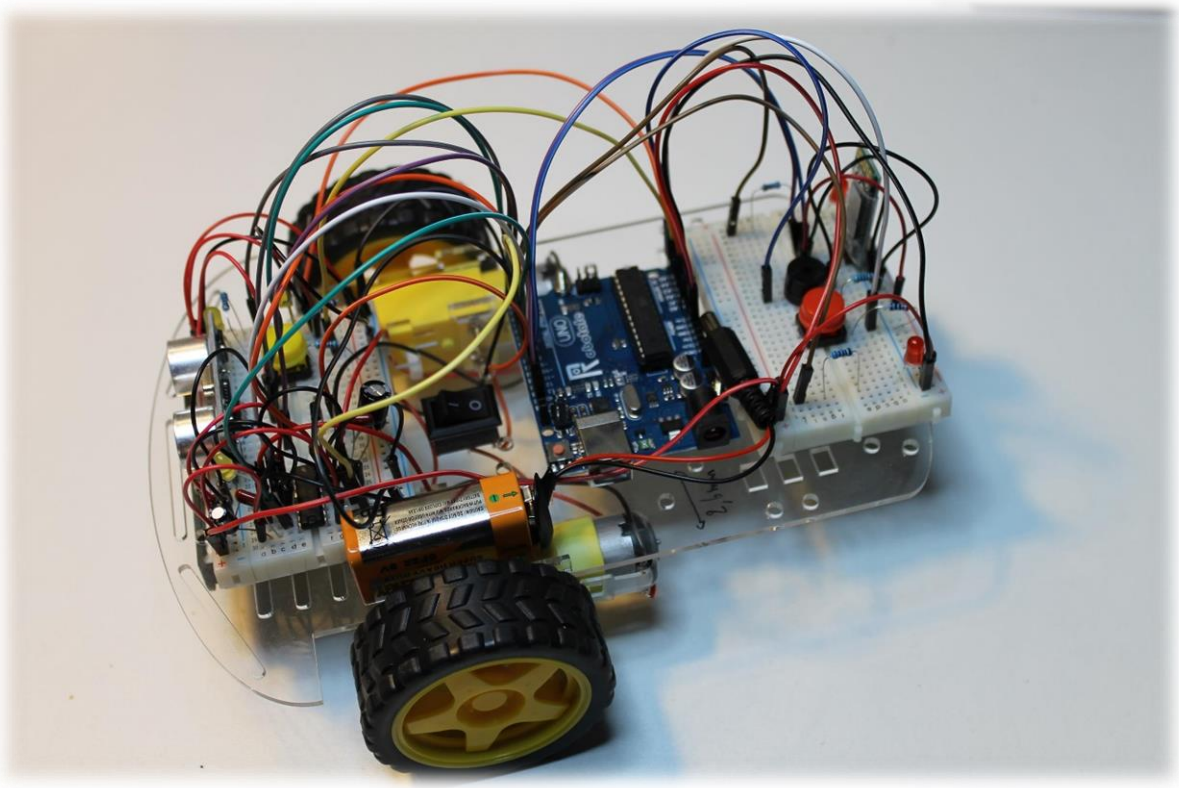
Basisoefeningen programmeren in Arduino



Versie: 8.0

Datum: 27/01/2021

Auteur: Frank Marchal

Doelgroep: 11-77 jaar

Inleiding

Dit document heeft tot doel de cursist zover te krijgen dat hij de **Arduino UNO** omgeving onder de knie kan krijgen. Er zijn ook voorbeelden van de Ardublock en Flowcode omgeving te vinden in deze cursus. Gebruik **enkel wat je nodig hebt** om jouw omgeving te leren kennen.

Eerst wordt er met **basiselektronica (LED, switch, buzzer, motor zonder/met PWM)** gewerkt en gaan we zo de programmeer omgeving leren ontdekken en uitzoeken. Daarna zoeken we uit hoe we met een Arduino UNO bordje allerlei sensoren zoals een **afstandssensor** kunnen aansturen. Ook leren we de robot met **bluetooth** besturen.

Tenslotte wordt de **robot met lijnvolger** aangestuurd. Ook regelkringen, subroutines en arrays worden toegelicht.

Het Arduino UNO bordje wordt geprogrammeerd via een **USB kabel** dewelke verbonden is met een PC.

Op de PC voorzien we de **Arduino IDE** omgeving. Hoe deze software moet geïnstalleerd worden kan je terugvinden in het meegeleverde **installatiedocument**.

Arduino c-code voorbeeld

Veel succes met jouw eerste Arduino ervaringen.

Moesten er nog vragen zijn, dan kan je mij altijd mailen op info@edulab.be

1. Korte kennismaking met een microcontroller

Een microcontroller is te vergelijken met een **kleine computer**. Wanneer je deze niet vertelt wat hij moet doen, dan zal hij gewoon weg ook niets doen.

De microcontroller wordt dus via USB geprogrammeerd.

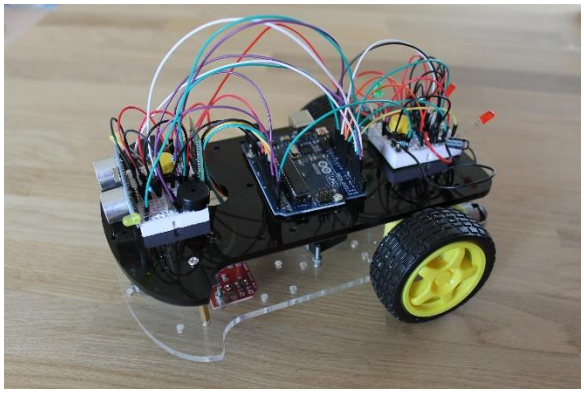
We kunnen aan de microcontroller signalen aanbieden (druk op de knop, afstandssensoren, lijnvolgers, bluetooth …). Dit noemen we de **INPUTS**. Deze inputs worden door de controller afgevraagd en afhankelijk van het programma dat we hebben geschreven wordt er iets nuttig mee gedaan.

➔ Zo werkt een koffiemachine bijvoorbeeld pas als je op de **startknop** drukt, wanneer er een **tas** is voorzien om de koffie in te doen, wanneer er **water en koffiebonen** aanwezig zijn in de machine. In dit geval moeten we 4 inputs hebben voordat de koffie kan gemaakt worden.

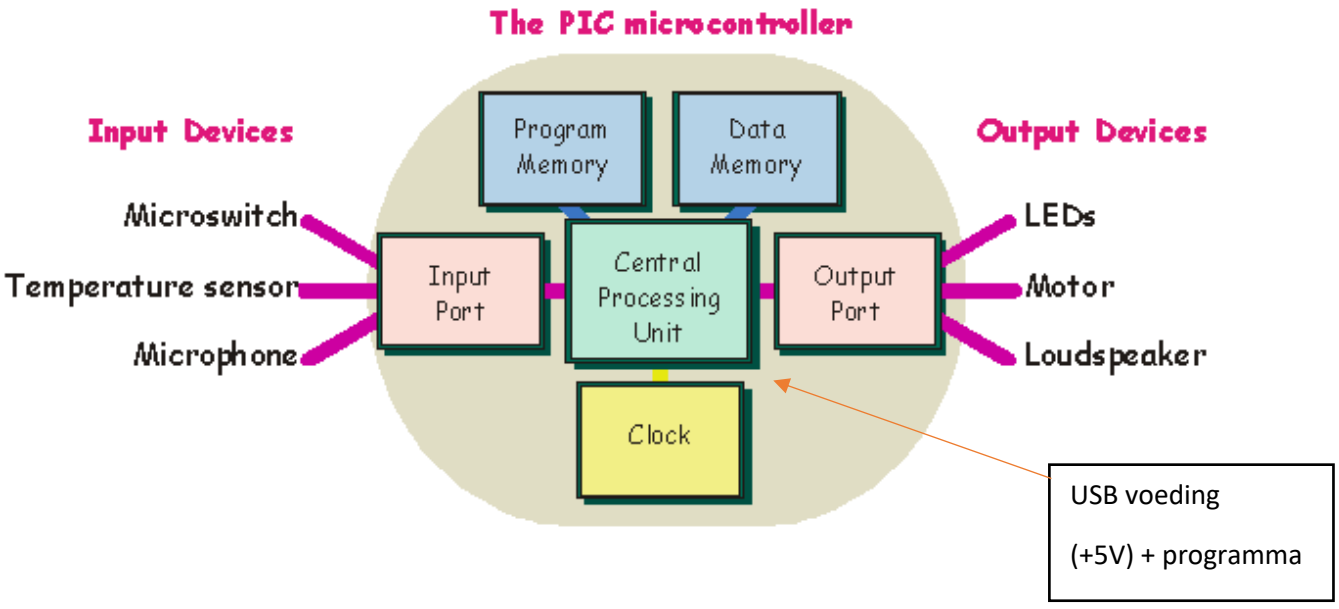
De **VERWERKING** gebeurt in de microcontroller van de koffiemachine. Hij zal moeten zorgen dat bijvoorbeeld de bonen gemalen worden, het water verwarmd wordt, alles in verhouding bij elkaar wordt gevoegd, enz …. Hiervoor moeten er dus de nodige metingen (extra INPUTS) en aansturingen (**OUTPUTS**) gedaan worden.

Een verwarmingselement, een motor die bonen maalt, een klep die opent en de koffie naar buiten laat komen zijn dus allemaal voorbeelden van OUTPUTS.

➔ Bij onze Arduino robot werkt dit net zo.



We kunnen deze info ook als volgt voorstellen:



De robot heeft als **inputs** de schakelaars, afstandssensor, bluetooth verbinding, …

De robot **verwerkt** alles in de microchip microcontroller ATMEGA 328P-PU.

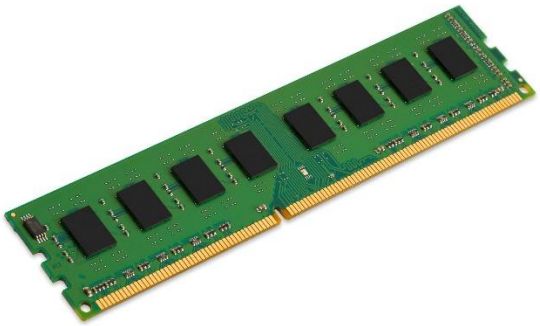
De robot heeft als **outputs** de LEDs, motoren en een zoemer. Ook een seriële verbinding met de PC is mogelijk om tekst op het scherm af te drukken.

De clock van de Arduino draait op een nauwkeurig 16MHz X-TAL.

Welke geheugens zitten er in een microcontroller?

1. **Flash** geheugen (niet vluchtig) dient als het programma geheugen. Hier wordt, net als bij een USB stick het programma in gedownload.
2. **RAM** geheugen (vluchtig) dient als werkgeheugen. Tijdens het draaien van jouw programma wordt er bijvoorbeeld een waarde opgeslagen in dit geheugen. Een tijdje later wordt dit weer opgehaald om te verhogen. Denk aan een teller. Zit typisch in een PC als werkgeheugen (druk maar eens windows + pauze en lees deze waarde af).
3. **EEPROM** geheugen (niet vluchtig) dient om bijvoorbeeld meetwaarden of kalibratie waarden in op te slaan die we niet mogen kwijt geraken wanneer de batterij zou plat gaan van de opstelling. Denk aan het meten van temperatuurwaardes gedurende een ganse week in een lokaal. De BIOS van een PC bevat ook een gedeelte EEPROM met de basisinstellingen van de PC.

FLASH



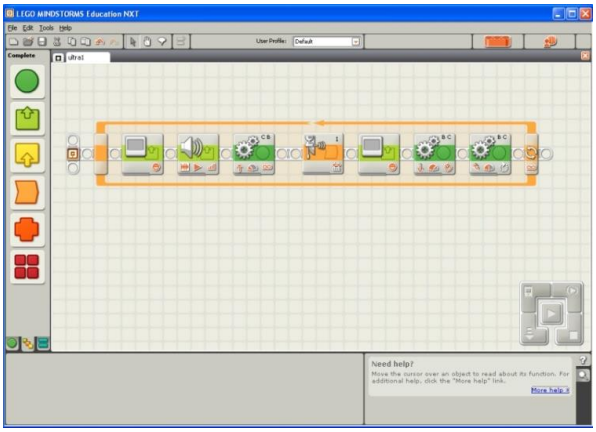
RAM

EEPROM

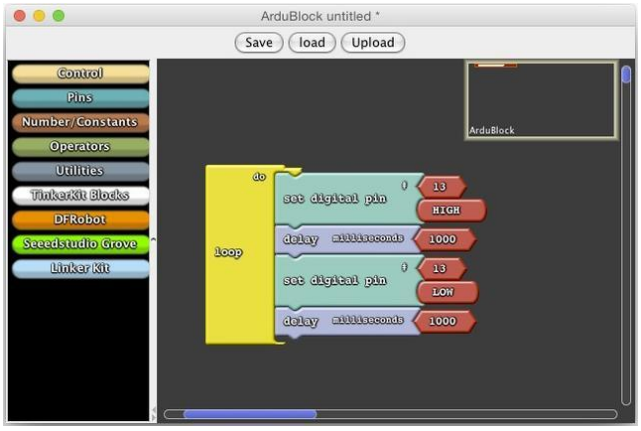
Hoe denkt nu een microcontroller?

Een microcontroller is een elektronische bouwblok die denkt, net als een computer. Hij werkt zijn instructies (wat hij moet uitvoeren) **stap per stap** af.

Bij de **LEGO NXT robot** zit er ook een microcontroller in het witte bakje. Via draden kunnen we INPUTS en OUTPUTS op de VERWERKINGS blok aansluiten. Het programma schrijven we in een LEGO flowcode omgeving.



Bij onze Arduino omgeving gaat dit net zo wanneer je gebruik maakt van o.a. **ARDUBLOCK of Snap4Arduino**. Dit is een **visuele omgeving** waar we via **blokjes**, die in elkaar passen, programma's kunnen maken.

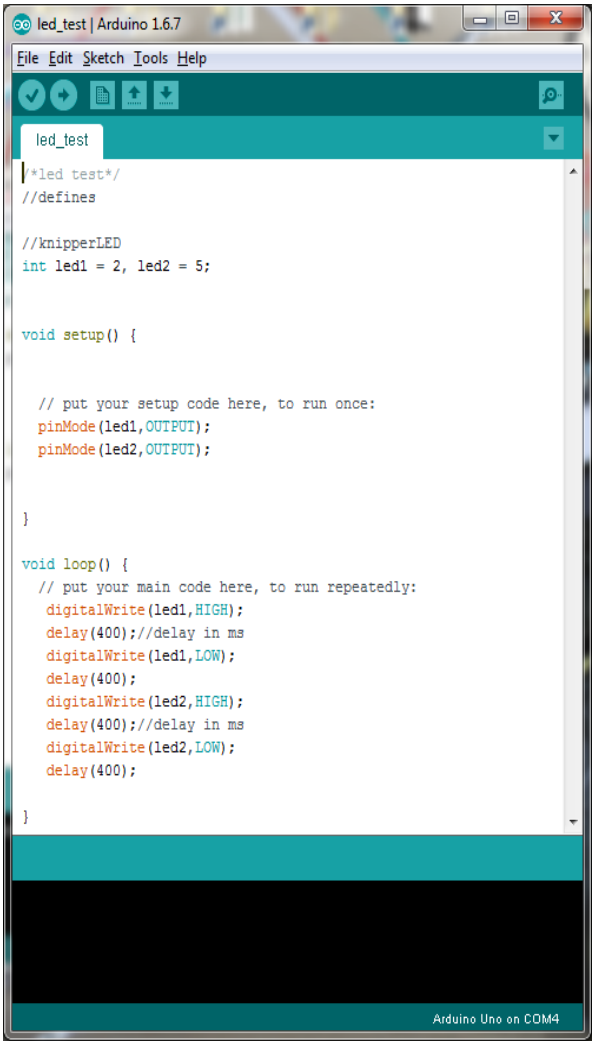


Dit programma heeft als taak gekregen om een **LED te laten knipperen**.

De LED zal 1 seconde aangaan en dan weer 1 seconde uitgaan.

Snap4arduino heeft veel weg van Scratch. Deze omgeving laat ook toe dat je met blokjes kan programmeren. **Nadeel is dat je altijd een USB kabel** nodig hebt. **Voordeel** is dat je iets **visueel** kan tonen terwijl je met het arduino bord iets aanstuurt.

Wij gaan eerst werken met **ARDUBLOCK** waarna we parallel aan de slag zullen gaan met de **Arduino IDE** omgeving. **IDE** staat voor “Integrated Development Environment”.



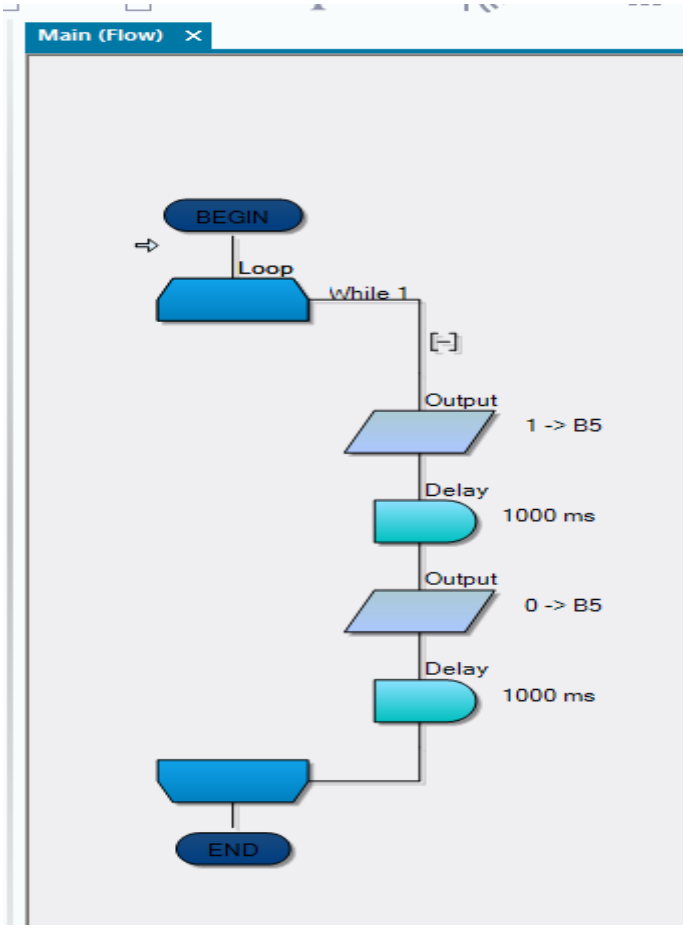
Deze code zal **2 LEDs laten knipperen**. We moeten in deze omgeving echter **meer gegevens zelf toevoegen** dan bij S4A of ARDUBLOCK opdat de microcontroller weet wat hij moet doen.

