidige fout.
- Integrale Term ( $K_i * \text{integral}$ ): Deze term probeert de cumulatieve fout over tijd te verminderen.
- Derivatieve Term ( $K_d * \text{derivative}$ ): Deze term probeert toekomstige fouten te voorspellen en te corrigeren door naar de verandering van de fout te kijken.

De som van deze drie termen vormt de output van de PID-regelaar:

```
long output = (Kp * error + Ki * integral + Kd * derivative) / 10; // Schaal factor van 10
```

**2.5.1.1 Integral Windup en Anti-windup** In PID-regeling is de integral term verantwoordelijk voor het verminderen van de constante fout die overblijft nadat de proportionele term zijn werk heeft gedaan. Echter, zonder een soort van beperking kan deze integral term uit de hand lopen, vooral bij een grote of aanhoudende fout. Dit fenomeen staat bekend als “integral windup”.



Figuur 4: FlowChart van de main Arduino airpendulum PID loopDiagram of the Main Arduino Sketch Workflow

Om te voorkomen dat de integral term onbeheersbaar wordt, worden boven- en ondergrenzen ingesteld. Dit is wat de regels `if (integral >/< MIN/MAX_INTEGRAL) integral = MAX_INTEGRAL;` Ze zorgen ervoor dat integral binnen een vooraf bepaald bereik blijft. Dit wordt ook wel anti-windup genoemd en het helpt bij het stabiel houden van het systeem. Zo ontstaan er geen over or underflows als het systeem lange tijd niet dichter bij zijn setpoint kan komen.

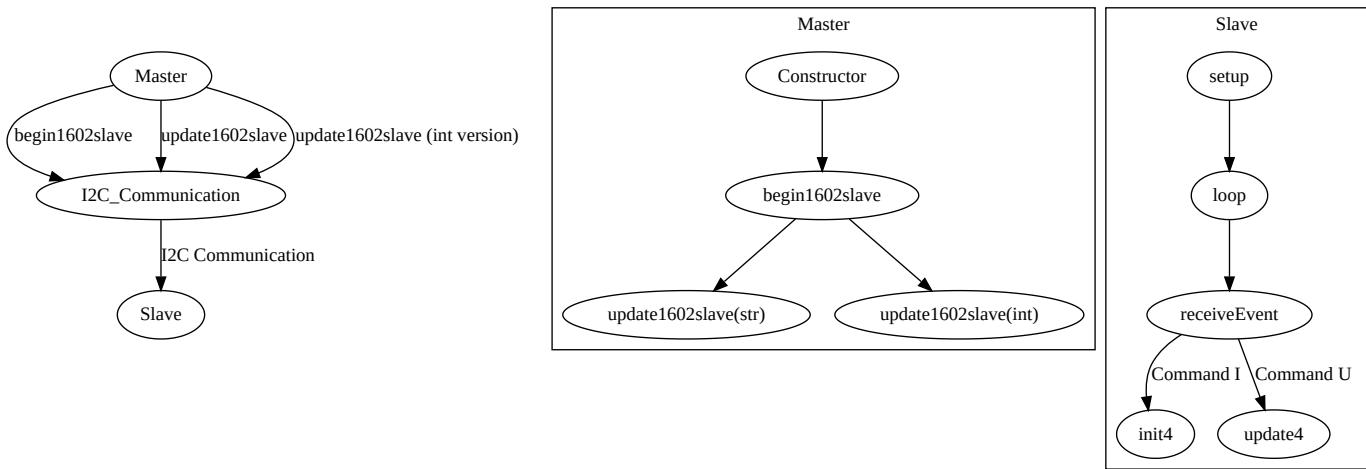
**2.5.1.2 Schaal Factor** Zoals te zien aan het einde waarat wij **output** berekenen, delepen wij deze waarde door 10. Dit is een schaalfactor die helpt om de grootte van de output te beheren zonder floating-point berekeningen te gebruiken. Omdat dit script integers gebruikt in plaats van floating-point getallen, kan het veel efficiënter zijn qua geheugen en verwerkingssnelheid. Dit is van cruciaal belang voor systemen met beperkte resources zoals een Arduino.

**2.5.1.3** Ondanks dat we Millis() en elapsed time gebruiken voor het berekenen van de loopTime, is in dit specifieke geval gekozen voor een eenvoudige PID-regelaar zonder de dt (delta tijd) term. Dit is voldoende, en verder hadden we veel vaker bug en undefined behavior op de arduino Nano/Uno bij gebruik van deze dt manier van PID controll. Het weglaten van dt vereenvoudigt de berekeningen en maakt het gebruik van integers, en zoals eerder gezegd, gunstig is voor beperkte systemen zoals een Arduino.

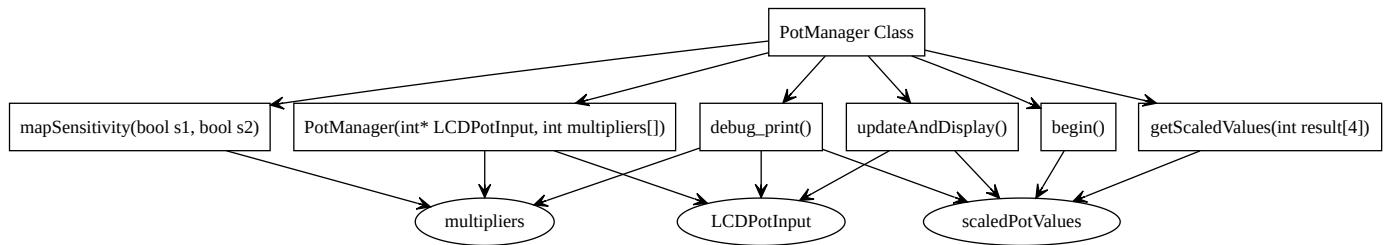
Door deze drie elementen—anti-windup, schaalfactor en het weglaten van dt—kunnen we een efficiënte en effectieve PID-regeling implementeren die geschikt is voor embedded devices.

