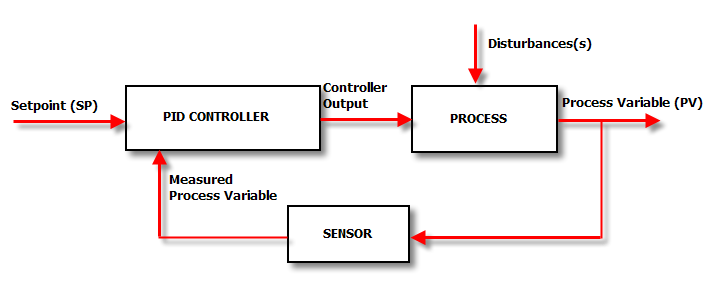
**Labo PID en orde process van een stappenmotor**

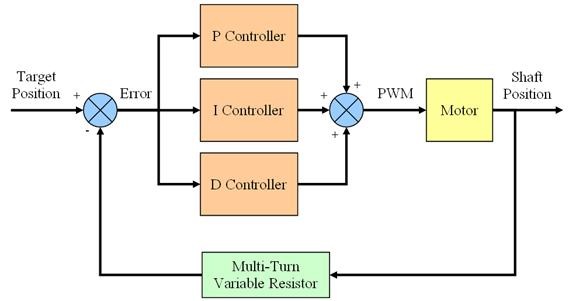
**Doelstelling**

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.

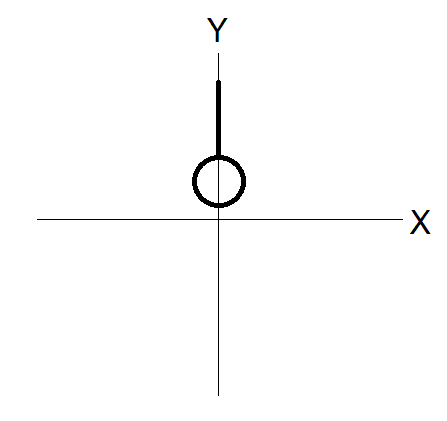
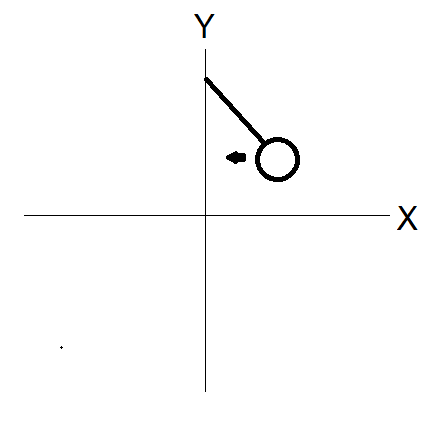
De formules voor PID zijn:

* P ->
* I ->
* D ->

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.