We schrijven dus nu de commando’s in de vorm van **tekst** i.p.v. met blokjes.

Deze omgeving laat ons toe om veel **moeilijker projecten** aan te vatten en zelf meer in te stellen.

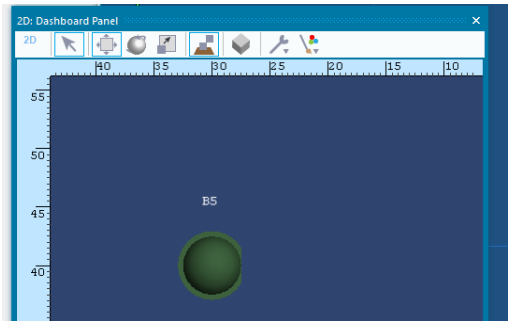
In de bedrijfswereld kiest men eerder voor deze programmeertaal die we **c-code** noemen.

Arduino programmeren via **Flowcode 8? (4EE only)**



Met het **visuele programma Flowcode** kunnen we ook **blokjes slepen** in een flowdiagram. Samen vormen ze een programma dat de microcontroller moet gaan uitvoeren. In dit geval knippert een LED tegen 1 seconde op de digitale output pin 5. In de blokjes kan je eigenschappen instellen.

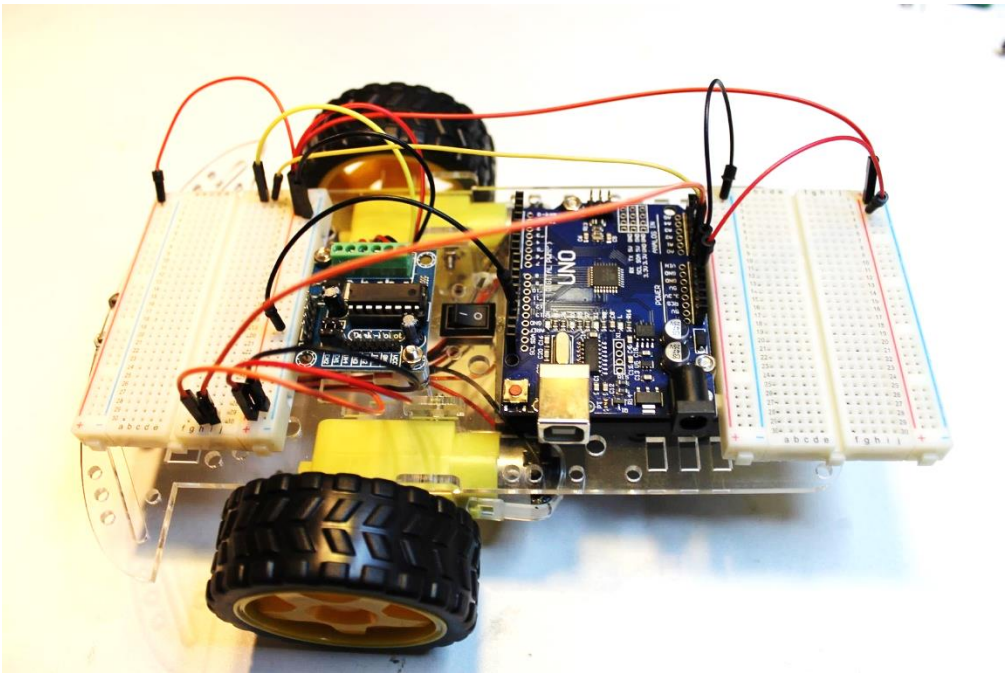
Het voordeel aan flowcode is dat we een simulatie omgeving hebben, waarbij we de tijd kunnen trager zetten en stap per stap kunnen debuggen. We kunnen ook onze componenten tonen in het programma en zien of ze juist worden aangestuurd.



2. De Arduino programmeer omgeving ontdekken

Wat hebben we allemaal nodig om aan de slag te kunnen gaan?

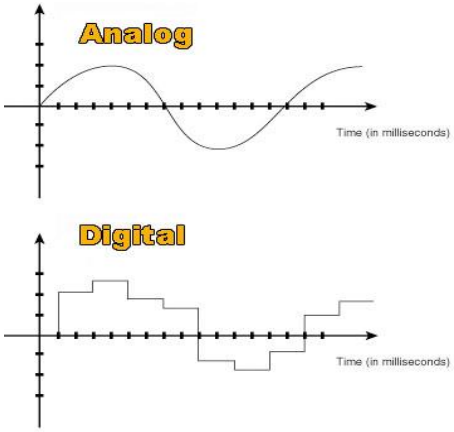
1. Een Arduino UNO bordje
2. Een USB kabel (voeding + programmatie)
3. Een Arduino IDE omgeving geïnstalleerd op onze PC
4. Andere elektronica componenten (afhankelijk van het project)
5. Een robotchassis met 2 breadboards en 2 DC motoren (zie aparte montage **handleiding Arduino robotchassis**)
6. Draadjes



Onbestukt robotchassis

Enkele basisbegrippen:

Wat is het **verschil** tussen analoog en digitaal?



Een **analoog signaal** is een waarde die **tussen 0 en de maximum waarde** kan zijn. Zo kan dit bij een Arduino tussen 0 en 1023 (2^{10}) zijn wanneer een analoog signaal wordt ingelezen door de ADC (analoge digitale converter in de microcontroller).

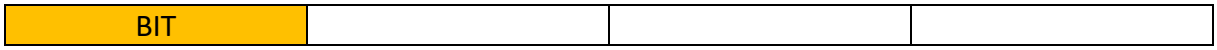
Typische toepassing van een analoog signaal is het inlezen van geluid via een microfoon of het meten van licht met een LDR (komt later nog aanbod).

Een **digitaal signaal** is een waarde die slechts **alleen 0 of 1** kan zijn. In dit geval spreken we van bijvoorbeeld een gesloten of open schakelaar, of van een LED die aan of uit is.

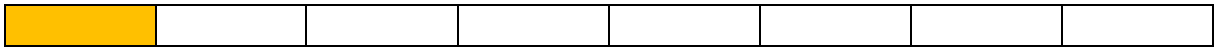
Wat is de **relatie** tussen de bits?

BIT

NIBBLE = 4 bits

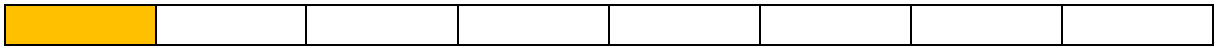


BYTE = 8 bits

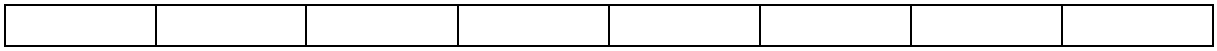


WORD = 2 BYTES = 16 bits

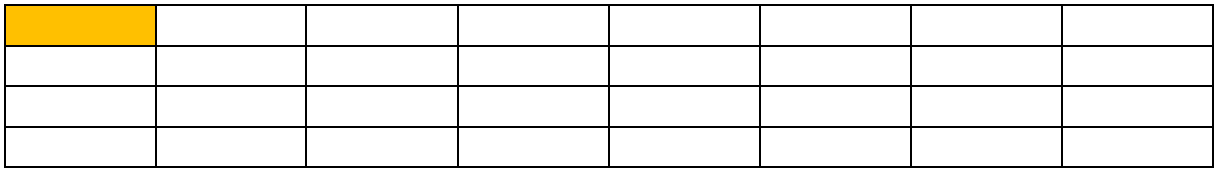
MSB (most significant byte)



LSB (least significant byte)

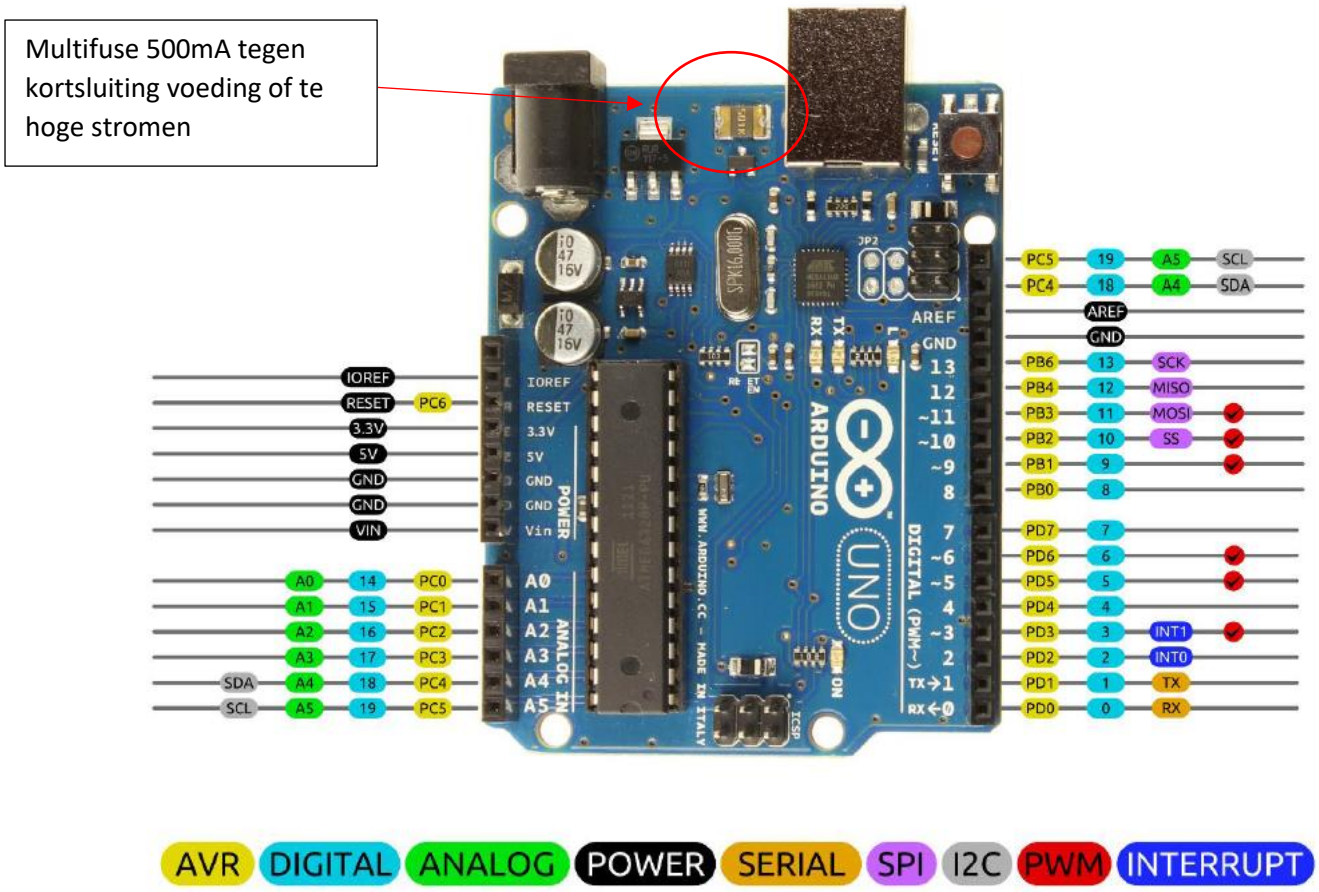


DOUBLE WORD = 4 BYTES = 32 bits



Hoe werkt het **Arduino UNO** bordje?

Volgende figuur geeft een overzicht van alle pinnen die we kunnen gebruiken en aansturen.



De blauwe pinnen zijn de digitale pinnen (1 of 0)

De groene pinnen zijn de analoge pinnen (waarde 0 tot 255)

De zwarte pinnen zijn de voedingen (wij gebruiken de +5V e, de massa (GND))

De controller waarmee we werken is nu een **ATMEGA328P-PU**. Hij heeft een **16MHz X-tal** en er zit een B, C en D poort aan met allerlei functies: analoog, digitaal, PWM, I2C enz …

Deze controller bevat **geen USB module** intern, dus deze functie moet toegevoegd worden door een 2de microcontroller op het bordje: de **ATMEGA16U2**. Deze heeft wel USB aan boord en heeft ook een **16MHz X-tal**. Beide chips communiceren via UART met elkaar.

Voor de rest zijn alle pinnen gewoon naar buiten gebracht op het bordje.

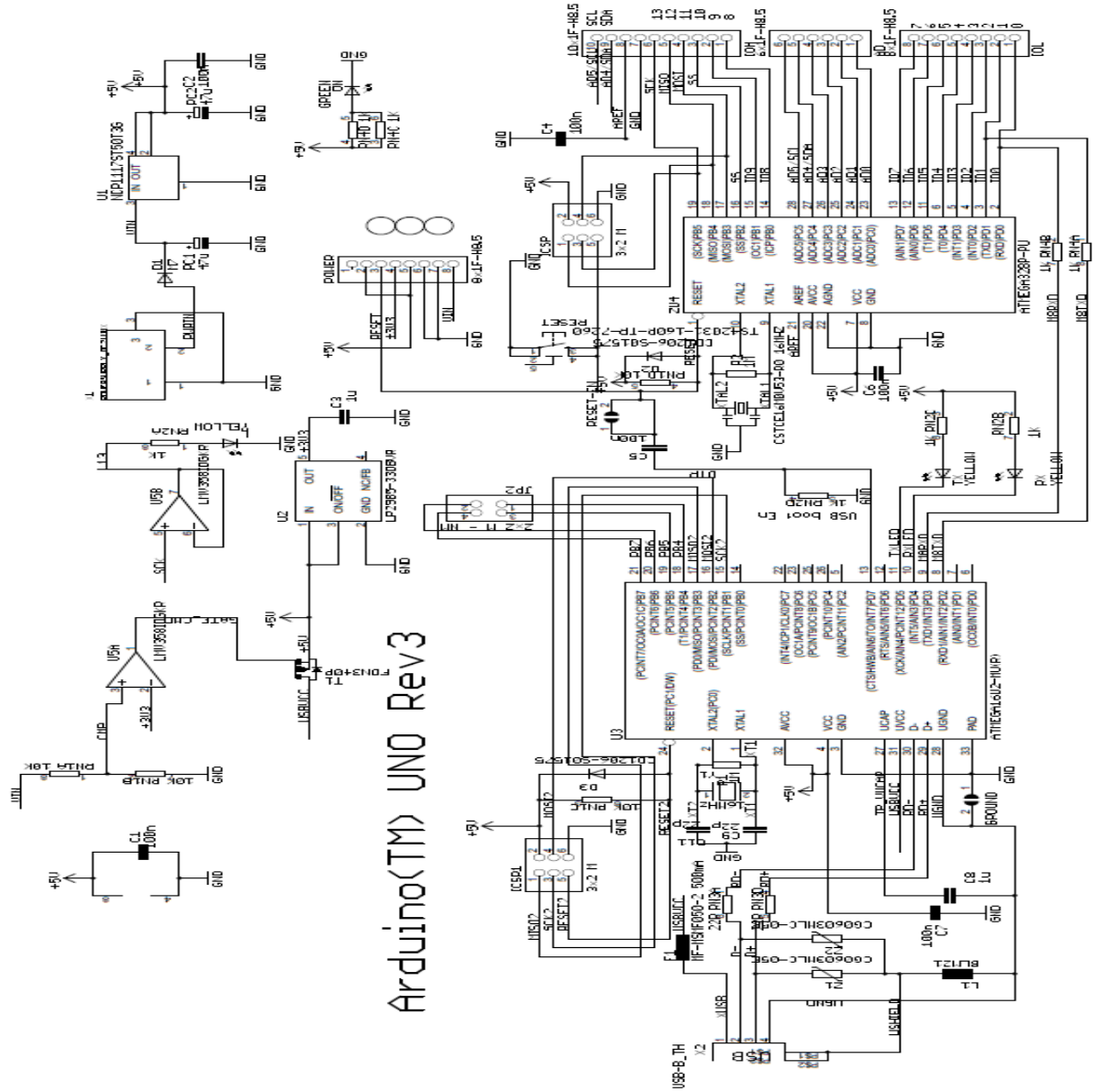
Er zit ook een **5V naar 3V3 omzetter** op het bordje om deze spanning te voorzien op een headerpin. Deze spanning kan je aftappen naar externe elektronica. **Maximum 500mA** mag je verbruiken (zie multifuse). De microcontrollers en de elektronica draaien echter allen op 5V. Dus we kunnen **met de 5V signalen aan de slag**.

Er is een spanningsregelaar **LM1117-5.0** voorzien om, wanneer je geen USB aansluiting en dus voeding wil voorzien van +5V, een adapter van 7V tot 12V aan te sluiten op het bordje. Een **9V batterij** is hier dus een handige oplossing (zie de Arduino robot).

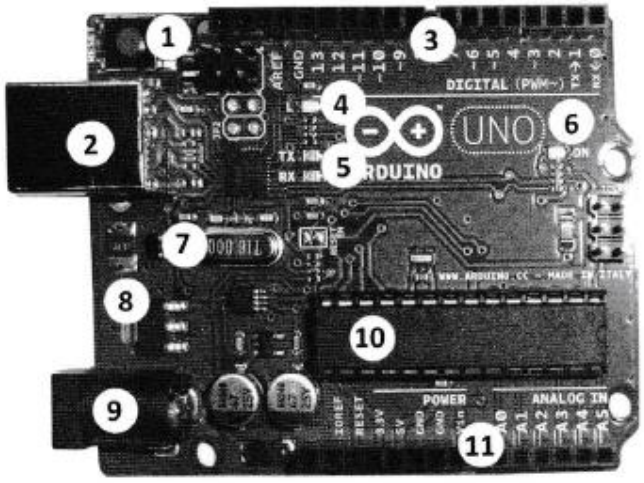
Tot slot is er een **reset knop** voorzien om de controller te resetten wanneer nodig.

De LEDs **TX en RX** (beide geel) lichten op wanneer er **communicatie** is tussen het bordje en de PC (vb bij het downloaden van code of het gebruik van de seriële monitor (zie verder)).