## 2.6 LCD 1602 Slave and Master

Het block diagram illustreert een eigen implementatie van I2C-communicatie tussen een Master en een Slave voor het beheren van LCD 1602 displays. In de Slave-component wordt de receiveEvent methode getriggerd door I2C-communicatie, wat leidt tot de uitvoering van óf init4 óf update4, afhankelijk van het ontvangen commando. Het doel van deze architectuur is om een tweede LCD 1602 display te kunnen aansturen via de bestaande I2C-verbinding.



Figuur 5: Block Diagram van de LCD1602 Slave en LCD1602Master (LCD1602SlaveController.h)



Figuur 6: Block Diagram van de PotManager Class

## 2.7 PotManager Class Architecture

### 2.8 Digital filtering:

Voor de digitale filtertechniek heb ik gekozen voor Savitzky-Golay. Grotendeels omdat ik voor mijn scriptie ook te maken heb gehad met filtering van ruis op runtime-constrained real-time systemen met een lage sensor poll rate. (In dat geval ging het over het ‘smoothen’ van helderheidsregio’s uit videobeelden; de frametijd daar was rond de 33 ms(30fps), dus vergelijkbaar met dit systeem.) Savitzky-Golay werkt met twee parameters: de `windowSize` en de `polynomial`. De `windowSize` is simpelweg de hoeveelheid datapunten die gebruikt wordt voor het filteren. Door de verhouding tussen `windowSize` en de `polynomial` te veranderen, kun je meer of minder ‘smoothing’ toepassen. Een voordeel van Savitzky-Golay over bijvoorbeeld een moving average is dat er geen signaalresolutie verloren gaat bij het verwijderen van ruis.

- Door te kiezen voor een polynoom van orde 0 hebben we eigenlijk gewoon een moving average filter.
- Bij een polynoom van orde 1 wordt een lineaire regressie toegepast over de `windowSize`.
- Vanaf een polynomiaal van orde 2-3 (in combinatie met hogere `windowSizes`) begint het filter heel goed te werken om veel ‘noise’ te verwijderen.

Echter, voor een polynomiaal van orde 2 of hoger, wordt de berekening een stuk complexer. Boven de 2e orde wordt het onhaalbaar om deze berekeningen (met floating-point precisie) snel genoeg uit te voeren op embedded hardware, zonder dat dit de poll rate (en dus de effectiviteit van het PID-systeem) naar beneden brengt.

Daarom heb ik een Savitzky-Golay filter geïmplementeerd in C++ voor Arduino, dat alleen met integers werkt voor een polynomiaal van 0, 1, en 2. Dit is voldoende voor deze PID-use-case, omdat kleine afrondingsfouten door het systeem zelf kunnen worden opgevangen.

Hierdoor werd zelfs mijn non-float code eigenlijk te langzaam om te gebruiken in mijn PID-setup(150ms+ loop-time). Dit bleek  $\pm 600$  instructies voor een testcase met `polynomial = 2`, `windowSize = 7`. (zie appendix).

Om de performance te verbeteren, heb ik coëfficiënten vooraf berekend voor een polynomiaal van orde 2 en

```

8 if(polynomial == 2){
9     long sumY = 0;
10    long sumX = 0;
11    long sumX2 = 0;
12    long sumX3 = 0;
13    long sumX4 = 0;
14    long sumXY = 0;
15    long sumX2Y = 0;
16
17    for (uint8_t i = 0; i < WINDOW_SIZE; i++) {
18        sumY += SGdata[i];
19        sumX += i;
20        sumX2 += i * i;
21        sumX3 += i * i * i;
22        sumX4 += i * i * i * i;
23        sumXY += i * SGdata[i];
24        sumX2Y += i * i * SGdata[i];
25    }
26
27    long D = WINDOW_SIZE * (sumX2 * sumX4 - sumX3 * sumX3)
28        - sumX * (sumX * sumX4 - sumX2 * sumX3)
29        + sumX2 * (sumX * sumX3 - sumX2 * sumX2);
30
31    if (D == 0) {
32        return 0; // div 0 error-
33    }
34
35    long D0 = sumY * (sumX2 * sumX4 - sumX3 * sumX3)
36        - sumX * (sumXY * sumX4 - sumX3 * sumX2Y)
37        + sumX2 * (sumXY * sumX3 - sumX2 * sumX2Y);
38
39    long D1 = WINDOW_SIZE * (sumXY * sumX4 - sumX3 * sumX2Y)
40        - sumY * (sumX * sumX4 - sumX3 * sumX2)
41        + sumX2 * (sumX * sumX2Y - sumXY * sumX2);
42
43    long D2 = WINDOW_SIZE * (sumX2 * sumX2Y - sumXY * sumX3)
44        - sumX * (sumX * sumX2Y - sumY * sumX3)
45        + sumY * (sumX * sumXY - sumX2 * sumX2);
46
47    long a0 = D0 / D;
48    long a1 = D1 / D;
49    long a2 = D2 / D;