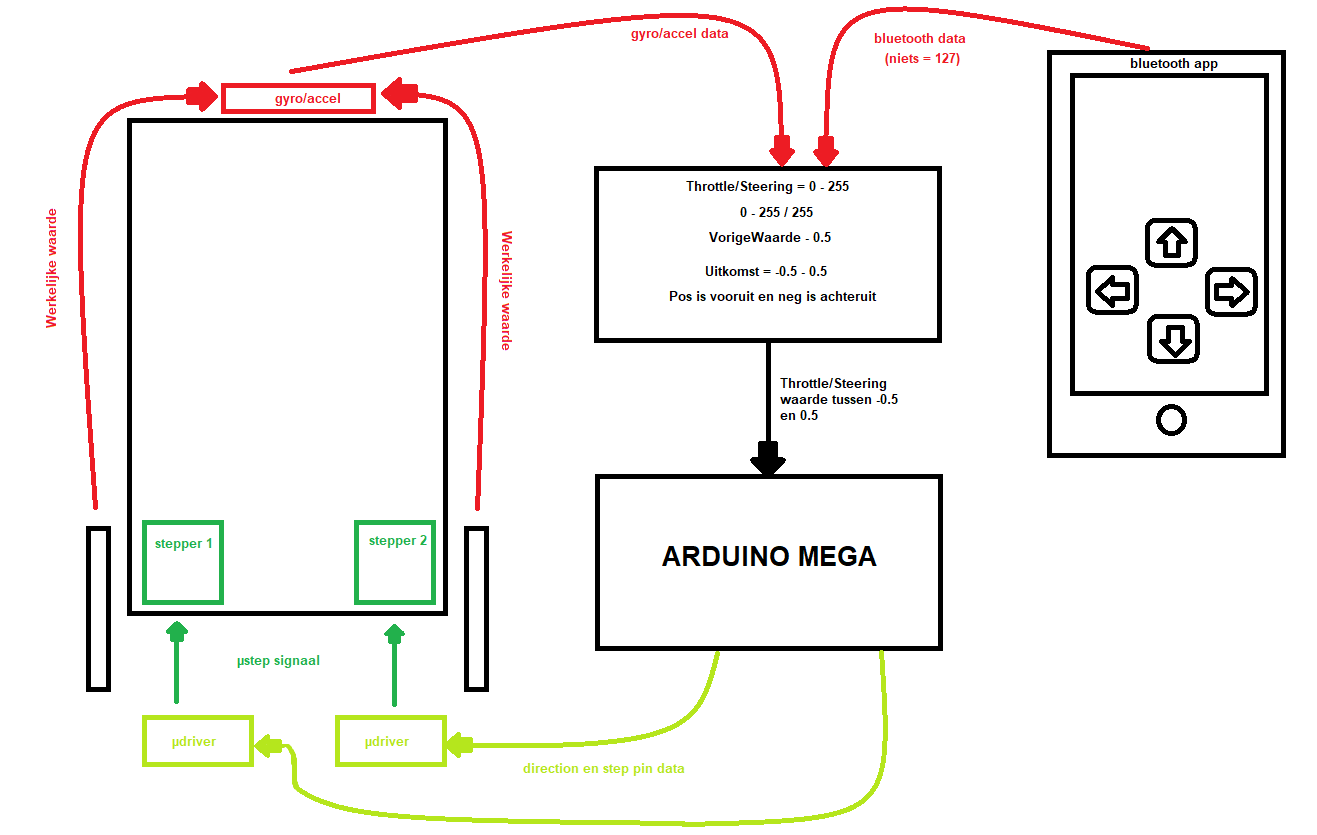
* Als we dan P I en D tezamen zouden gebruiken

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.

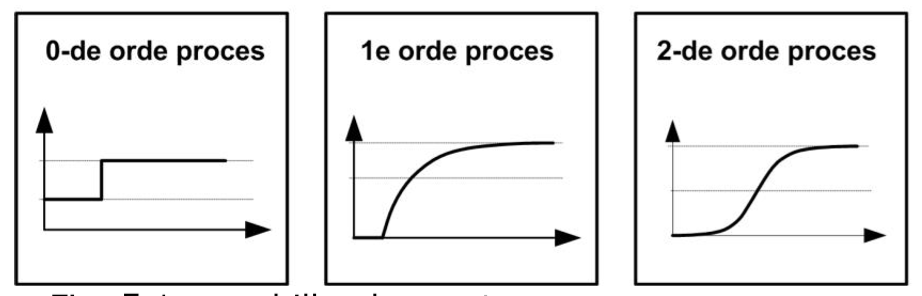
**Toepassing bij balancerende robot**

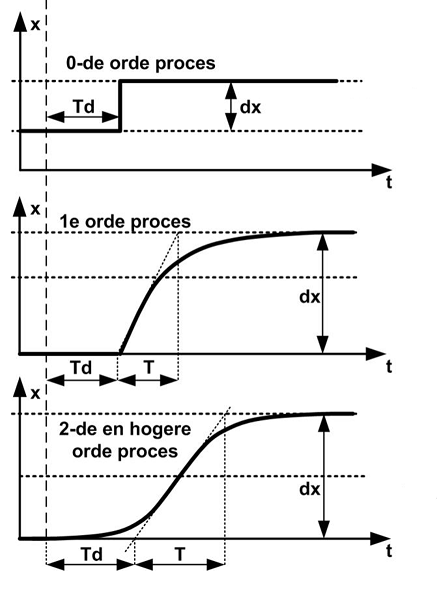
Aan het vorige voorbeeld kunnen we duidelijk zien waarom we PID nodig gaan hebben voor een balancerende robot. Ik zal het nut van PID voor de robot uitleggen ook al is er niet veel verschil met het vorige voorbeeld. Het enige verschil is dat (volgens de oude versie van deze GIP) er 2 regelaars werden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