Wanneer de 5V in orde is brandt het **groene ledje "ON".** Dit kan ook rood zijn bij een Chinese Arduino versie!



Volledig schema van de Arduino UNO rev 3



Figuur 1. Arduino Uno.

nummer	Component
1	Reset-knop
2	USB-aansluiting
3	Headers
4	LED op pin 13
5	Communicatie-LEDs
6	aan/uit-LED
7	kristal
8	5V-stabilisator
9	externe voeding (+ op de middenpin)
10	ATmega328P microcontroller
11	Headers

Plaatsing componenten op het Arduino UNO bordje.

Eigenschap	Waarde
Microcontroller	ATmega328P
Werkvoltage	5V
Voeding	7-12V (plus op de middenpin, 500 mA of meer) of via de USB-aansluiting.
Digitale in- en uitvoerpinnen	14 (waarvan 6 PWM)
Analoge invoerpinnen	6
Maximale stroom per I/O pin	40 mA[1]
Maximale stroom uit de 3,3V-pin	50 mA
Programma geheugen, Flash	32 KB (de bootloader verbruikt 0,5 KB)
Data geheugen, RAM	2 KB
Idem, EEPROM	1 KB
Kloksnelheid	16 MHz

Technische gegevens van ons bordje.

3. Starten met de ARDUBLOCK en Arduino IDE omgeving:

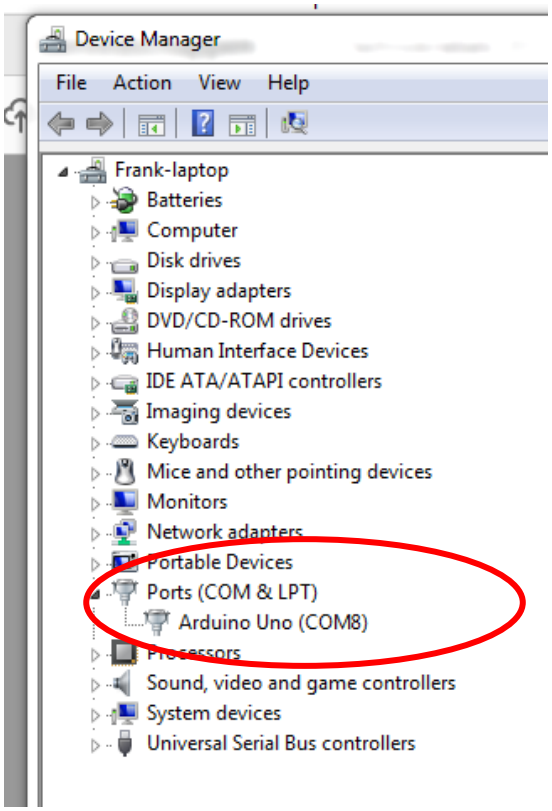
We pluggen onze **USB kabel** in, in het bordje. Hierdoor voeden we de ganse schakeling. Dan gaat de klok van de microcontroller draaien en voert hij het laatste programma uit.

Zorg ervoor dat eerst de **Arduino IDE software en ARDUBLOCK** zijn geïnstalleerd op jouw PC (zie apart installatie document voor Arduino). Wanneer je dan jouw Arduino UNO verbindt met de PC zal de **driver automatisch geladen** worden en kan je snel aan de slag.

Belangrijk is om te weten op welke **COM poort** jouw PC het Arduino bordje heeft voorzien. Deze poortnummer wordt door jouw OS (Windows, Linux …) automatisch toegekend.

Als je wil weten op welke poort de Arduino is geïnstalleerd, dan kan dit bijvoorbeeld in Windows via de **device manager**.Merk op dat in de Arduino IDE dit ook aangeven wordt wanneer je de COM poort moet kiezen.

Druk **“windows” + “pauze” knop** om een mogelijkheid te krijgen om te gaan kijken in de **device manager**.

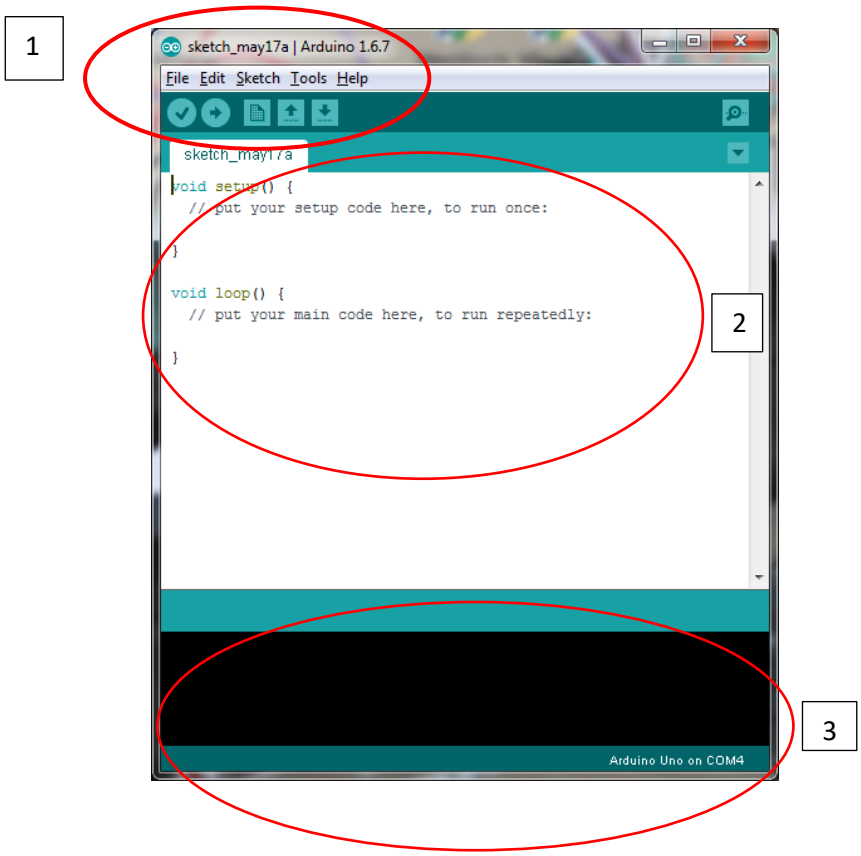


COM8 is op deze PC gekozen voor de Arduino Uno

Merk op dat bij de Chinese Arduino versie hier de tekst “USB 2.0 Serial” of een vergelijkbare tekst kan staan. Zolang er geen geel uitroepteken staat is het prima. Onthoud de COM poort nummer.

Start de Arduino IDE omgeving door te dubbel klikken op volgend desktop icoon:

Als openingsvenster krijg je het volgende window:



1: Het **menu** van het Arduino IDE programma

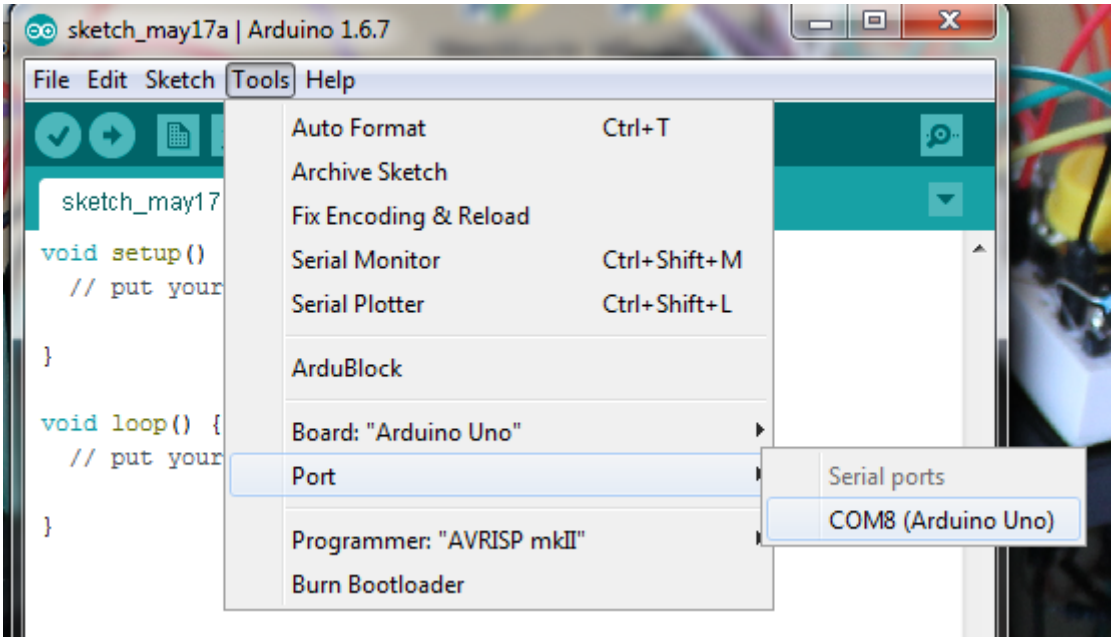
2: De plaats waar we **code** gaan schrijven (noemen ze ook een **sketch**)

3: De **debug informatie** komt hier terecht + welk Arduino bordje je hebt gekozen en op welke COM poort je aan het werken bent

De menu bevat de volgende indeling:

1. **File:** om sketches te laden of te saven. Je kan er ook voorbeelden vinden van oefeningen en de eigenschappen van jouw omgeving instellen
2. **Edit:** hiermee kan je copy, paste en andere bewerkingen op jouw code uitvoeren
3. **Sketch:** hier vind je de mogelijkheid om een sketch te uploaden (in het bordje te stoppen), te compileren (code omzetten van tekst naar begrijpbare taal voor de microcontroller), code te verifiëren (code op fouten nakijken) en eventueel extra bibliotheken (libraries) met voorgemaakte code toe te voegen.
4. **Tools:** hier stellen we de juiste COM poort en kiezen we het Arduino bordje. Ook de mogelijkheid om een serial monitor te gebruiken vind je hier (zie verder).
5. **Help:** voor als je extra hulp nodig hebt of de versie nummer wil controleren.

Zorg dat in de menu TOOLS de juiste **COM poort** en het juiste Arduino bordje (**ARDUINO UNO**) is ingesteld.



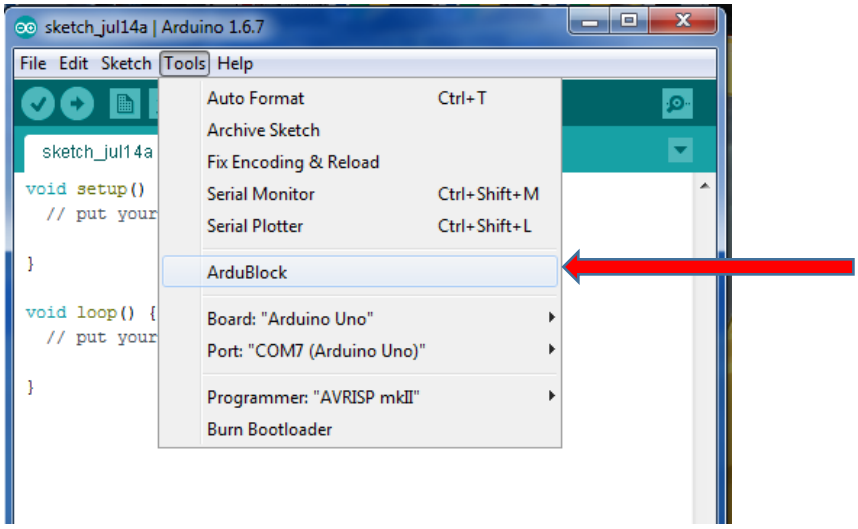
In deze figuur is COM8 gekozen en het Arduino UNO bordje.

Vanaf nu kan er code ingegeven worden en kunnen we deze gaan testen op het bordje.

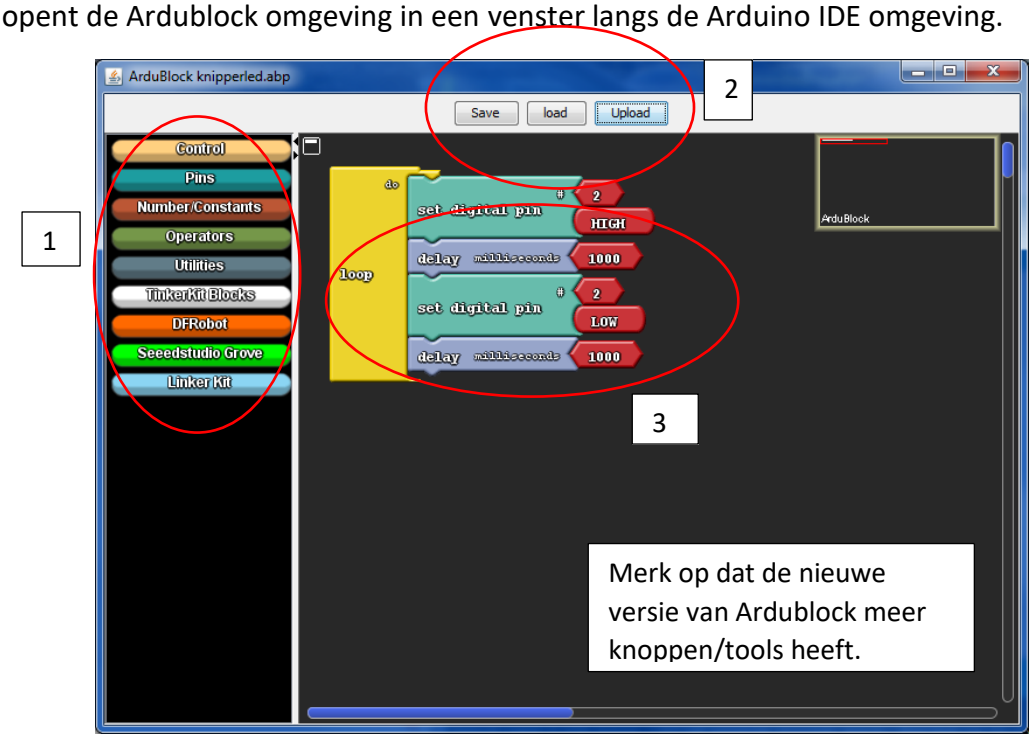
4. Waar vind ik ARDUBLOCK in de Arduino IDE omgeving terug?

Wanneer je eerst de Arduino IDE en het programma ARDUBLOCK hebt geïnstalleerd, dan is het heel makkelijk om het programma ARDUBLOCK te starten in de Arduino IDE omgeving.

Start eerst Arduino IDE op en ga naar de menu “tools”. Klik nu op de tool “Ardublock”.



Nu opent de Ardublock omgeving in een venster langs de Arduino IDE omgeving.

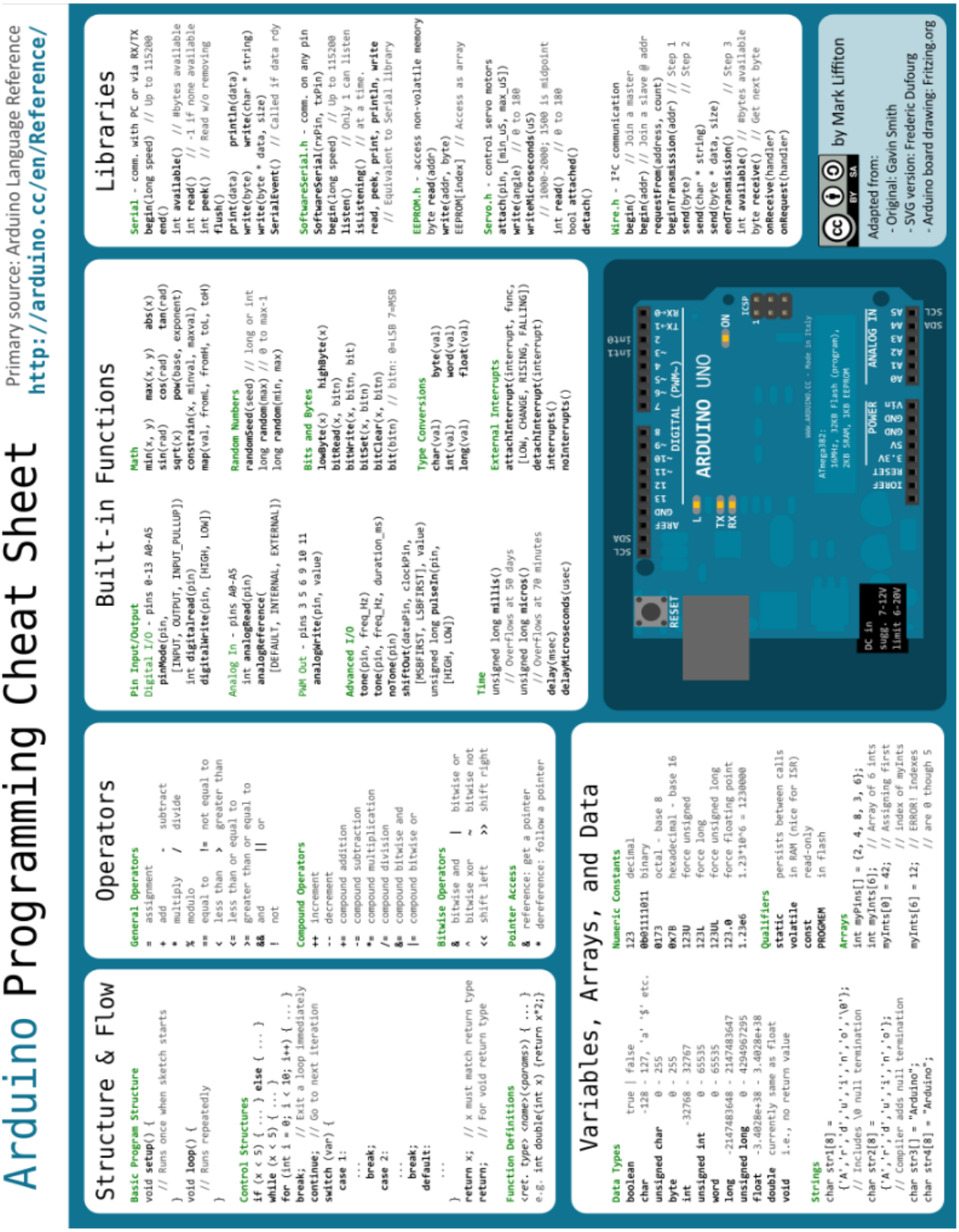


De indeling van het programma is eenvoudig:

1. Mogelijk te gebruiken functies/tools
2. Plek waar we de programma blokjes naar toe zullen sleuren en het programma opbouwen.
3. Opslaan, laden van programma’s en uploaden van het programma in de Arduino.