```

Figuur 7: Deel C++ code voor second order Intgers Sav-gol

oneven windowSizes t/m 11. Dit maakte het mogelijk de berekeningen in slechts 60 instructies uit te voeren (voor die testcase zie appendix), wat de loop snelheid verbeterde naar 29-38 ms.

```

sam3xdue handin python3 SavGolCoefficientFinderArduino.py
For window size 5, coefficients are [-86, 343, 486, 343, -86] with a coefficientDivider of 1344
At least one coefficient is negative, requiring a signed data type.
The highest coefficient multiplied by 1023 is 497178
This would fit in a uint32_t data type on Arduino.

For window size 7, coefficients are [-95, 143, 286, 333, 286, 143, -95] with a coefficientDivider of 1381
At least one coefficient is negative, requiring a signed data type.
The highest coefficient multiplied by 1023 is 340659
This would fit in a uint32_t data type on Arduino.

For window size 9, coefficients are [-91, 61, 169, 234, 255, 234, 169, 61, -91] with a coefficientDivider of 1365
At least one coefficient is negative, requiring a signed data type.
The highest coefficient multiplied by 1023 is 260865
This would fit in a uint32_t data type on Arduino.

For window size 11, coefficients are [-84, 21, 103, 161, 196, 207, 196, 161, 103, 21, -84] with a coefficientDivider of 1337
At least one coefficient is negative, requiring a signed data type.
The highest coefficient multiplied by 1023 is 211761
This would fit in a uint32_t data type on Arduino.

For window size 13, coefficients are [-11, 0, 9, 16, 21, 24, 25, 24, 21, 16, 9, 0, -11] with a coefficientDivider of 187
At least one coefficient is negative, requiring a signed data type.
The highest coefficient multiplied by 1023 is 25575
This would fit in a signed int data type on Arduino.

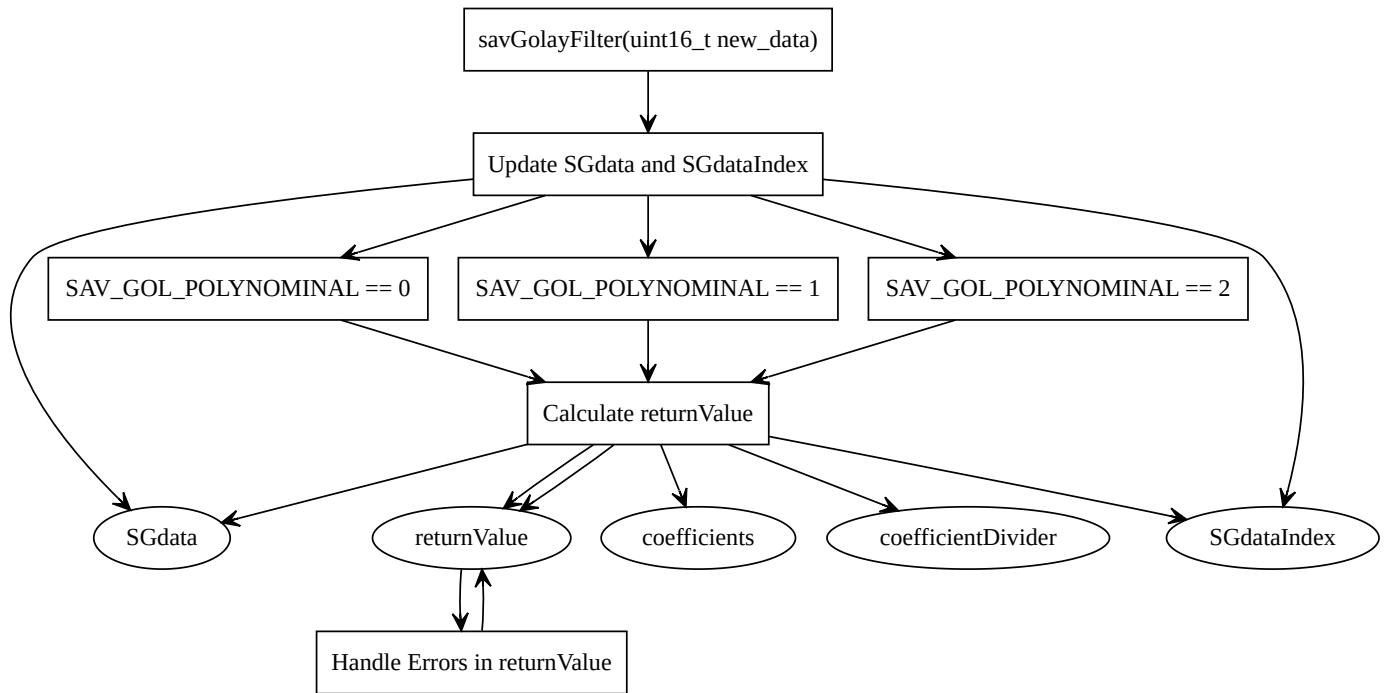
```

Figuur 8: Python-tool output voor coëfficiënten en aangeraden datatypes

Met behulp van een Python-tool,`src/SavGolCoefficientFinderArduino.py`, heb ik de coëfficiënten vooraf berekend. Deze coëfficiënten worden dan geschaald naar gehele getallen om efficiënt te kunnen werken op de hardware. Dankzij deze aanpak maakt het mogelijk om snel en nauwkeurig te filteren, zelfs op minder krachtige microcontrollers.