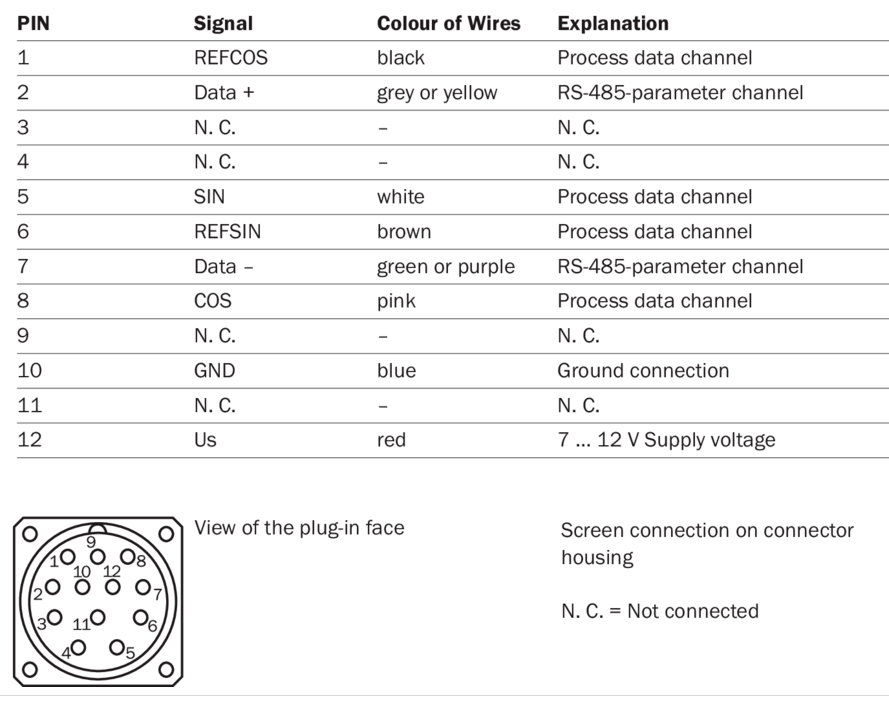
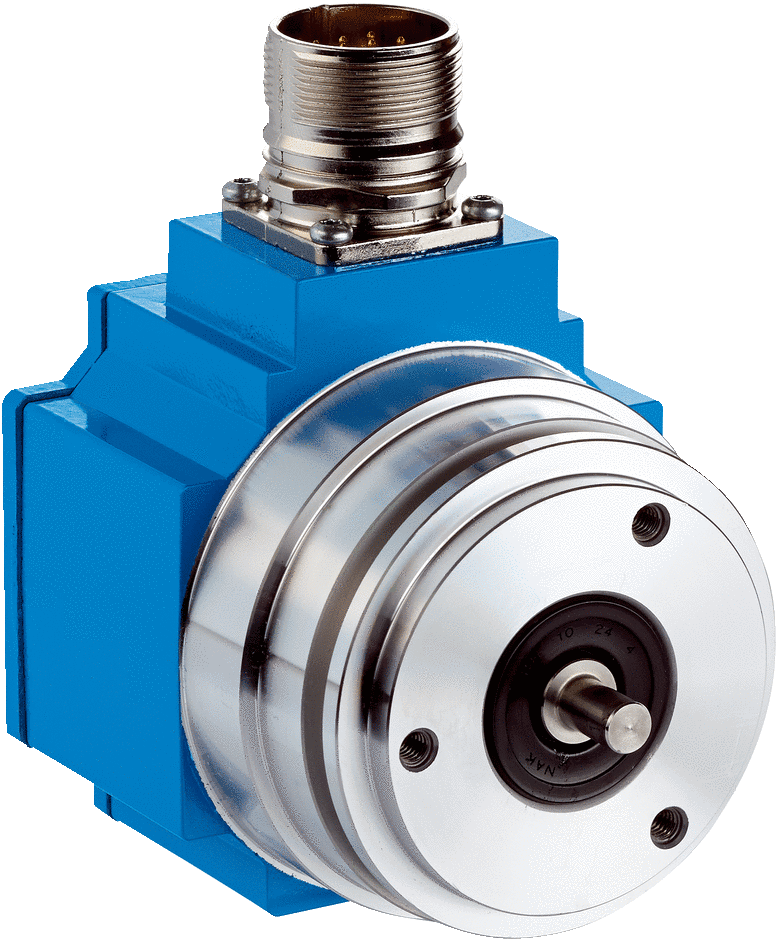
Een orde process is simpelweg de manier hoe het process zal reageren op verandering door de controller. Dus bij mijn robot is het hoe de stappenmotoren zouden reageren op een verandering van hun toerental. Hieronder zie je de 3 verschillende orde processen.