5. Cheat code

Deze pagina is een referentie voor de meest voorkomende Arduino commando's.



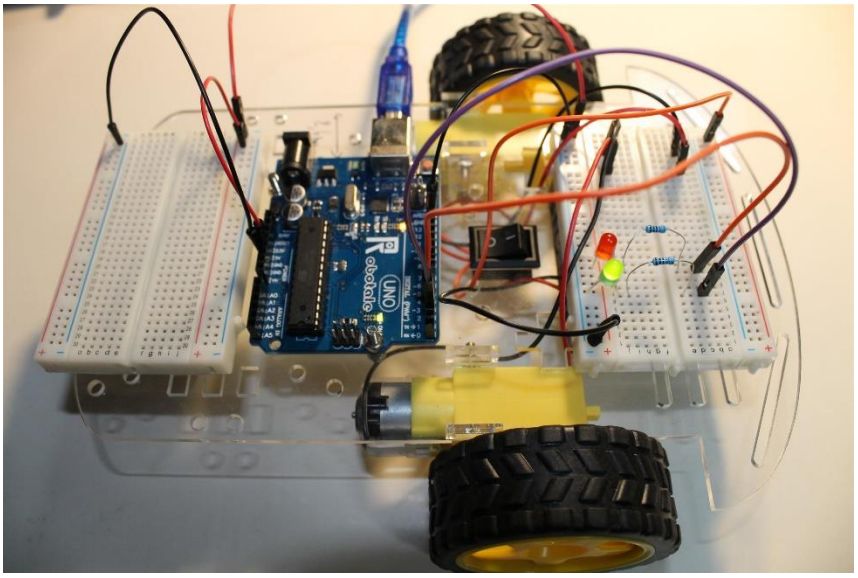
6. Leds aansturen

Doel: we gaan 2 leds op een breadboard leren aansturen.

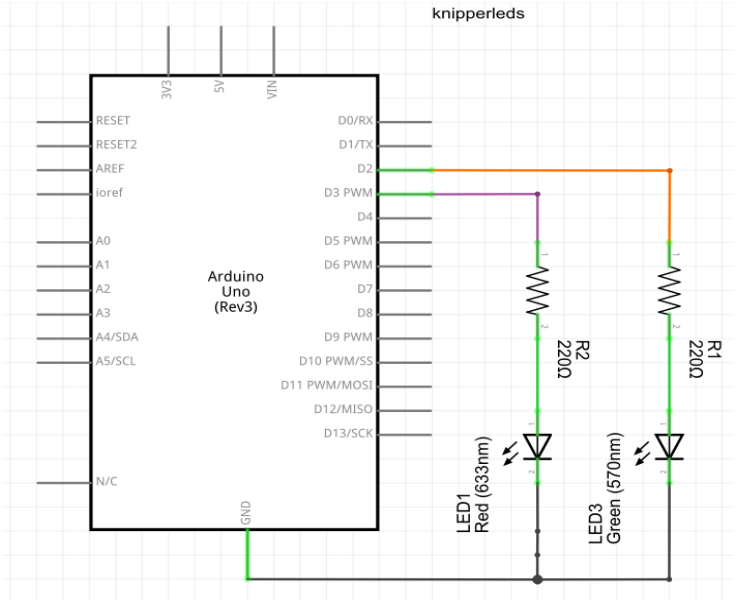
Benodigdheden:

- Arduino UNO
- 2 LEDs en 2 x 220 ohm weerstanden
- USB kabel en robotchassis

1. Finale opstelling op breadboard:

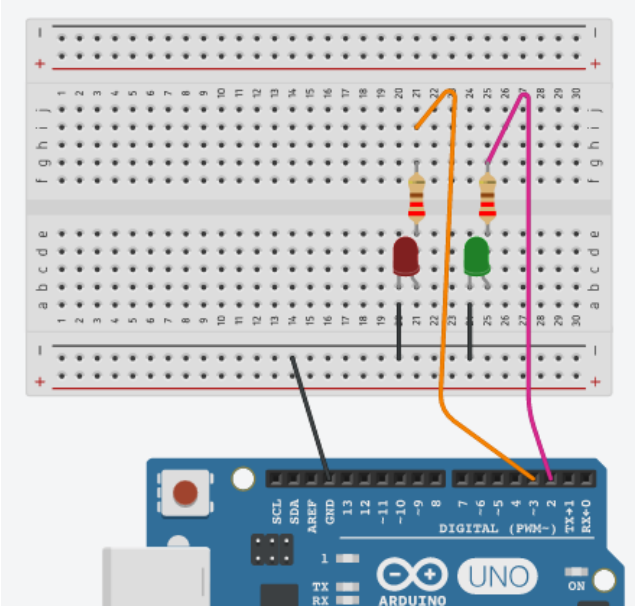


2. Schema:



Dit is het circuit wat we gaan maken op het breadboard. Merk op dat je dit schema zelf kan tekenen in het gratis programma van Fritzing, zie hiervoor http://fritzing.org/download/

3. Het schema opgebouwd op het breadboard ziet er als volgt uit:

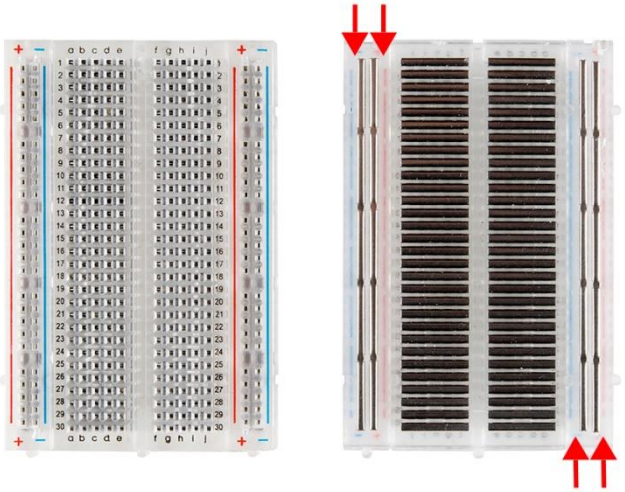


Deze breadboard opstelling kan je ook tekenen via Fritzing of het gratis online programma **Tinkercad**, zie www.tinkercad.com . Je kan hier zelfs de Arduino code uploaden in jouw bord en de schakeling simuleren. Met Tinkercad kan je zowel tekenen als jouw code simuleren. Op dit ogenblik valt er bij Fritzing enkel te tekenen (handig om in een verslag/cursus te gebruiken).

We sluiten dus op de **digitale pin 2 en 3** van de Arduino een LED aan. De weerstanden zijn nodig als stroombeperking (zie verder).

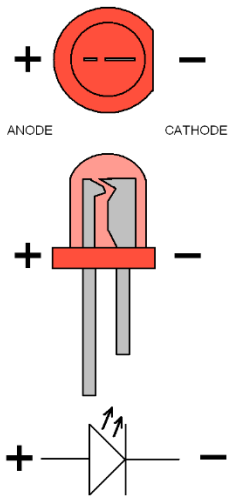
We maken gebruik van een breadboard, weerstanden en LEDs. Hier volgt info over deze componenten:

- Hoe zit een **breadboard** in elkaar?



Bovenstaande tekening geeft aan hoe de **doorverbindingen** in het breadboard zijn voorzien. Als je een weerstand of LED aansluit mag je deze **dus nooit met 2 pootjes in dezelfde doorverbinding** steken. Want de elektronen moeten kunnen bewegen “door” de LED of weerstand.

- Weet dat een LED **een anode en kathode** kant heeft. Het **korte pootje is de kathode** (min kant). Deze hangen we aan de massa (min kant van de voeding). Als je de LED niet juist aansluit brandt deze niet. De kathode wordt ook aangeduid met een **plat vlakje** aan de behuizing. Voor Chinese LED's is dit soms niet waar, meet daarom de LED vooraf door met de multimeter in diode stand!



- De **voorschakelweerstand** van 220 ohm, nodig om de **stroom van de LED te beperken**, kan je herkennen aan zijn kleurencode.
- Wanneer je een **koolstofweerstand (bruin)** hebt ziet deze er zo uit:

Rood – rood – bruin - goud

2 – 2 – 1 = 22 x 10 ohm = 220 ohm 5% tolerantie

- Wanneer je een **metaalfilm** weerstand (meestal **blauw**) hebt heb je 5 banden:

rood – rood – zwart – zwart – bruin.

Dit staat voor 2 – 2 – 0 x 1 = 220 ohm met 1% tolerantie.

- Wanneer de code niet goed zichtbaar is gebruik je best een **multimeter** om de ohmse waarde te achterhalen.
- Waarom 220 ohm?

I = U / R = VCC – U_{LED} / R = 5V – 2V / 220 = 13.6mA (Wet van Ohm)

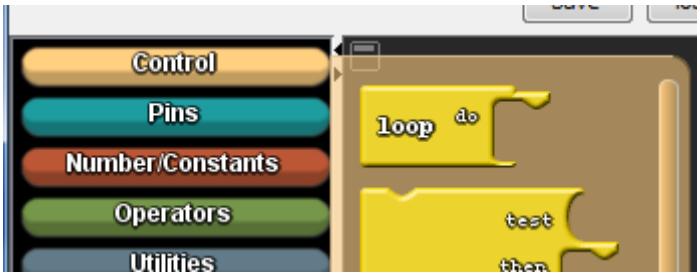
De maximum stroom dewelke onze LED kan verdragen is +/- 20mA.
De maximum stroom die we uit de Arduino pin kunnen trekken is +/- 40mA (zie technische gegevens tabel).
Dus we zitten goed met de gekozen weerstandswaarde.

4. Maak nu de volgende ARDUBLOCK code om de groene LED te laten knipperen:

Wat hebben we nodig voor ons programma?

(merk op dat bij nieuwere versies van Ardublock de kleuren van de blokjes kunnen veranderen!).

In de menu "**controls**" selecteren we de "loop do" en sleuren deze in het programmeer window (als hier nog geen loop staat).



De "loop do" **moet steeds als eerste aanwezig** zijn in jouw programma. Alle code wordt geplaats binnen deze lus (= loop). Het programma wordt dus **oneindig keren uitgevoerd**, zolang de processor spanning heeft. Druk je op de reset knop van de Arduino, dan start het programma terug van het begin van de "loop".

De "**set digital pin**" functie, te vinden in de "**pins**" menu, zorgt ervoor dat je een digitale waarde kan sturen naar de aangegeven uitgangspin van de Arduino. In dit geval sturen we de digitale waarde "1" of "HIGH" naar pin 2. Op deze pin hangt onze groene LED, weet je nog!

De pin waarde kan je vinden in de menu “**number/constants**” en sleur je op de juiste plek achter de “set digital pin” functie. Klik op het blokje en vul de pinwaarde “**2**” in.

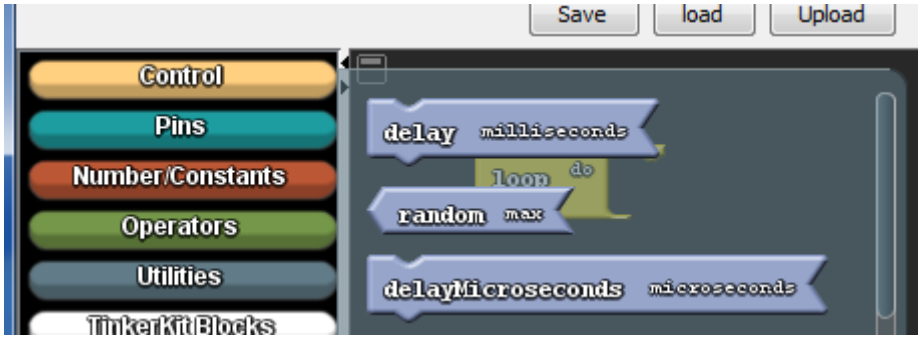
Nu moet je ook nog aangeven of de pin hoog (HIGH = +5V) of laag (LOW = 0V) moet worden. Dit doe je door opnieuw in de “**number/constants**” het juiste blokje te selecteren en te sleuren naar de “set digital pin” functie. Klik hierop en selecteer **HIGH of LOW**.

Merk op **dat alle blokjes op puzzelstukjes** lijken. Wanneer de blokjes niet passen is er vast iets mis! Zoek dit dan even uit. Zo helpt het programma jou de juiste keuzes maken bij het programmeren. Dit zien we ook terugkomen in het programma Scratch.

Wanneer je een **blokje niet meer wil gebruiken** sleur je dit terug in de functie/tools window. Op deze manier verdwijnt dan dit blokje.

Als we de LED willen laten knipperen moeten we deze eerst HIGH maken, dan een **tijdje wachten** en daarna terug LOW maken en weer een tijdje wachten.

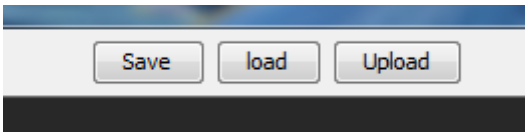
De tijd wordt gemaakt via het blokje “**delay milliseconds**”, dewelke je kan terugvinden in de “**Utilities**” menu. Bij de nieuwste versie van Ardublock zit dit blokje in de “control” menu!



De tijd is uitgedrukt in milliseconde. Dit is 1/1000 van een seconde.

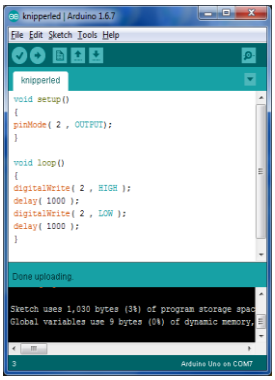
5. Hoe nu het programma testen?

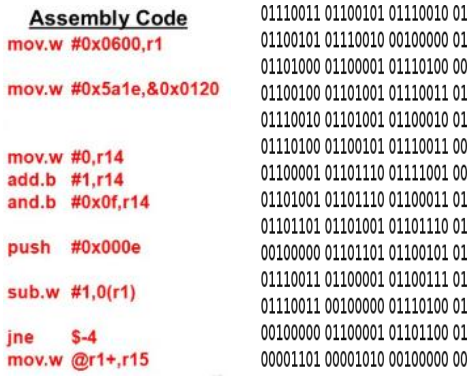
Nadat je het programma hebt geschreven (= gesleurd) kan je dit best even **opslaan** door op de knop "**save**" te drukken. Sla de code op in een mapje op jouw USB stick.



Controleer nog eens of de Arduino schakeling reeds via een USB connector is aangesloten op jouw PC. Check ook even of de naam van het Arduino bordje klopt en de COM poort correct is gekozen (meer info vind je hiervoor terug in hoofdstuk 3).

Druk daarna op de knop "**upload**". Nu wordt het programma in de IDE omgeving nagekeken op fouten. Daarna worden de **tekeningen** omgezet naar **c-code** die getoond wordt in de Arduino IDE omgeving. Tenslotte wordt de code gecompileerd (in begrijpbare microcontroller taal omgezet = **machine code**) en daarna doorgestuurd naar de Arduino UNO **microcontroller** in de vorm van een HEX-file (allemaal 0-en en 1-en).





Als alles goed is gegaan moet onderaan het code venster in de Arduino IDE omgeving de tekst "**Done uploading"** staan.

6. Hoe ziet de code er in de visuele omgeving **flowcode** uit?

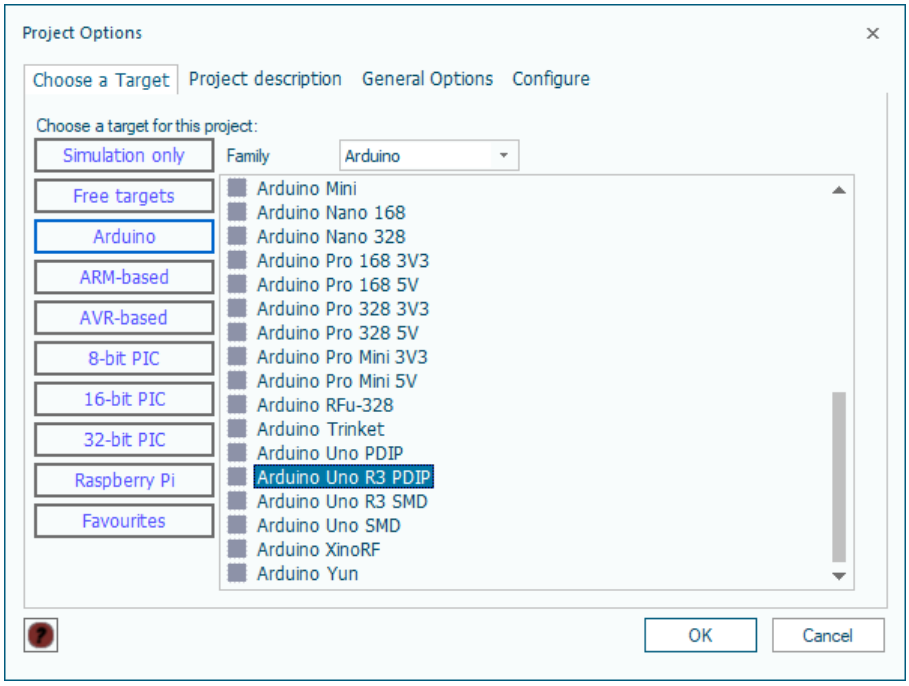
Het flowcode programma kan je gratis voor 30-dagen downloaden van de Matrix Multimedia website: https://www.matrixtsl.com/flowcode/download/

Vergeet ook niet van voor Arduino de **juiste compiler extra te downloaden** en te installeren alvorens te beginnen met het programmeren: **AVR Arduino toolchain**

Na het installeren en het aanduiden van de **licentie** dewelke jij wenst (free of betalend zoals op school) start je het programma.