NB 25/08 : Ik heb hier kort mee geëxperimenteerd tijdens mijn scriptie (precompilatie van Savitzky-Golay-coëfficiënten, uiteindelijk niet nodig door gebruik van een andere library, alleen stukje python-tool code), maar dat was meer in theorie. Ik heb dit pas in de afgelopen maand geïmplementeerd; achteraf misschien een paar uur te veel tijd ingestopt omdat ik dacht dat PID-tuningproblemen lagen aan slechte filtering, en ik de Sav-gol filter meestal gewoon met polynomial 0 (moving average) gebruikte.

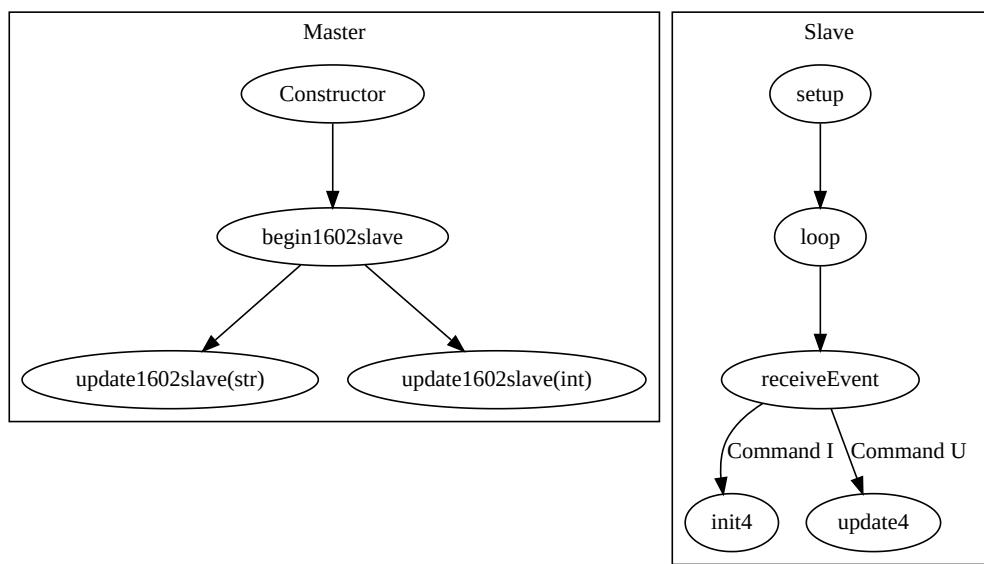
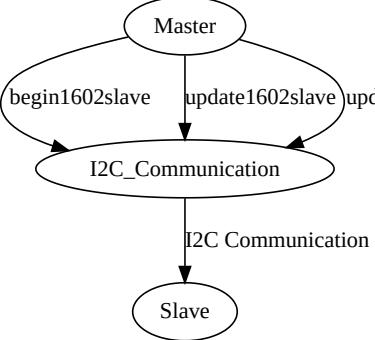
### 2.8.1 Savitzky-Golay Filter Function