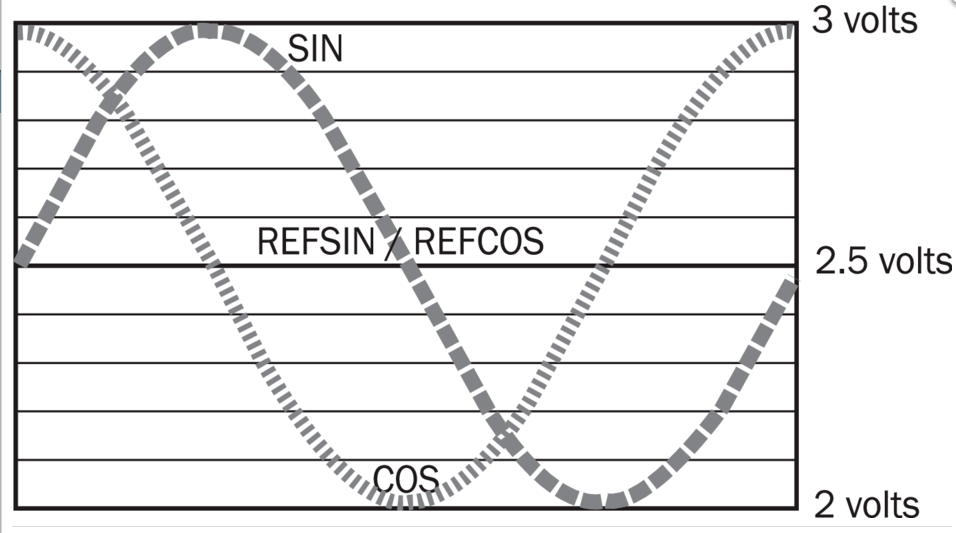
Er is iets waar wel rekening mee moet gehouden worden : de dode tijd of Td. Td begint wanneer de controller een nieuw signaal stuurt naar het process en het eindigt wanneer PV begint met veranderen door dat signaal. Het andere waar rekening mee moet gehouden worden is : T. T begint wanneer Td eindigt 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

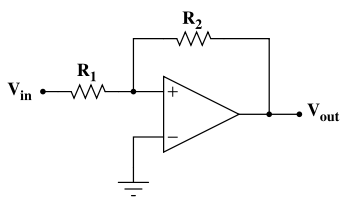
**Orde process bepalen**

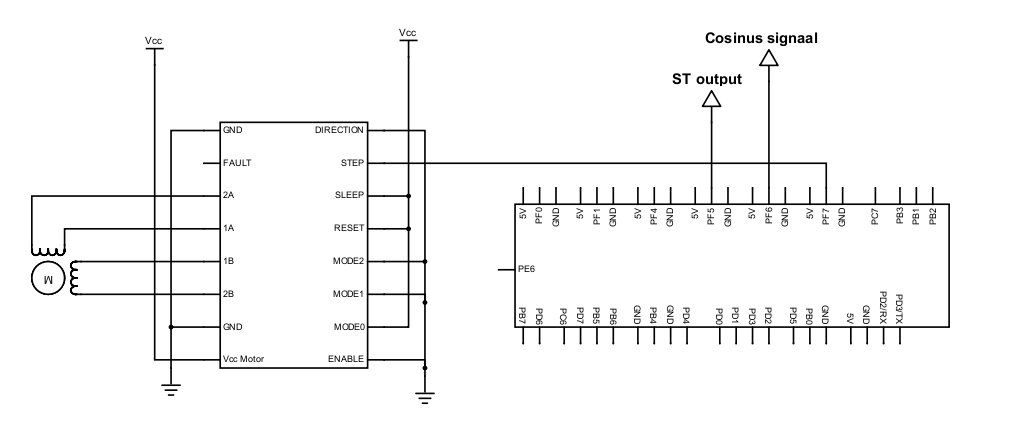
Ik kreeg de taak om uit te zoeken hoe de stappenmotoren die ik gebruik, zouden reageren op snelheidsveranderingen. Dat moest ik dan doen met een Rotary encoder (de srs50-hxa0-k21). Dat kon ik doen door de rotor van de motor aan de rotor van de encoder te verbinden en de motor te laten draaien. Als je 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.

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**

Meetpunt

**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**));**

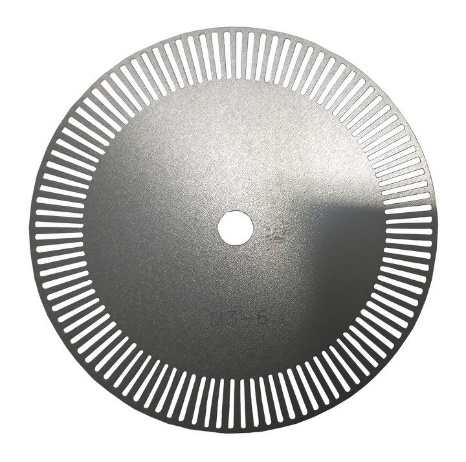
// return A/D conversion result

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.