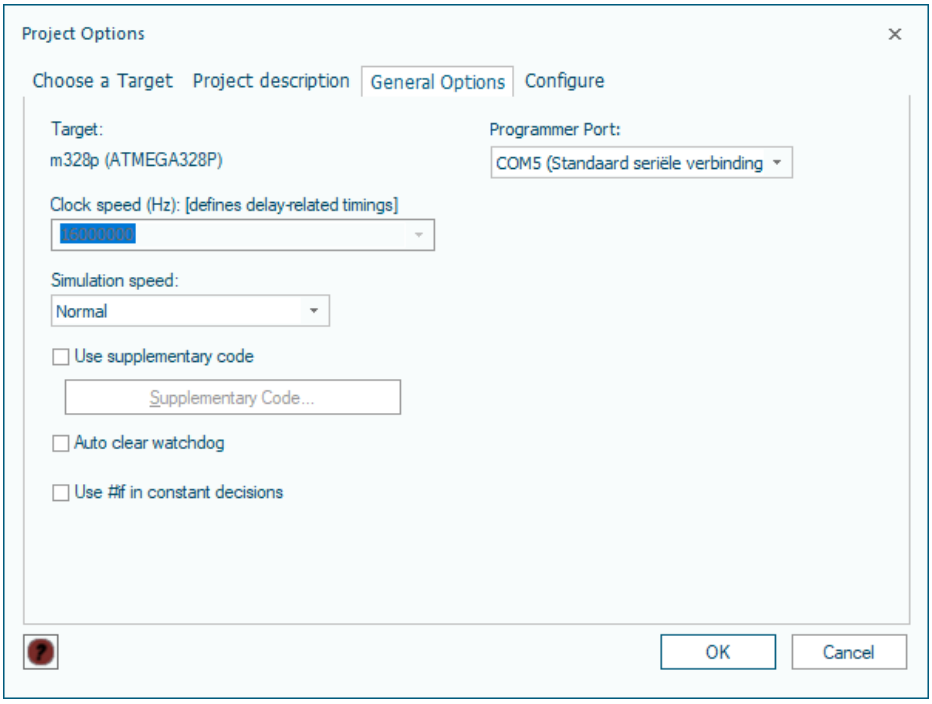
Kies voor een **nieuw project**.

Selecteer als **Target** jouw **Arduino UNO R3**



Plug jouw Arduino bordje in en zorg dat de **driver correct is geïnstalleerd**.

Klik nu op “**General Options**” en **selecteer hier de juiste COM poort** van jouw Arduino (eventueel via de device manager of windows+pauze opzoeken)

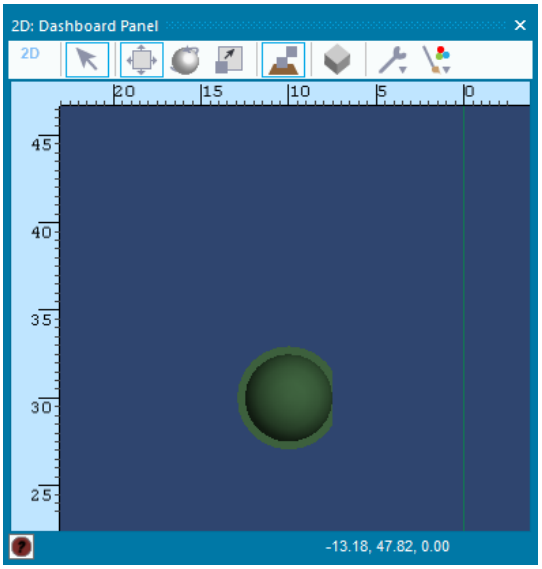


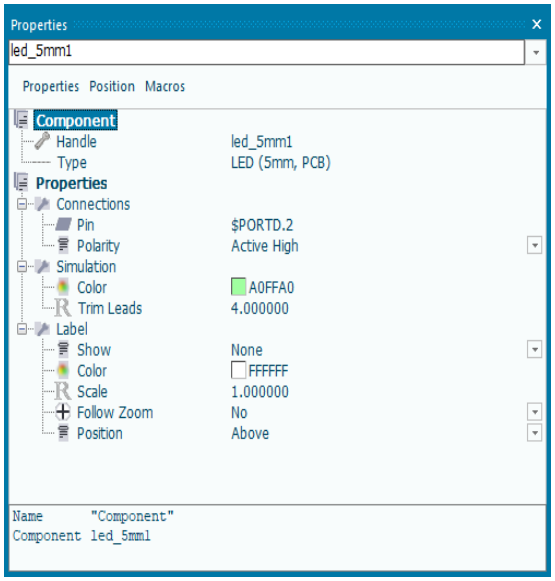
Merk op dat de snelheid vast ingesteld staat op 16MHz.

De chip is de MEGA328P.

Open nu het nieuwe project.

We willen een LED laten knipperen. Daarom gaan we als **OUTPUT een 5mm LED** selecteren uit de menu “Outputs”. Sleur deze naar het **2D window** (of het Dashboard Panel).

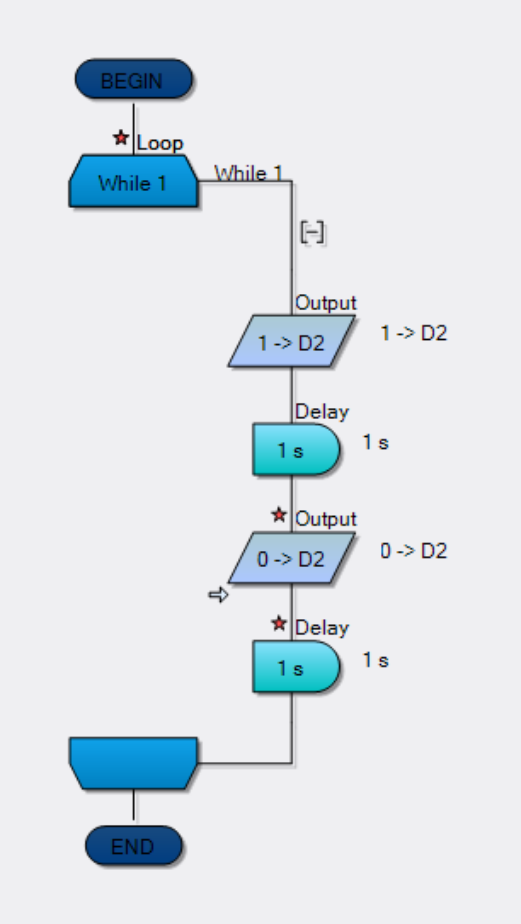


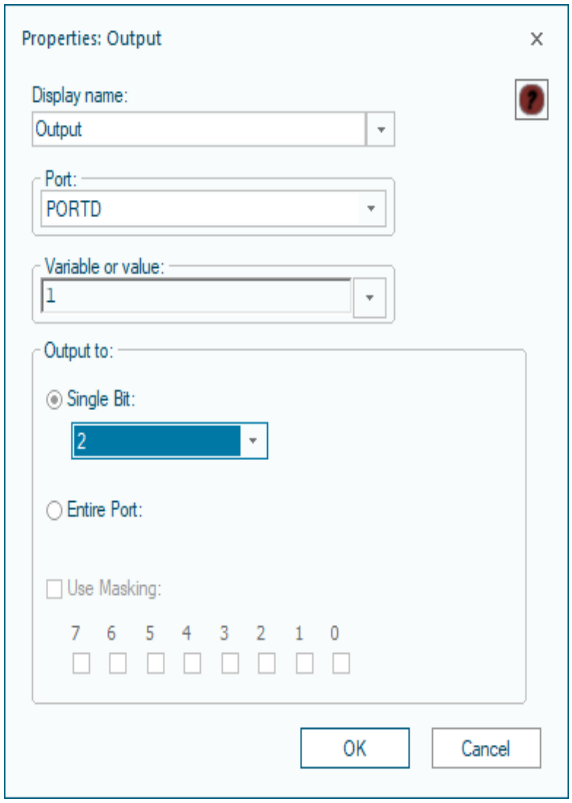


In de **properties** van de LED (rechts klikken op de LED) stellen we de pin in op **digitale pin 2.**

Sleur nu jouw flowcode blokken op het scherm.

Stel in de **Output** blok (op dubbelklikken) het volgende in: Port D.2 = 1 of 0





Druk op "**Run**" (blauwe pijl) om de LED te zien flikkeren tegen 1 seconde.

Sluit op jouw Arduino een LED aan op pin D2 via een voorschakelweerstand.

Klik nu "**compile to chip**" om de Arduino code te downloaden in jouw UNO.

Als alles goed zit moet nu de LED knipperen tegen 1 seconde.

Tip:

Flowcode heeft alleszins als **extra voordeel** dat je de code **stap voor stap kan simuleren**.

Je kan ook de flowcode **omzetten naar C-taal** door op de knop **Macros -> Show as C code** te klikken.

Selecteer dan Main. Nu kom je uit op een laag waarbij je een Flowcode achtige C-code terugvind met hen eigen subroutines. Dit is echter niet bruikbaar voor ons.

Hoe ziet de **c-code** eruit voor 1 knipper LED?

Om de overgang van ARDUBLOCK naar c-code te maken is het goed dat we de gegenereerde c-code eens bekijken. Bedoeling is dat we op een bepaald moment geen ARDUBLOCK meer gaan gebruiken maar enkel nog c-code gaan schrijven.

knipperled

```
void setup()
{
  pinMode( 2 , OUTPUT);
}

void loop()
{
  digitalWrite( 2 , HIGH );
  delay( 1000 );
  digitalWrite( 2 , LOW );
  delay( 1000 );
}
```

Wat zien we in bovenstaande code?

Je kan deze opsplitsen in 2 delen:

- De "**setup**" functie: hier vullen we in hoe we onze pinnen willen gaan gebruiken. We gebruiken hiervoor de **pinMode (pin nummer, INPUT/OUTPUT)** opdracht. Onze LED hangt aan een **OUTPUT** pin en is aangesloten op digitale pin 2. Deze functie **wordt 1-malig uitgevoerd.**

```
void setup()
{
  pinMode( 2 , OUTPUT);
}
```

  - Merk op dat je in de code **de juiste hoofdletters en kleine letters typt** zoals in het voorbeeld. Anders herkent de Arduino IDE de functies of opdrachten niet tijdens het compileren. Je kan dit reeds tijdens het typen merken als de code niet in de juiste kleur komt te staan. Dit leer je door veel te programmeren en ervaring op te doen. Arduino geeft een error tijdens het compileren wanneer er iets niet juist is getyped.
  - Het woord "**void**" betekent "leeg" en geeft aan dat de functie niets terug geeft nadat deze is uitgevoerd.
  - Na elke functie staan er **( )**. Hier tussen zit op dit moment geen code.
  - Merk op dat alle programmeerlijnen van de functie SETUP tussen **{ }** staan. Zo groepeer je makkelijk wat bij elkaar hoort.
  - Op het einde van elke lijn staat ook een "**;**".
  - Na een } moet je geen ";" zetten.

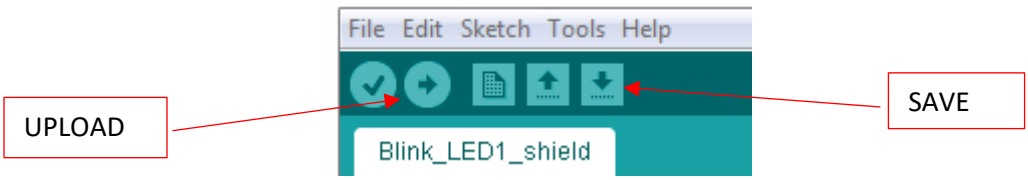
- De “**loop**” routine bevat de code die **herhaaldelijk wordt uitgevoerd** totdat je op reset drukt of de spanning afzet van de controller.

```
void loop()
{
  digitalWrite( 2 , HIGH );
  delay( 1000 );
  digitalWrite( 2 , LOW );
  delay( 1000 );
}
```

  - In dit geval willen we de LED op pin 2 eerst laten branden. De opdracht hiervoor is **digitalWrite(nummer pin, HIGH/LOW)**. De pinnummer is 2 en we kiezen **HIGH** om de LED **aan** te zetten (dus **+5V** naar toe te sturen).
  - Willen we de LED laten knipperen, dan moeten we deze aan doen, een tijdje wachten en daarna weer uitdoen. Wachten kan met de opdracht **delay(nummer)**. De nummer geeft het aantal milliseconden aan. Dus in dit voorbeeld wacht de microcontroller 1000ms = 1 seconde. Indien je nog sneller wil is er ook de **delayMicroseconds(nummer).**
  - Daarna moeten we de LED1 weer uitdoen door digitalWrite(2 **,LOW**) te sturen. Hiermee zetten we de uitgang van de processor aan de **massa.**

  In de Arduino IDE omgeving kan je makkelijk de code aanpassen.

- Sla daarna de code op door op de “**save**” knop te drukken. Kies altijd een naam dewelke iets vertelt over de code, voorbeeld “blink_LED1”.
- Daarna druk je op de knop “**upload**” om de code in jouw Arduino te laden.



- Wanneer alles goed is gegaan moet de LED op pin 2 nu knipperen met een tempo van 1 seconde op de Arduino.

7. **Tips om betere c-code** te maken:

   Tip 1: Je kan jouw **programma overzichtelijker** maken door de pin nummers te veranderen in de naam/functie van het elektronica component.

   We doen dit met een "**#define**".

   Met de "**#define**" maken we o.a. **variabelen** (= geheugen plaats) aan (vb LED1) en geven deze direct een pin nummer mee (in dit geval pin 2 waar LED1 aanhangt). Voordeel is dat je vanaf de volgende regel in de code niet meer moet werken met pin nummers maar met het equivalent , namelijk LED1. Zo blijft de **code duidelijker** om te lezen.

   Je kan deze code makkelijk aanpassen in de Arduino IDE. Bij ons voorbeeld voor 1 LED krijg je dan het volgende. Test deze code zeker eens uit.

knipperled_met_define

```
#define LED1 2
#define LED2 3

void setup()
{
  pinMode( LED1 , OUTPUT);
  //indien je LEDs niet gebruikt met je deze op nul zetten!
  pinMode( LED2 , OUTPUT);
  digitalWrite (LED2, LOW); //niet gebruikte LED op nul gezet
}

void loop()
{
  digitalWrite( LED1 , HIGH );
  delay( 1000 );
  digitalWrite( LED1 , LOW );
  delay( 1000 );
}
```

Tip2 : **Een manier om niet met HIGH en LOW te moeten werken** in jouw code is gebruik te maken van een variabele “**state**”. Bij de defines geef je eerst aan dat state = HIGH. Daarna kan je overal in jouw code HIGH vervangen door de variabele “**state**”. Waar LOW staat gebruik je “**!state**” . De “**!**” staat voor de **invertering** (NOT poort) van de bit inhoud van state. Als deze “1” is wordt deze nu “0”. Zie hiervoor de volgende code. Test ook deze code uit.

knipperled_met_not §

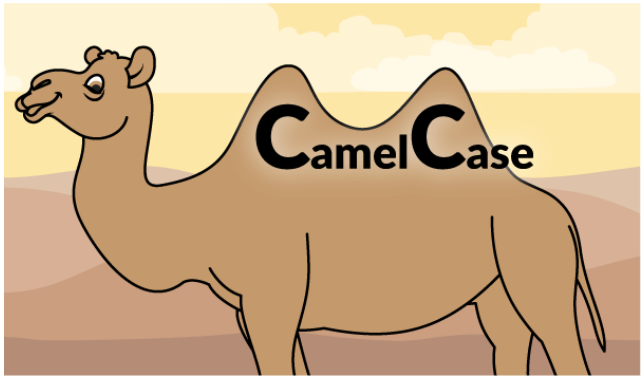
```
#define LED1 2
#define state HIGH // HIGH = 1

void setup()
{
  pinMode( LED1 , OUTPUT);
}

void loop()
{
  digitalWrite( LED1 , state );
  delay( 1000 );
  digitalWrite( LED1 , !state );//! = not poort
  delay( 1000 );
}
```

Je kan “!” ook door het woordje “not” vervangen.

Tip 3: Maak een **verschil tussen hoofdletters en kleine letters**!

Wanneer je code schrijft dan is het belangrijk dat je deze leesbaar houd.
Hoe kunnen we dit vergemakkelijken?

- Zorg dat de constanten in HOOFDLETTERS worden geschreven, vb LED, SWITCH, POWER, ….
- Zorg dat de variabelen in KLEINE LETTERS worden geschreven, vb teller, ldr, ….
- Wanneer een variabele uit meerdere woorden bestaat moet je **CamelCase** toepassen, vb ChronometerTeller, PwmMotorLinks …



- Bij Arduino gebruiken ze in de functies ook een vorm van **CamelCase,** vb digitalWrite(), analogRead() … Alleen is nu de eerste letter ook een kleine letter.
- Bij een object (gaan we verder in de cursus zien, bv Serial.begin) mag je de eerste letter wel met hoofdletter schrijven.

(merk wel op dat niet op elke plek in deze cursus al is rekening gehouden met de bovenstaande regels 😉
Die kwamen mij pas achteraf door een cursist ter oren, waarvoor zijn dank 😊)

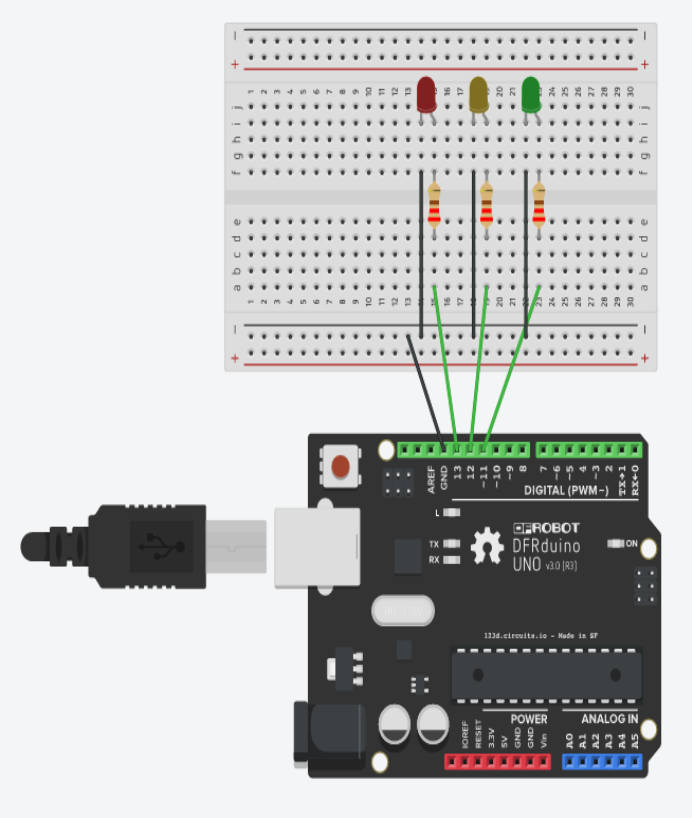
8. Uitdagingen met LEDs:

1. Zorg dat de 2 LEDs ombeurten knipperen.
2. Maak een looplicht met 5 LEDs, aangesloten op de arduino.
3. Maak een verkeerslicht
4. Maak een dobbelsteen met 7 LEDs

Deze code kan je nog steeds in ARDUBLOCK maken, maar het is nog beter wanneer je dit **rechtstreeks in c-code programmeert**. Probeer beide programmeertalen uit.