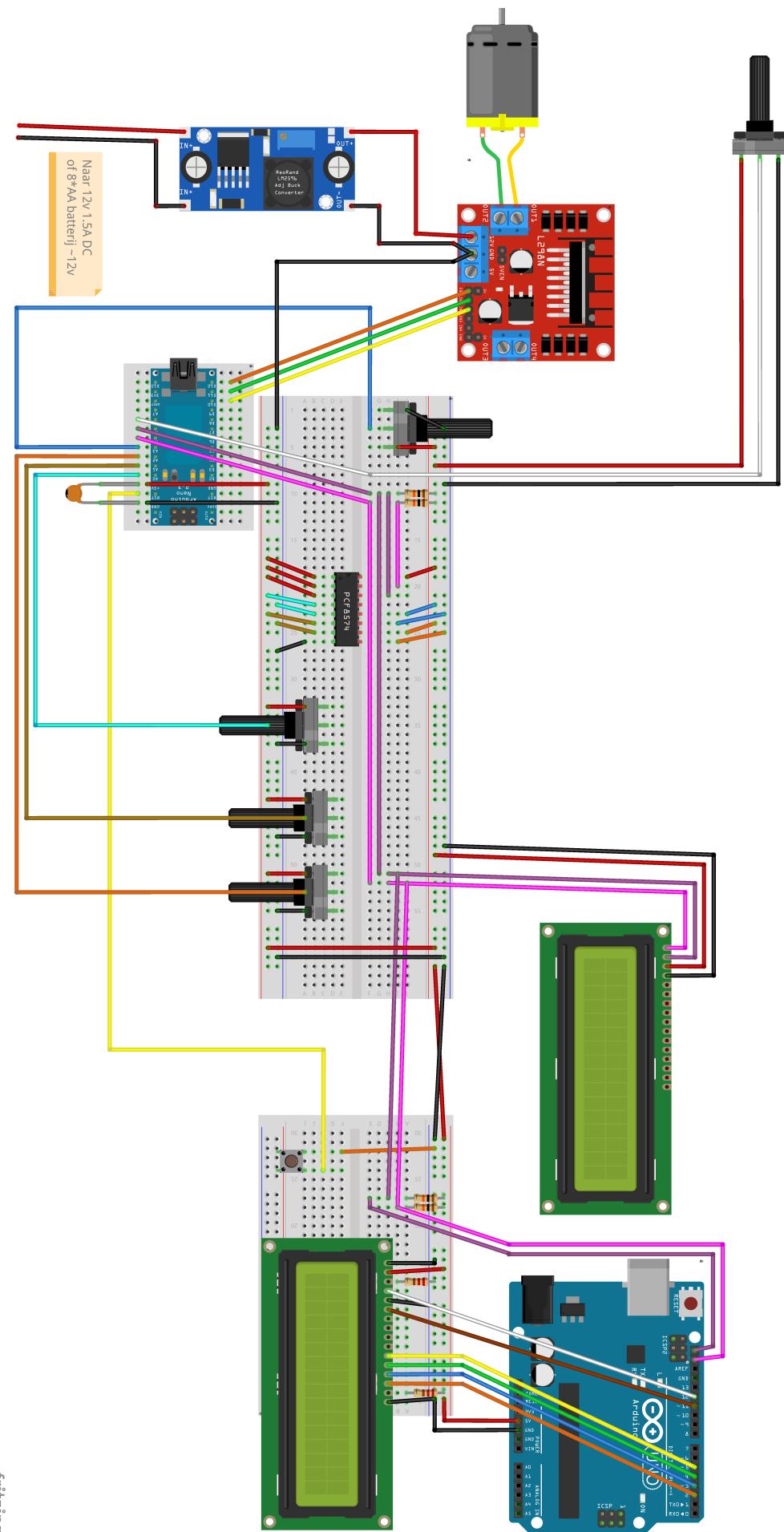


Figuur 9: Diagram of the SavGolayFilter Function

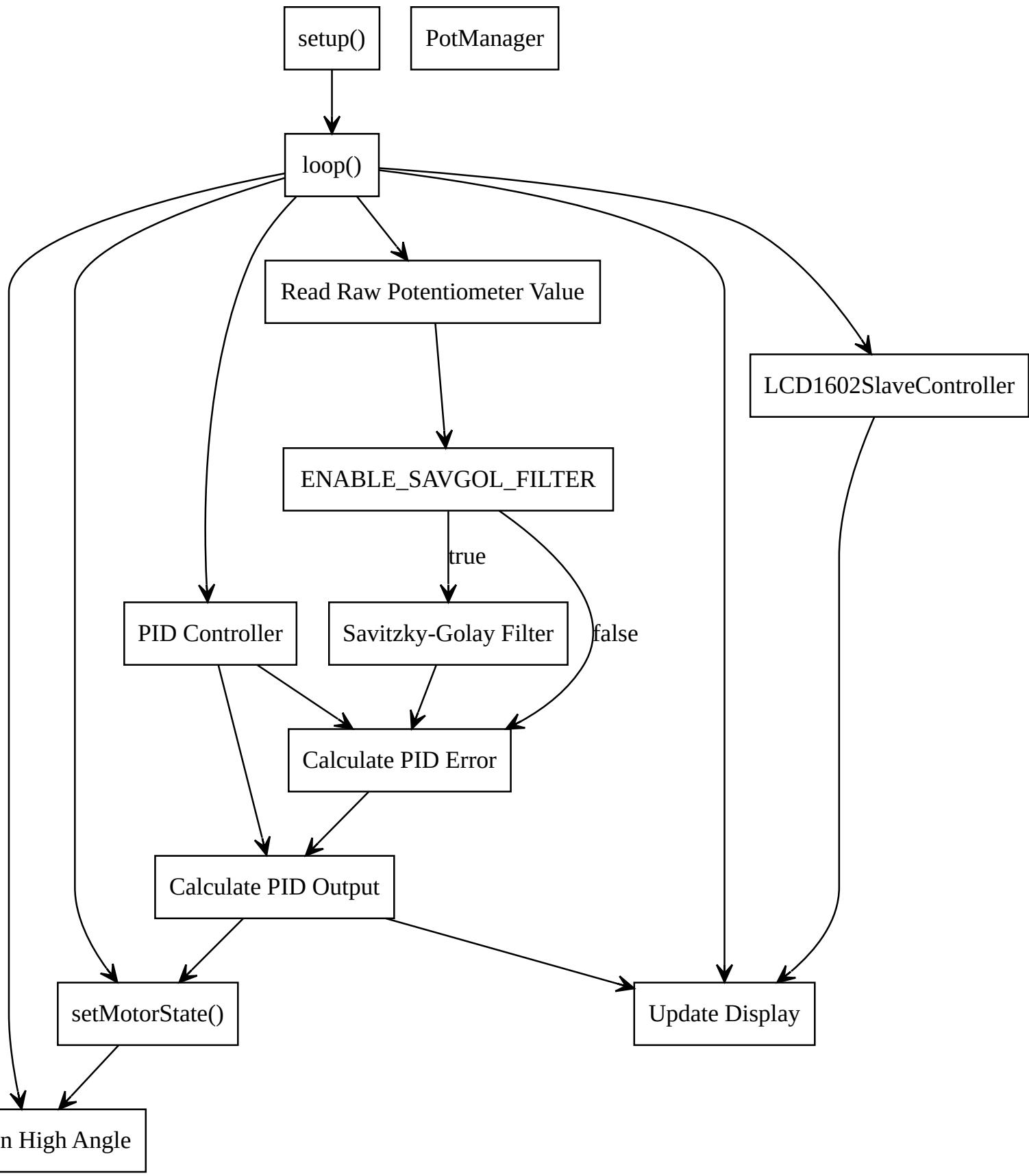