Save steeds jouw code onder een nieuwe nuttige naam en test deze uit op de hardware.

Voorbeeld van een verkeerslicht: de LEDs zijn aangesloten op pin 11, 12 en 13.



Merk op dat je de code **altijd nog efficiënter** kan schrijven. Zo kan je de dobbelsteen ook in code verkorten door gebruik te maken van for lussen en if then else te gebruiken. Ook het commando “random” kan de dobbelsteen een nog echter gevoel geven van kansberekening.

We gaan deze manier van programmeren aanpakken bij de knop uitdagingen 😉

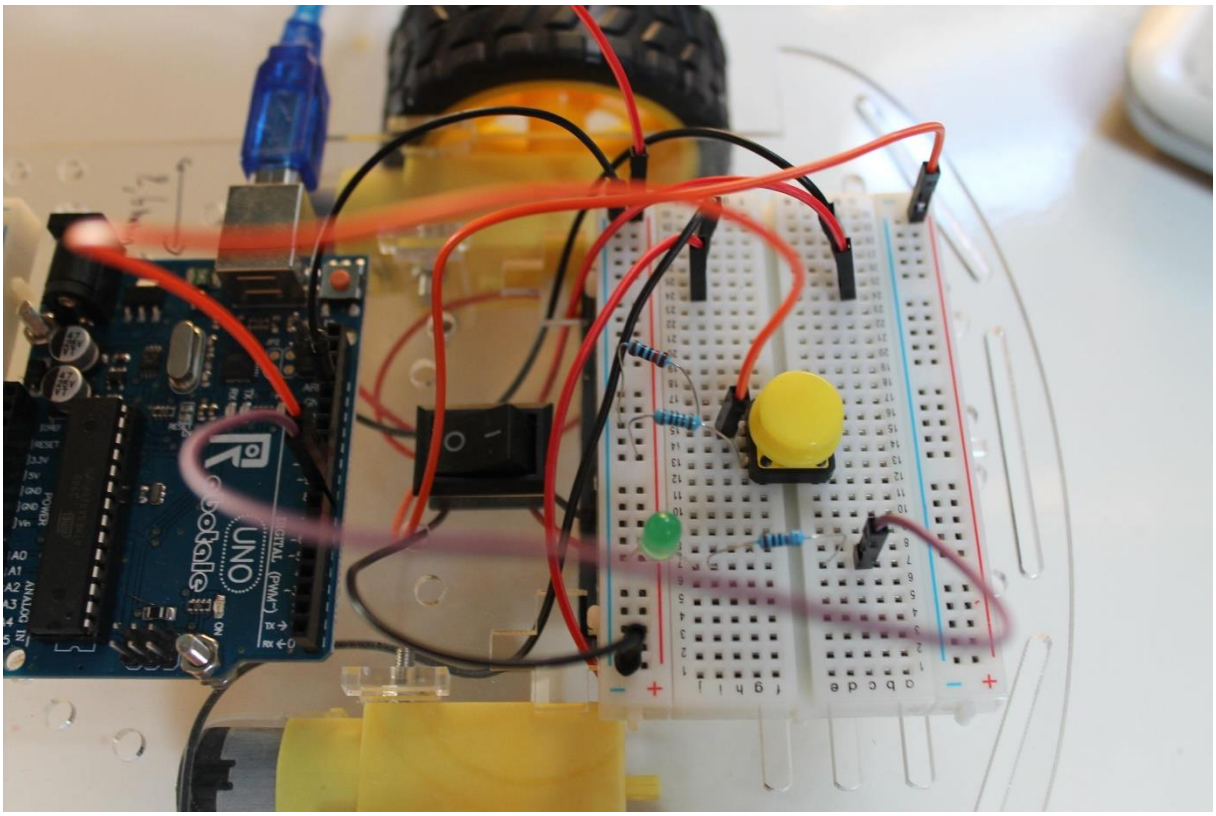
7. Schakelaars inlezen

Doel: Nu we LEDs kunnen aansturen willen we deze graag alleen laten aan gaan wanneer we op een schakelaar drukken.

Benodigdheden:

- Arduino Uno
- 1 LED
- 1 x 220 ohm weerstand
- 1 schakelaar
- 1 x 1K weerstand
- 1 x 10K weerstand

1. Finale opstelling:

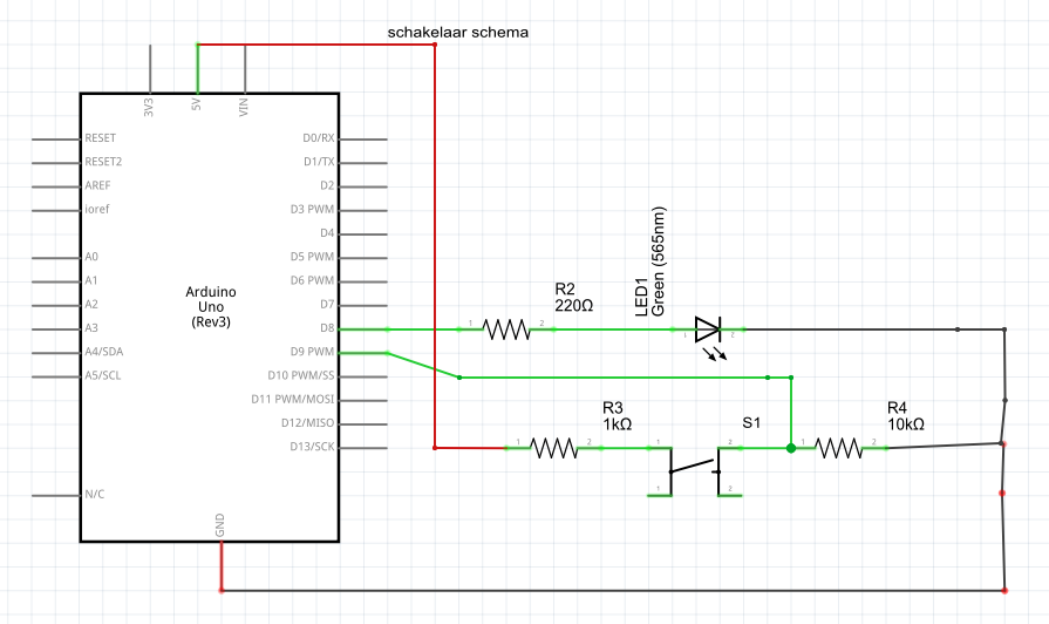


Vermits de waarde van een knop “0” of “1” is kunnen we dus deze gebruiken als **digitale input**.

De knop hangen met één zijde aan de +5V, terwijl de andere kant aan de Arduino pin hangt. Wanneer we de knop niet zouden indrukken dan meet de Arduino een ongedefinieerde waarde (open ingang). Dit is een oncontroleerbare situatie en die willen we zeker voorkomen.

Daarom hangen we aan de Arduino kant van de knop een weerstand van 10K naar massa. Is de knop niet ingedrukt, dan zal er nu een “0” gelezen worden.

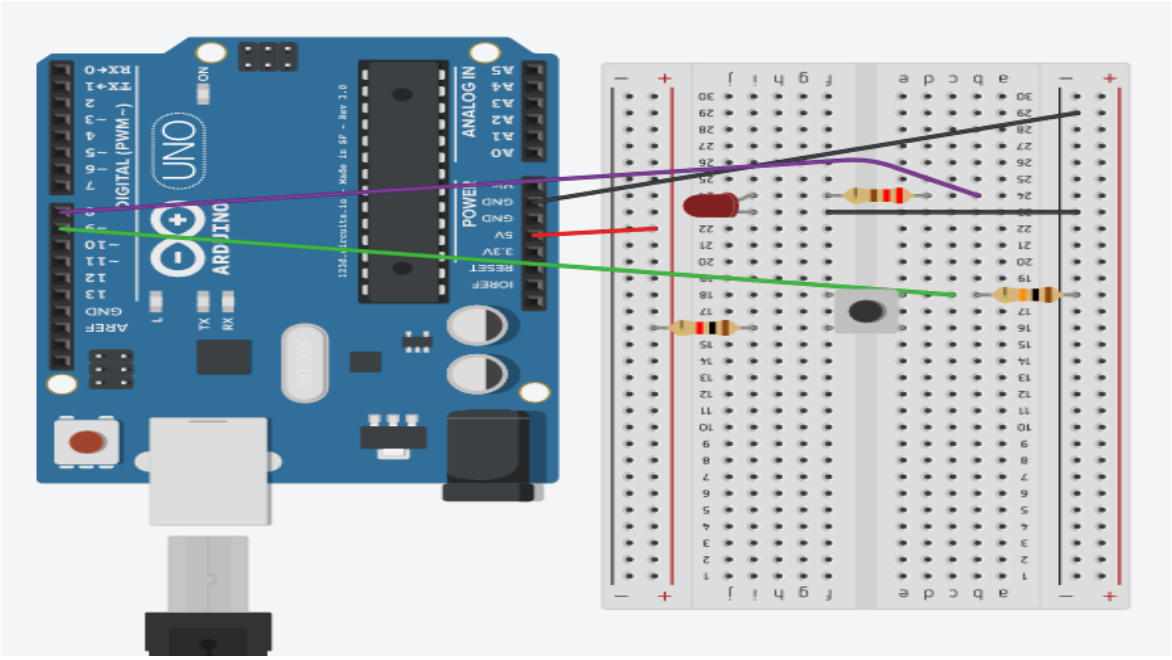
2. Schema:



De 10k weerstand is dus gebruikt als **pull-down weerstand**, want we willen een **hoog actieve schakelaar** maken. Wanneer we drukken krijgen we een “1”.

De 1k weerstand staat er om de **chip ingang pin 9 te beschermen** wanneer er per ongeluk een output is geprogrammeerd i.p.v. een ingang. Wanneer je dan een “0” op de output zou sturen ga je de +5V via S1 en de interne transistor naar massa trekken. Wanneer de 1k er niet zou staan zou er geen stroombeperking zijn en zouden we dus de chip pin gewoon stuk doen.

3. Breadboard opstelling:



Bouw nu zelf het schema. **Trek steeds de USB kabel uit wanneer je de schakeling bouwt**. Controleer dan of alles goed zit. Daarna pas terug de USB kabel insteken en de schakeling spanning geven.

Info over de nieuwe gebruikte componenten:

- Weerstanden:

1K metaalfilm weerstand = bruin – zwart – zwart – bruin (1-0-0 x 10)

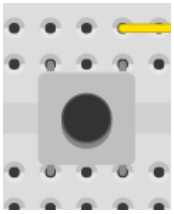
Wanneer dit een koolstofweerstand zou zijn had de 1K weerstand de kleuren bruin-zwart-rood.

10K metaalfilm weerstand = bruin – zwart – zwart – rood (1-0-0 x 100)

Wanneer dit een koolstofweerstand zou zijn had de 10k weerstand de kleuren bruin-zwart-oranje

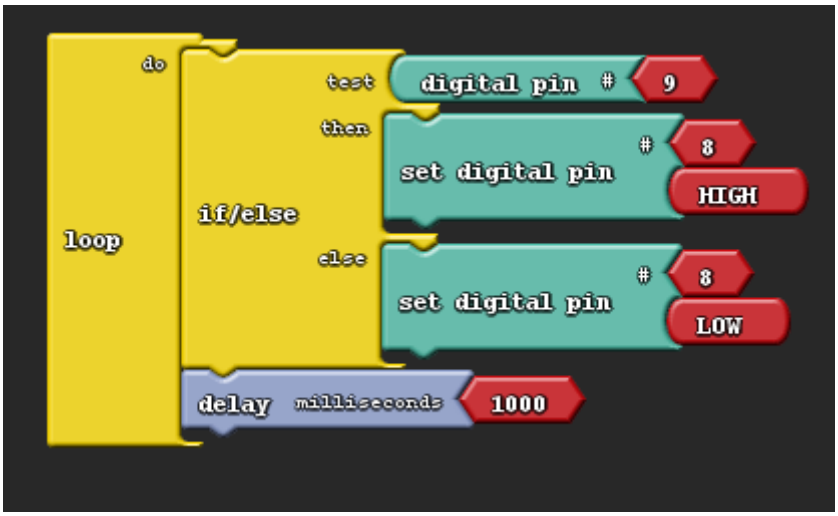
- Knoppen:

Merk op dat de knop in 1 richting doorgeeft en in de andere richting niet.



Neem een multimeter in piep stand en zoek uit hoe de knop doorgeeft. Houd hier dan rekening mee in de opbouw van de schakeling.

4. Schrijf nu de volgende ARDUBLOCK code om de LED aan te laten gaan, enkel wanneer er op de knop wordt gedrukt:



Wat zien we in de code?

Na de “loop” control plaatsen we een “**if then else**” control. Bedoeling hierbij is dat we eerst een bepaalde voorwaarde = test gaan controleren, alvorens we verdere code gaan uitvoeren.

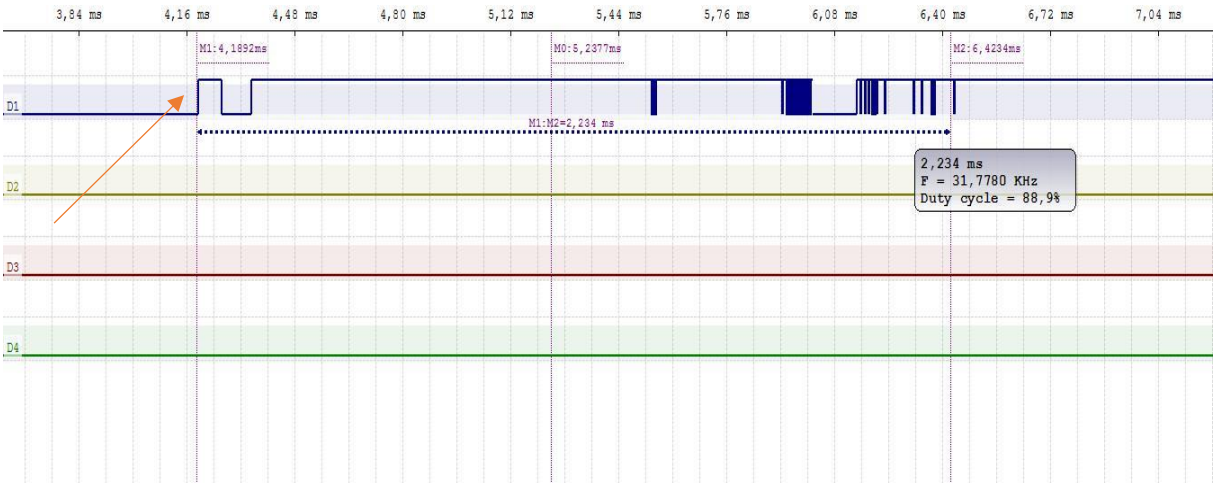
Als de voorwaarde “TRUE = waar = 1” is, dan wordt de code van “**then**” uitgevoerd. Als de voorwaarde “FALSE = onwaar = 0” is, dan wordt de code van “**else**” uitgevoerd.

De voorwaarde is in dit geval de waarde die we inlezen van de schakelaar, aangesloten op digitale pin 9. Als **pin 9 = “1“** (schakelaar is ingedrukt), dan zal pin 8 = “1” worden, dus de **LED gaat aan**. Als de knop niet is ingedrukt zal de ingelezen waarde “**0**” zijn en zal pin 8 = “0” worden. Nu gaat de **LED weer uit**.

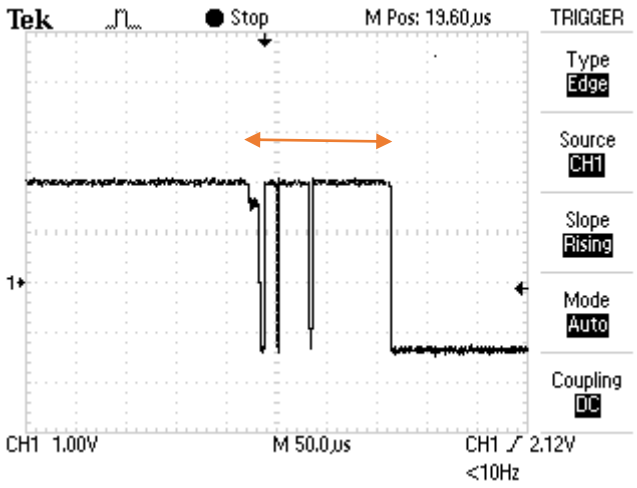
De schakelaar ondervindt “**dender**”. Dit is een mechanisch verschijnsel waarbij wanneer je op de knop drukt de schakelaar in feite meerdere keren open en toe gaat alvorens hij in de bedoelde stand terecht komt. De microcontroller is zo snel dat hij al deze toestanden kan inlezen. Zo krijg je makkelijk een onstabiel effect.
We kunnen “**antidende**r” toevoegen door een bepaalde **vertraging** na het inlezen van de knop te voorzien. In dit geval hebben we 1 seconde in de code voorzien, maar 100ms is normaal al meer als voldoende.

Upload deze code en test jouw hardware.

Dender kan je makkelijk van een schakelaar meten door er een logic analyzer of geheugen scope op aan te sluiten.



Deze meting is met een **logic analyzer** gedaan. Zie uitdagingen om het zelf eens uit te proberen. Je kan zien dat de hoog actieve knop ingedrukt wordt (laag naar hoog), maar dat er daarna door de mechanische eigenschappen van de schakelaar nog verschillende keren pulsen te meten zijn.
De microcontroller draait aan 16MHz (periode T = 1/f = 1/16000000 =62.5ns) en zal dus al deze pulsen kunnen meten. In de software lijkt het dan alsof de knop meerdere malen wordt ingedrukt. Zo meten we teveel waardes.



Ander voorbeeld van dender.
Om dit voor de microcontroller niet zichtbaar te maken gaan we, na het indrukken van de knop een delay toevoegen, minstens zolang als de dender gaat duren.

Een **andere methode om antidender te voorzien** is i.p.v. 1 grote delay tijd te programmeren de volgende code te voorzien voor een actief hoge schakelaar:

```
while(digitalRead(SWITCH) == 1)
{

        delay(10);

}
```

Nu wordt ook de processor even tegen gehouden, maar wordt er **sneller getest** of de dender al voorbij is.

5. Hoe ziet de c-code eruit van dit programma?

lees_schakelaar.ino

```
void setup()
{
pinMode( 8 , OUTPUT);
pinMode( 9 , INPUT);
}

void loop()
{
  if (digitalRead( 9))
  {
    delay( 100 );//antidender
    digitalWrite( 8 , HIGH );
  }
  else
  {
    digitalWrite( 8 , LOW );
  }

}
```

In de code zie je de **if(voorwaarde) else** opdracht staan. De microcontroller kijkt eerst na of de voorwaarde **WAAR** is, dan voert hij de code uit tussen de {} van de **if**. Indien de **voorwaarde NIET WAAR** is wordt de code tussen {} van de **else** uitgevoerd.

Wanneer er in dit geval op de schakelaar gedrukt wordt is deze pin ingang laag. Via **digitalRead(pin)** komen we dit te weten. Als de ingelezen waarde hoog is, dan is aan de voorwaarde voldaan want 1 == HIGH. De delay van 100ms vormt de **antidender**.

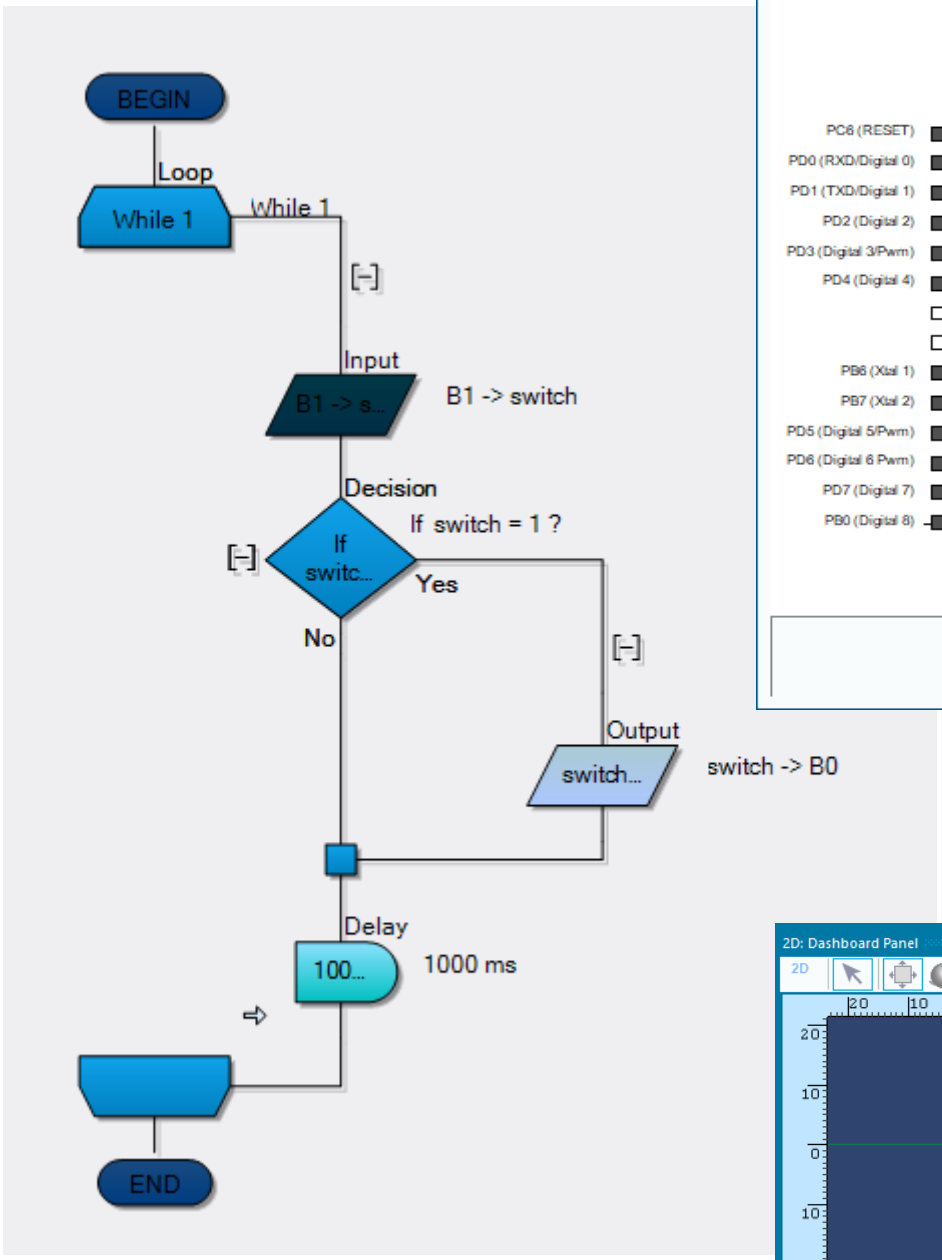
c-code voor knop in te lezen met efficiëntere antidender methode:

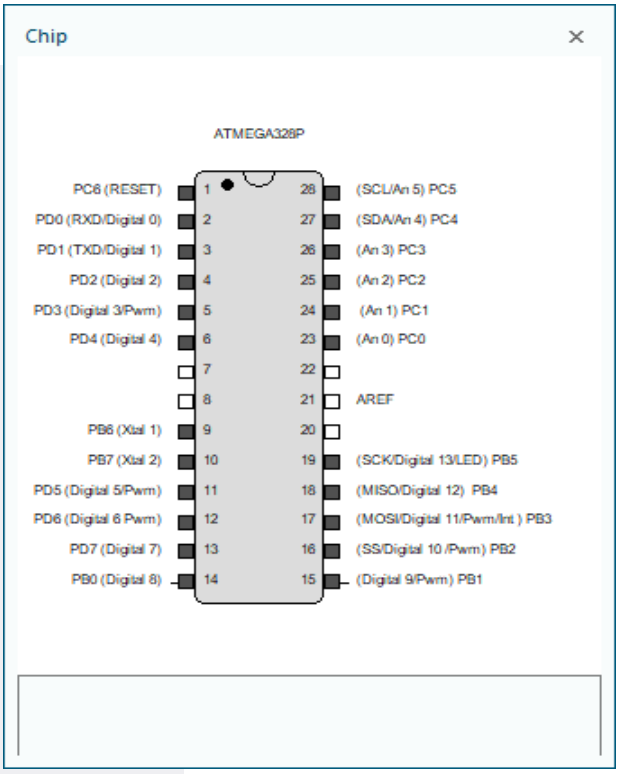
lees_schakelaar_effici_nter_antidender

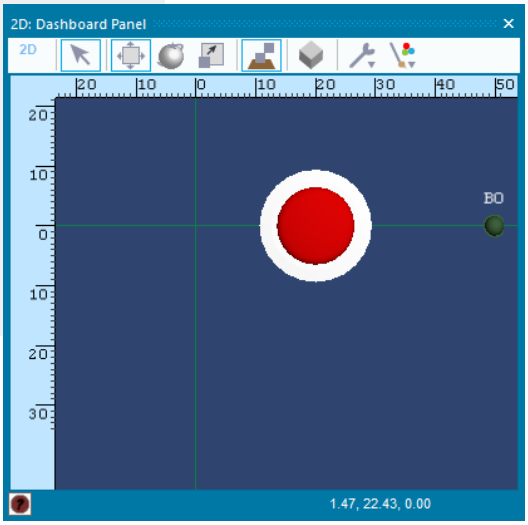
```
void setup()
{
  pinMode( 8 , OUTPUT);
  pinMode( 9 , INPUT);
}

void loop()
{
  if (digitalRead( 9))
  {
    while(digitalRead(9) == 1) //antidender
    {
      delay(10);
    }
     digitalWrite( 8 , HIGH );
     delay(1000); //hoelang de led brandt
  }
  else
  {
     digitalWrite( 8 , LOW );
  }

}
```

Test ook deze code uit. Merk op dat de LED pas brandt als je de knop hebt losgelaten.

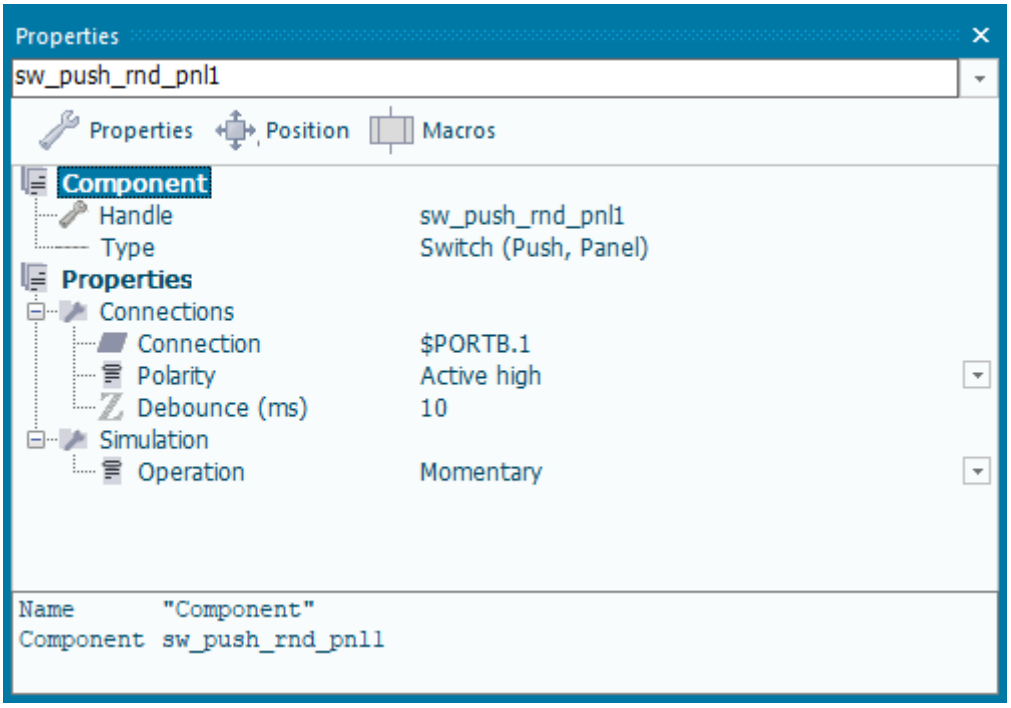
Hoe maken we deze code in **flowcode?**







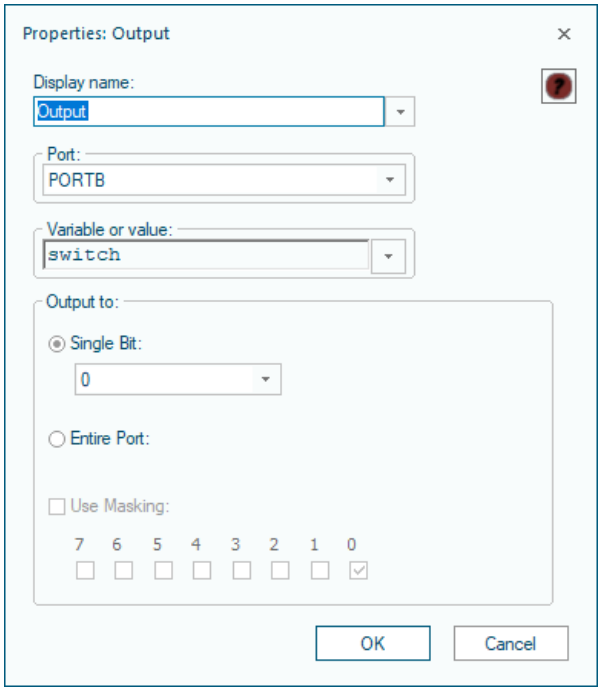
Met deze code kunnen we een schakelaar inlezen. Op B1 (D9 op de Arduino) lees je de schakelaar in en stop je de waarde in een booleaanse (0 of 1) variabele “switch”. De waarde toon je daarna op de LED op pin B0 (D8 op de Arduino).



Zo stel je de knop in op B1. Merk op dat flowcode zelf een mogelijkheid voorzier om de Debounce tijd (antidender) in de stellen in ms.

Je kan ook tussen een actief lage en hoge knop kiezen.

Tot slot kan je kiezen tussen een “latch” (aan/uit schakelaar) of “momentary” (drukknop).



De LED terug instellen op B0.

6. Programmeer tips:

Tip 1: zolang als je 1 knop moet vergelijken met “1” of “0” als voorwaarde, heb je geen andere operatoren nodig.

Stel dat je nu eerst de knop wil inlezen in een **variabele** (= geheugenplaats) en deze dan vergelijken met een andere waarde, dan heb je wel **logische operatoren** nodig.

De “==” is een logische operator en staat voor “is gelijk aan”.

Op die manier zou onze code er dan als volgt kunnen uitzien:

lees_schakelaar_met_operator

```
//code schakelaar met operator lezen
#define KNOP 9
#define LED 8
bool schakelaar = 0;//op begin op 0 zetten

void setup()
{
  pinMode( LED , OUTPUT);
  pinMode( KNOP , INPUT);
}

void loop()
{
  schakelaar = digitalRead( KNOP);

  if (schakelaar == 1)
  {
    delay(100); //antidender
    digitalWrite( LED , HIGH );
    delay( 1000 ); //tijd dat de led aan is
  }
  else
  {
    digitalWrite( LED , LOW );
  }
}
```

Eerst wordt een variabele “schakelaar” aangemaakt van het **type “bool**”. Dit wil zeggen dat straks in de code je een “1” of een “0” kan schrijven en uitlezen uit deze **geheugenplaats** met de naam “schakelaar”. We zetten deze dadelijk op “0” zodat we weten wat er na de reset van de Arduino in de geheugenplaats bevindt.

Nu lezen we de “knop” in en stoppen deze waarde in de “schakelaar”.

Daarna wordt de inhoud van “schakelaar” **vergeleken** met de waarde “1” via de “==”.

Merk op dat “=” overeenkomt met een waarde toekennen aan een variabele terwijl “==” een logische vergelijking is tussen 2 waardes. Als de vergelijking klopt, dan komt er een “1” uit, anders een “0”.

Er zijn nog andere **logische operatoren** die we ook **als voorwaardes** kunnen gebruiken om te **vergelijken**. Zie daarom onderstaande tabel:

Logische operator	Omschrijving
>	groter dan
>=	groter dan of gelijk aan
<	kleiner dan
<=	kleiner dan of gelijk aan
!=	ongelijk aan
==	gelijk aan

Wiskundige operatoren kunnen ook gebruikt worden in de Arduino code, vandaar deze tabel:

Wiskundige operator	Omschrijving
+	optellen
-	aftrekken
=	oplossen
*	vermenigvuldigen
Pow(base, exp)	machtsverheffen
/	delen
%	neem de modulus
>>	verschuif 1 bit naar rechts
<<	verschuif 1 bit naar links

Voorbeeld van modulus (rest van de deling):

Met integers als variabelen: 9/2 = 4

9 % 2 = 1 (want we kunnen 9 delen door 2, dan hebben we 8 op gebruikt, dus er blijft 1 over dat we niet kunnen delen).

Met een float variabele: 9/2 = 4.5

Opdracht: maak oefeningen in Arduino waarbij deze **logische operatoren worden uitgetest**.

Opmerking: maak bij de integers deel oefening alle variabelen A, B, result als integer. Wil je graag met float werken, dan moeten alle variabelen van het type float zijn.

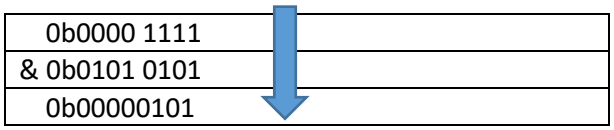
Als je wil delen en je wenst een komma getal te bekomen, dan moet je delen door een komma getal, ook al heb je een geheel getal in de noemer ! De microcontroller moet weten dat je met komma's wil werken!

Voorbeeld: float t = 0; t = 1 / 2.0 ; → t = 0.5

Er bestaan ook nog de volgende **logische operatoren**:

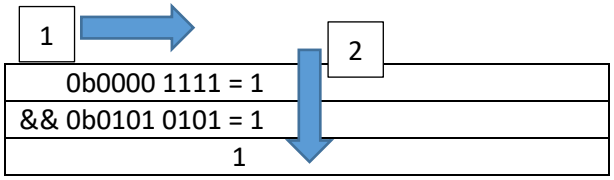
Wiskundige operator	Omschrijving
&	Bitgewijs EN
|	Bitgewijs OF
^	Bitgewijs EXOF
&&	Bytegewijs EN
||	Bytegewijs OF
^^	Bytegewijs EXOF
>> n	n bits schuiven naar rechts
<< n	n bits naar links schuiven
!	Not (enkel bit 0 geïnverteerd)
~	Complement (alle bits worden geïnverteerd)

Vb bitwise: bit per bit logisch uitwerken



Vb bytewise: eerst de bytes apart bekijken (1 bit of meer = “1”, anders “0”)

Daarna bitwise het resultaat uitwerken



Vb bitmaskering (bitwise techniek toepassen om 1 of meerdere bits uit te filteren)

Opdracht:

1. Maak een arduino oefening waarbij je kan aantonen hoe **bitwise / bytewise werkt**.
2. Maak een oefening waarbij **bitmaskering** wordt aangetoond

Opmerking: je kan de functie Serial.print(x,BIN); gebruiken om bits te tonen op de monitor. Verder in de cursus wordt meer over de Serial monitor vertelt. Handig bij debug 😊

Tip2: Hoe een **actief lage schakelaar** maken zonder externe weerstanden?

Ofwel hangen we de pinnen met een **externe 1K pull-up weerstand** aan de +5V. Wanneer we dan drukken op de knop wordt de ingang aan massa getrokken. Laten we deze los, dan hangt de ingang aan de +5V.