### **3 Appendix**

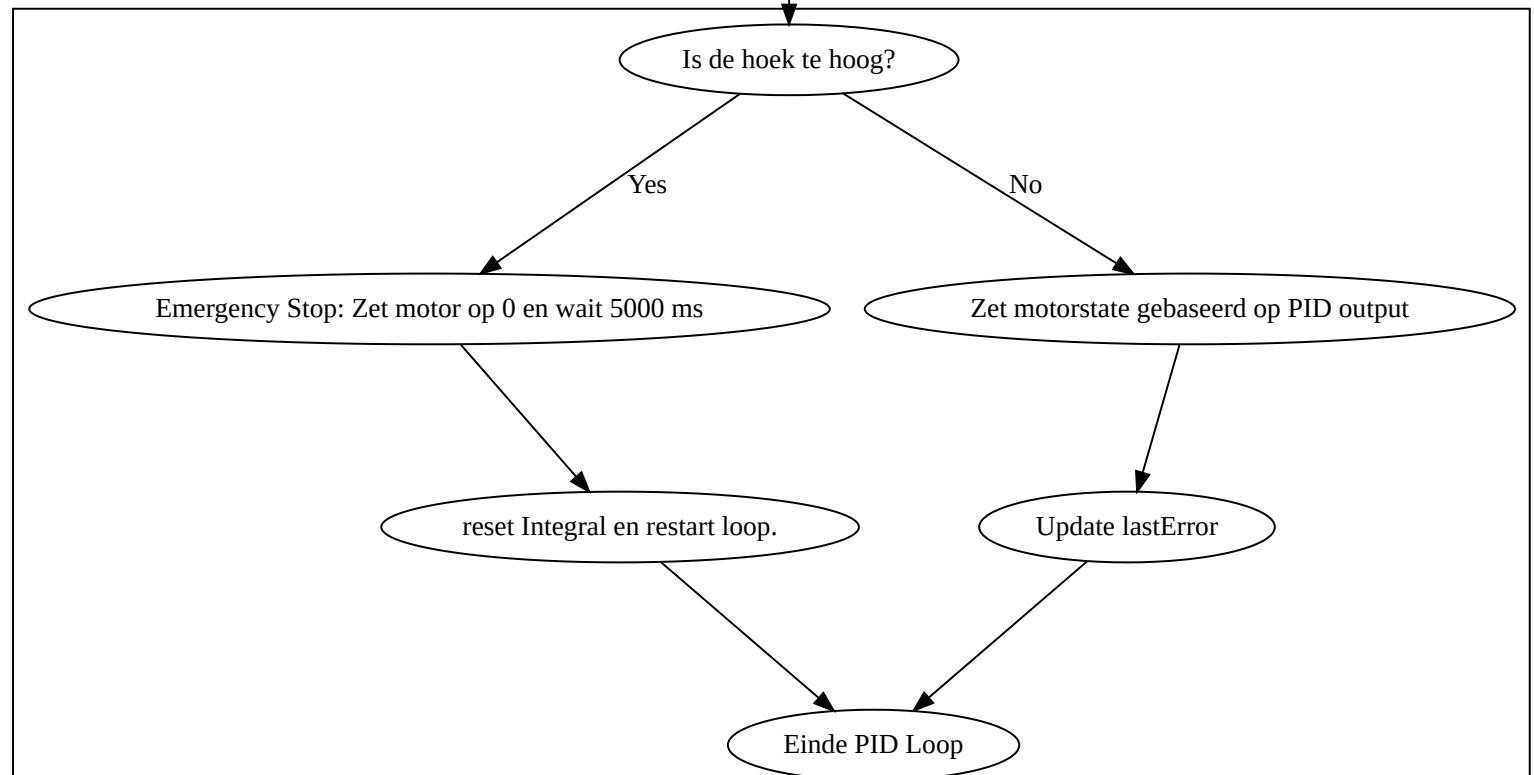
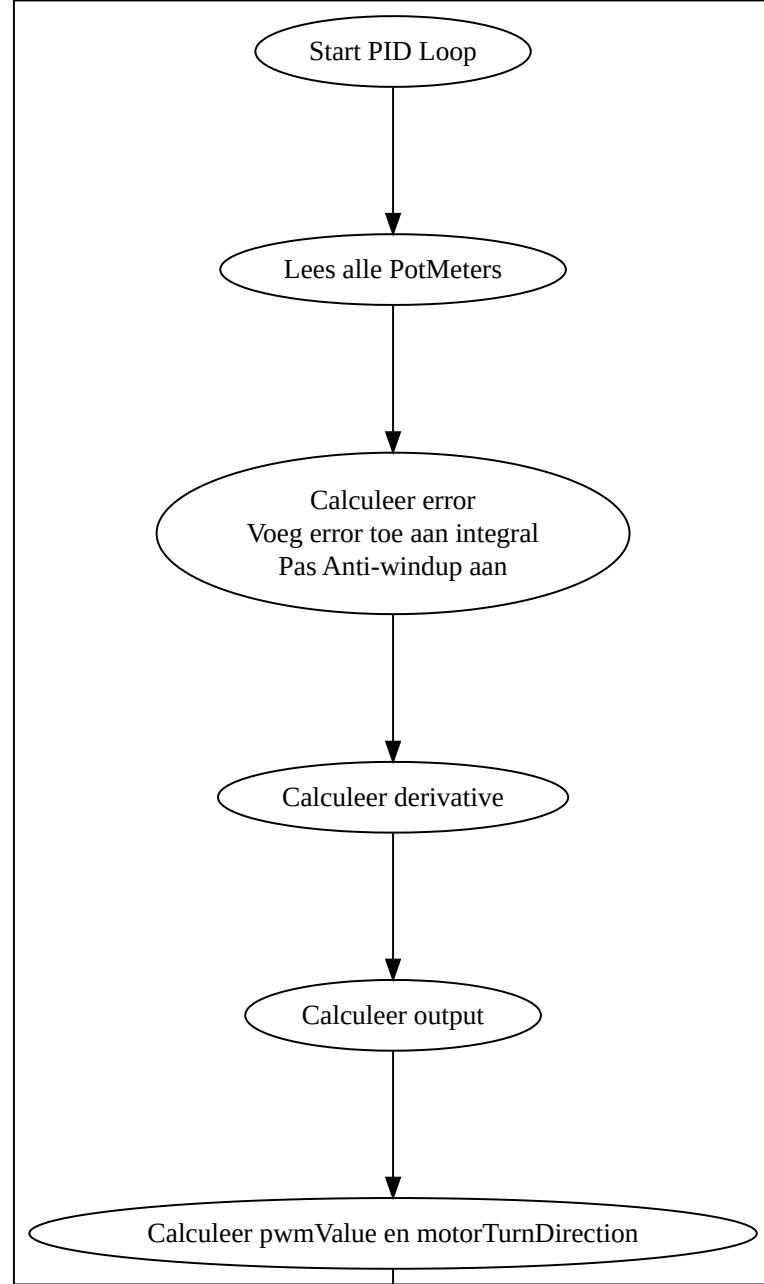
#### **3.1 Extra Afbeeldingen en Diagrammen (Vergroot)**

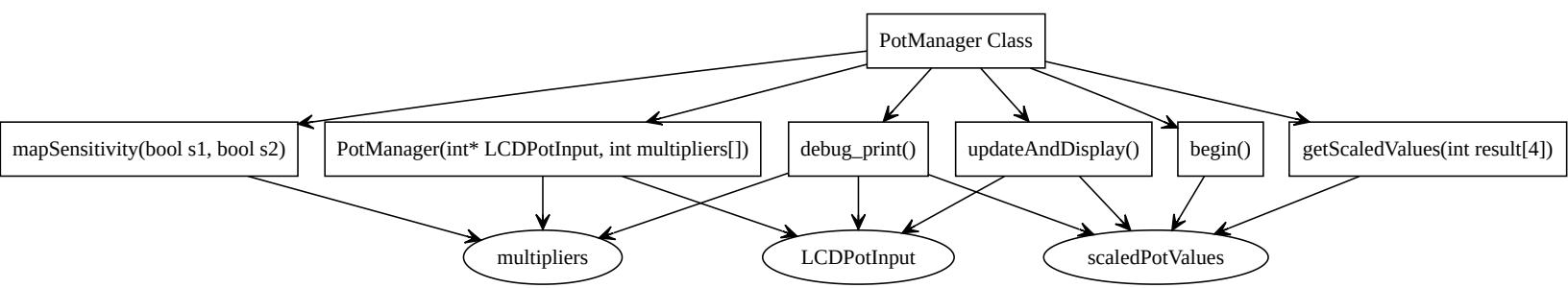


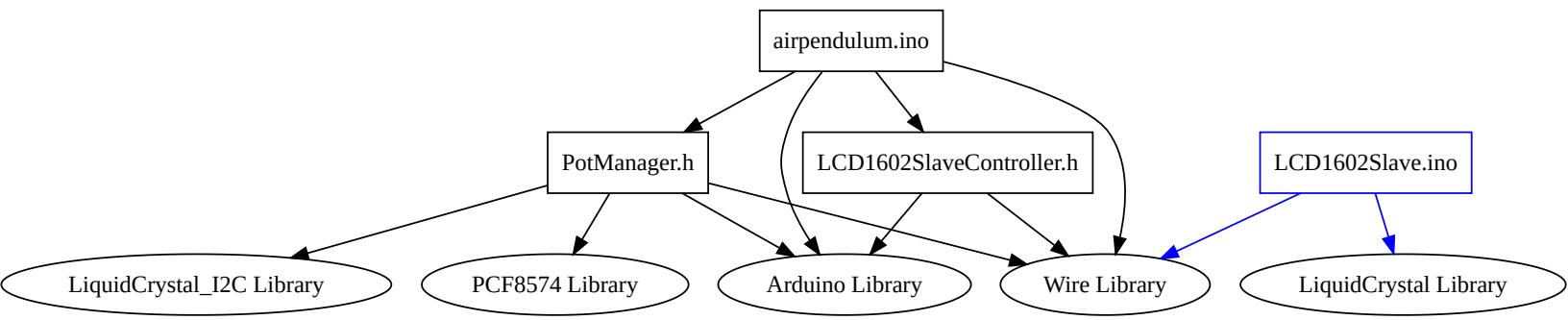


Figuur 10: Fritzing Hardware Diagram









```
587      cpc r10, ZERO_IEQ
588      brge .L6
589      movw r24,r12
590      rjmp .L3
591 .L5:
592      ldi r24,0
593      ldi r25,0
594      rjmp .L3
595 .L6:
596      ldi r24,lo8(-1)
597      ldi r25,lo8(3)
598 .L3:
599      adiw r28,40
600      in tmp_reg, SREG
601      cli
602      out SP_H,r29
603      out SREG, tmp_reg
604      out SP_L,r28
605      pop r29
606      pop r28
607      pop r17
608      pop r16
609      pop r15
610      pop r14
611      pop r13
612      pop r12
613      pop r11
614      pop r10
615      pop r9
616      pop r8
617      pop r7
618      pop r6
619      pop r5
620      pop r4
621      ret
622 main:
623     ldi r24,lo8(2)
624     ldi r25,0
625     call savGolayFilter(unsigned int)
626     ldi r24,0
627     ldi r25,0
628     ret
```

Figuur 11: ATMEL instructions voor originele code

```

1  __zero_reg__ = 1
2  savGolayFilter(unsigned int):
3      lds r18,SGdataIndex
4      ldi r19,0
5      moww r30,r18
6      lsl r30
7      rol r31
8      subi r30,lo8(-(SGdata))
9      sbci r31,hi8(-(SGdata))
10     std Z+1,r25
11     st Z,r24
12     moww r24,r18
13     adiw r24,1
14     ldi r22,lo8(5)
15     ldi r23,0
16     call __divmodhi4
17     sts SGdataIndex,r24
18     ldi r30,lo8(SGdata)
19     ldi r31,hi8(SGdata)
20     moww r26,r30
21     ldi r24,0
22     ldi r25,0
23 .L2:
24     ld r18,X+
25     ld r19,X+
26     add r24,r18
27     adc r25,r19
28     ldi r18,hi8(SGdata+10)
29     cpi r26,lo8(SGdata+10)
30     cpc r27,r18
31     brne .L2
32     ldi r22,lo8(5)
33     ldi r23,0
34     call __udivmodhi4
35     cp r22,__zero_reg__
36     ldi r24,4
37     cpc r23,r24
38     brlo .L3
39     ldi r24,0
40     ldi r25,0
41 .L4:
42     ld r18,Z+
43     ld r19,Z+
44     add r24,r18
45     adc r25,r19
46     ldi r18,hi8(SGdata+10)
47     cpi r30,lo8(SGdata+10)
48     cpc r31,r18
49     brne .L4
50     ldi r22,lo8(5)
51     ldi r23,0
52     call __udivmodhi4
53 .L3:
54     movw r24,r22
55     ret
56 main:
57     ldi r24,lo8(2)
58     ldi r25,0
59     call savGolayFilter(unsigned int)
60     ldi r24,0
61     ldi r25,0
62     ret
63 SGdataIndex:
64     .zero 1
65 SGdata:
66     .zero 10
67 coefficientDivider:
68     .word 139
69 coefficients:
70     .word -10
71     .word 14
72     .word 29
73     .word 33
74     .word 29
75     .word 14
76     .word -10

```

Figuur 12: ATMEL instructions voor geoptimaliseerde Sav-gol code.

## 3.2 Natuurkundige statica berekeningen voor benodigde kracht motor om staaf te bewegen.

Met dank aan Wolfram Alpha, ChatGPT en goedkeuring van mijn 3 huisgenoten die werktuigbouwkunde/Natuurkunden studeren.

### 3.2.1 Step 1: Calculate the Moment of Inertia for the Aluminum Bar

We first calculate the moment of inertia for the aluminum L profile bar of length 1 m and mass 0.115 kg. It is pivoted 70 cm from one end and 30 cm from the other end.

$$I_{\text{bar}} = \frac{1}{12} \times 0.115 \text{ kg} \times (1 \text{ m})^2 + 0.115 \text{ kg} \times (0.2 \text{ m})^2 = 0.009583 \text{ kg} \cdot \text{m}^2 + 0.0046 \text{ kg} \cdot \text{m}^2 = 0.014183 \text{ kg} \cdot \text{m}^2$$

### 3.2.2 Step 2: Calculate the Moment of Inertia for Motor and Counterweight

For the motor side:

$$I_{\text{counter}} = 0.0404 \text{ kg} \times (0.3 \text{ m})^2 = 0.0036 \text{ kg} \cdot \text{m}^2$$

For the counterweight:

$$I_{\text{counter}} = 0.0404 \text{ kg} \times (0.3 \text{ m})^2 = 0.0036 \text{ kg} \cdot \text{m}^2$$

### 3.2.3 Step 3: Calculate Total Moment of Inertia

Summing the moments of inertia for the bar, motor, and counterweight:

$$I_{\text{total}} = I_{\text{bar}} + I_{\text{motor}} + I_{\text{counter}} = 0.014183 \text{ kg} \cdot \text{m}^2 + 0.0110 \text{ kg} \cdot \text{m}^2 + 0.0036 \text{ kg} \cdot \text{m}^2 = 0.028783 \text{ kg} \cdot \text{m}^2$$

### 3.2.4 Step 4: Calculate the Torque Required

To move 25 degrees (0.4363 rad) in 1 second, we calculate angular acceleration ( $\alpha$ ):

$$\alpha = \frac{\theta}{\Delta t} = \frac{0.4363 \text{ rad}}{1 \text{ s}} = 0.4363 \text{ rad/s}^2$$

Using the formula for torque ( $\tau$ ):

$$\tau = I_{\text{total}} \times \alpha = 0.028783 \text{ kg} \cdot \text{m}^2 \times 0.4363 \text{ rad/s}^2 = 0.0126 \text{ N} \cdot \text{m}$$

This is the torque needed to pivot the bar ±25 degrees within 1 second.

### 3.2.5 Step 5: Calculate Thrust Required

To convert this torque to thrust, use:

$$\text{Thrust} = \frac{\tau}{r} = \frac{0.0126 \text{ N} \cdot \text{m}}{0.7 \text{ m}} = 0.018 \text{ N}$$

To convert this to grams of thrust:

$$\text{Thrust (g)} = 0.018 \text{ N} \times \frac{100 \text{ g}}{9.81 \text{ m/s}^2} \approx 1.8 \text{ g}$$

Achteraf komt dit vrijaardig overeen met de gemeten thrust van 2.3 gram (door de motor + propellor vast te maken op een weeg schaal, en het verschil tussen motor aan en uit te wegen.)