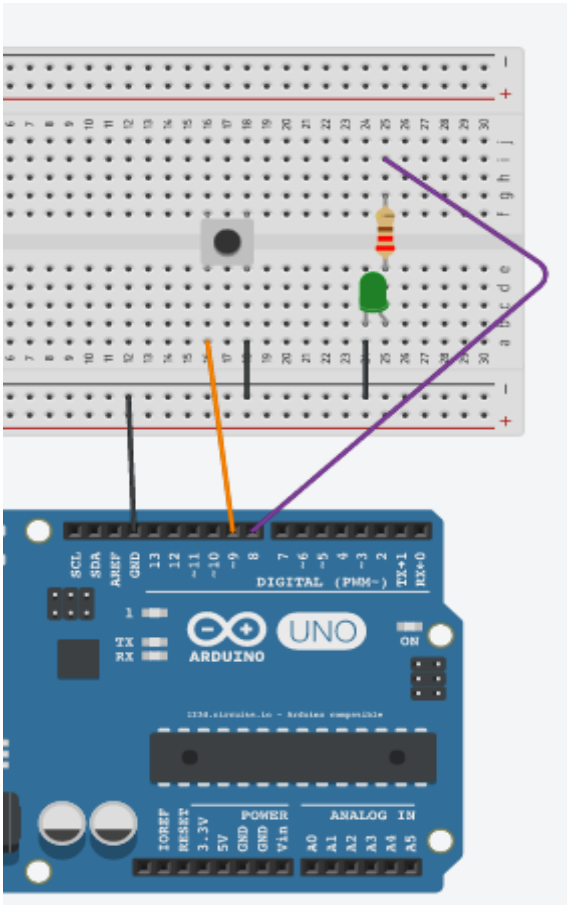
Handig is nu om te weten dat in de processor **reeds pull-up weerstanden zitten** op de poorten. Hoe kan je deze activeren? Door de volgende code te gebruiken:

Als je de functie **pinMode(nummer,INPUT_PULLUP)** gebruikt zal er **intern** in de microcontroller een pull-up weerstand worden voorzien naar +5V op de aangegeven pin.

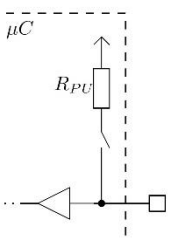
Dan krijg je bijvoorbeeld volgende code wanneer je een knop op pin 9 als laag actieve schakelaar voorziet:

lees_schakelaar_laag_actief§

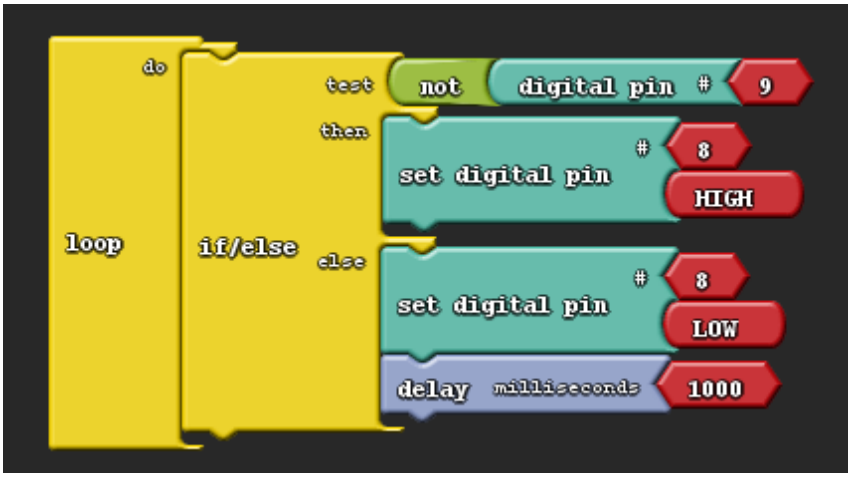
```
//code schakelaar met operator lezen
#define KNOP 9
#define LED 8
bool schakelaar = 0;//op begin op 0 zetten

void setup()
{
  pinMode( LED , OUTPUT);
  pinMode( KNOP , INPUT_PULLUP);//Laag actief, pull-up intern
}

void loop()
{
  schakelaar = digitalRead( KNOP);

  if (schakelaar == 0) //laag actief
  {
    delay( 100 );//antidender
    digitalWrite( LED , HIGH );
    delay(1000);//tijd dat LED aan is
  }
  else
  {
    digitalWrite( LED , LOW );
  }
}
```



In deze code gebruiken we geen NOT en gaan we de schakelaar gewoon checken of er een “0” aanwezig is als er gedrukt wordt. Dan brandt de LED.

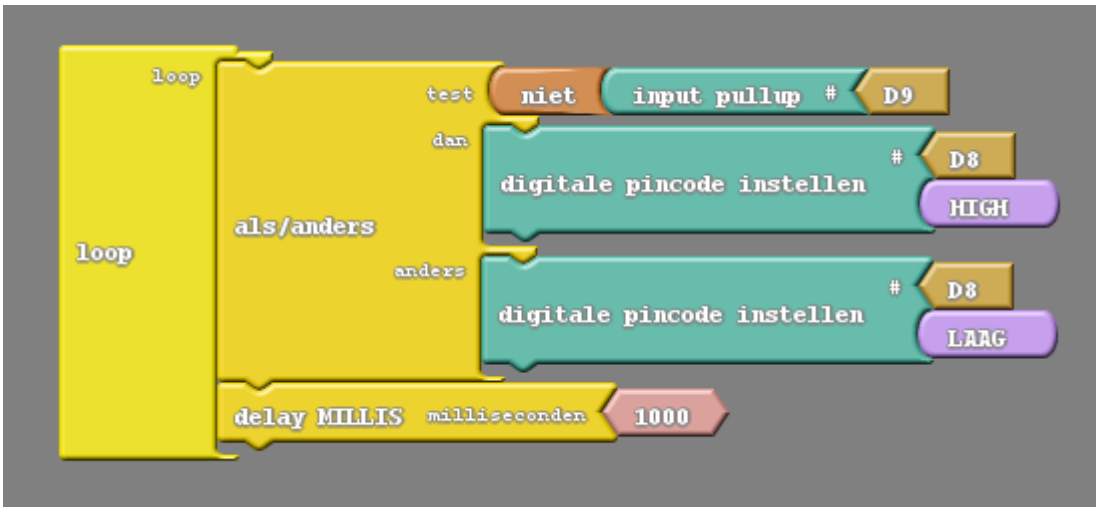


In Ardublock ziet de code er dan zo uit:



Oude ardublock versie

Merk op dat de bovenstaande code een NOT poort bevat die de laag actieve knop (als je er op drukt krijg je een “0”) terug omzet naar een “1”. Zo krijgen we toch nog positieve logica.



Nieuwe ardublock versie (21040826)

Hier is wel al een “input_pullup” functie voorzien en moeten we dit niet meer in de ardublock code zelf bijschrijven.

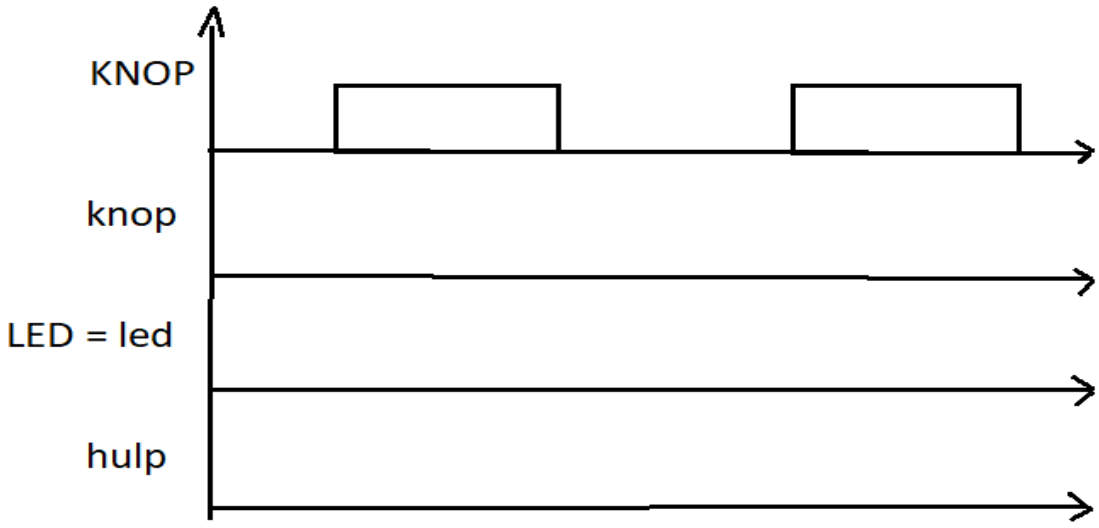
Extra: Je kan ook een knop checken door **flankdetectie** te doen. Wanneer je een laag actieve knop indrukt gaat deze van laag naar hoog (**opgaande flank**). Op het moment dat deze verandering gebeurt gaan we dit meten met de microcontroller. Eens we voorbij zijn aan deze verandering blijven we in dezelfde toestand staan. Dit kan een “0” of een “1” zijn. Zo krijgen we **een flipflop effect**, maar dan flankgestuurd.

Test volgende code uit en probeer te begrijpen hoe het werkt.

antidender

```
//flankgestuurde antidender
bool hulp = 0;
bool led = 0;
bool knop;
int KNOP=9;
int LED=8;

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(KNOP, INPUT);
}

void loop()
{
  if (digitalRead(KNOP)==1) {
    delay (100);
    knop=1;
    }
  else {
    knop=0;
    }
  if (knop == 1 && hulp == 0 && led == 0) {
       led=1;
       }
  else
   if (knop == 1 && hulp == 0 && led == 1) {
       led=0;
       }

  hulp=knop;//geheugenfunctie
  digitalWrite(LED, led);
}
```

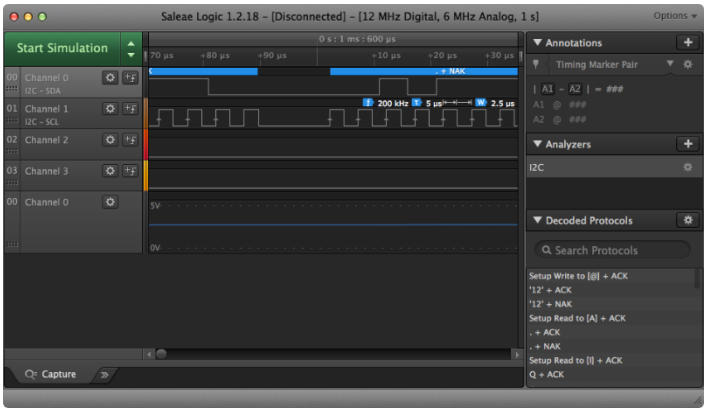
Vul onderstaande tijdsdiagram aan in functie van bovenstaande code!



7. Uitdagingen met schakelaars:

    1. Meet de antidender met jouw **logic analyser** en plaats de foto met uitleg in jouw verslag.
    2. Wanneer je op de schakelaar drukt gaat het **verkeerslicht** werken, anders niet.
    3. Wanneer je op een knop drukt (pin naar keuze) gaat een **dobbelsteen** van 7 LEDs lopen. Laat je de knop los dan stopt de dobbelsteen bij een waarde.
    4. Wanneer S1 (knop 1) **EN** S2 (knop 2) zijn ingedrukt gaat een rode LED branden. Wanneer S1 **OF** S2 gedrukt worden gaat een groene LED branden. Onderzoek ook wat je met de **NOT** functie kan doen.
    5. Maak een **tijdschakelaar**. Telkens je drukt gaat een LED voor 3 seconden aan. Daarna gaat deze automatisch uit.
    6. Maak een **flipflop** (bistabiele multivibrator). Met 1 knop ga je een LED aandoen maar daarna ook weer uitdoen. Gebruik hier de **NOT** operator (zie voorbeeld in PPP) of gebruik de **flankdetectie** van vorige pagina.
    7. Zorg voor 4 knoppen en maak een **codeslot**. Wanneer de code van 4 nummers in de juiste volgorde is doorlopen zal de LED aangaan. Tip: maak **eerst de flow van het codeslot in flowcode 8.** Daarna kan je veel makkelijker de code schrijven in Arduino.
    Je kan ook flowgarithm gebruiken. http://www.flowgorithm.org/
    Merk op dat je het codeslot met **IF THEN ELSE** kan maken, maar je kan evengoed met een **SWITCH of ARRAY** gaan werken (beide laatste komen nog later aanbod in de cursus).
    Meer inspiratie kan je ook vinden in het Arduino elektor boek van Bert Van Dam.
    8. Maak de dobbelsteen deze keer met een nummer generator om zo **random** getallen aan te maken. Meer info hierover vind je op de Arduino websites (zie o.a. randomSeed)
    9. Vervang de schakelaar in de dobbelsteen uitdaging 2 door een **tiltschakelaar.** Wanneer je beweegt met de dobbelsteen veranderen de dobbelsteenogen.

@ 4EE: leer nu met de logic analyzer omgaan



Saleae logic analyzer 24MHz 8 channels (maar dan de Chinese versie 😊

Software kan je downloaden via : https://www.saleae.com/downloads/

(in de gevorderden deel 1 cursus krijgt de cursist ook zo'n analyser en leren we er mee werken)

8. De Serial Monitor leren gebruiken

Wanneer we programmeren is er wel eens nood tijdens het **debuggen** om te weten te **komen wat er allemaal gebeurd in een microcontroller**. Om dit op te lossen kan je LEDs aansturen of metingen doen met een Logic analyzer op het juiste moment, maar je kan ook gebruik maken van de **seriële verbinding** tussen de Arduino en de PC COM poort. Voorwaarde is dus dat je de **USB kabel** verbinding blijft gebruiken. De tekst toon je op een **Serial Monitor.**

Doel: stuur tekst op de serial monitor op de PC wanneer je een knop hebt ingedrukt

Benodigdheden:

- Opstelling van het inlezen van de laag actieve knop (zie vorig hoofdstuk)
- Een USB verbinding met de PC

1. Het ARDUBLOCK programma ziet er dan zo uit:

Je leest de laag actieve knop in via pin 9. Wanneer deze ingedrukt is krijgen we een “0” aan de ingang. Deze nul wordt via de **NOT poort** omgezet naar een “1” of TRUE. Als de “if then” control een “1” ziet gaat hij de tekst “knop ingedrukt” versturen via de **seriële verbinding** (via de USB kabel) naar de PC. De tekst wordt hier getoond op de seriële monitor van Arduino. We voegen 100ms wachttijd toe om aan **antidender** te doen (zie hoofdstuk van de schakelaars).

2. Na de upload in c-code ziet de code er alsvolgt uit:

stuur_message

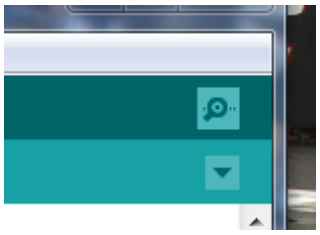
```
void setup()
{
  Serial.begin(9600);
  pinMode( 9 , INPUT_PULLUP);
}

void loop()
{
  if (!( digitalRead( 9) ))//laag actief
  {
    delay( 100 );
    Serial.print( "knop ingedrukt" );
    Serial.println("");
  }
}
```

Je kan tot 2000000 gaan (2Mb).

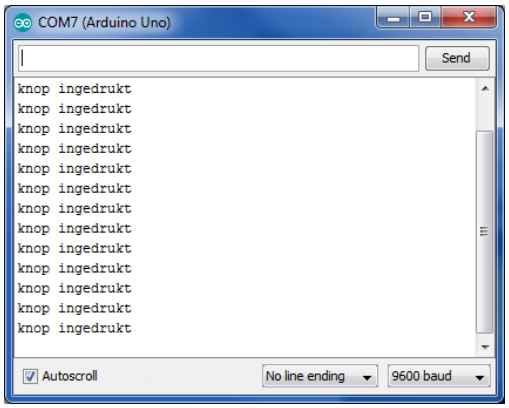
Zorg wel dat beide kanten dezelfde baudrate ingesteld hebben.

Merk op dat we in de c-code het woord **“_PULLUP”** hebben toegevoegd. Dit omdat we in hardware geen externe pull-up weerstand gebruiken maar deze intern aanzetten (zie vorig hoofdstuk).

Na het uploaden van de code moet je het volgende icoontje indrukken in Arduino om de serial monitor te starten:



Nu toont deze monitor de tekst “knop ingedrukt” telkens wanneer er op de knop is gedrukt. Er wordt automatisch naar de volgende lijn gesprongen.



Zorg dat de datasnelheid van de monitor op 9600 baud staat (default waarde).

Meer info over wat je in de c-code terug vindt, kan je in de volgende extra info terugvinden.

3. Meer info en programmeertips over het sturen van messages via de serial monitor in c-code:

De volgende tabel geeft aan wat de verschillende seriële functies zijn in Arduino:

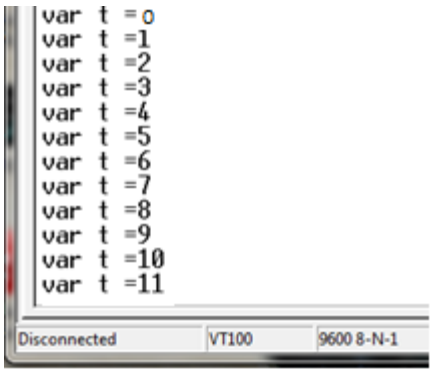
Opdracht	Uitleg
Serial.begin(baudrate)	Stel de snelheid van de communicatie op "baudrate" in.
Serial.print(variabele)	Druk de inhoud van de variabele af naar de seriële poort (in ASCII).
Serial.println(variabele)	Idem, maar spring daarna naar het begin van de volgende regel.
Serial.write(variabele)	Druk de inhoud van de variabele af naar de seriële poort (als byte). Vb een karakter afprinten
variabele = Serial.read()	Lees een teken (een enkele letter of cijfer) van de seriële poort en stop het in de variabele. Wanneer er niets staat om te lezen wacht deze opdracht niet, maar geeft de variabele waarde -1.
variabele = Serial.available()	Bekijk of er een gegeven ontvangen is op de seriële poort (variabele = true) of niet (variabele = false).
void serialEvent()	Speciale functie die aangeroepen wordt als er iets op de seriële verbinding gebeurd (een interrupt).

Opmerking: **Serial.end();** zorgt ervoor dat de monitor stopt met printen !

Bekijk het volgende voorbeeld programma eens:

serial_monitor_shield

```
int t= 0;

void setup() {

  Serial.begin(9600);//baudrate instellen

  }

void loop() {

  Serial.print("var t =");
  //print inhoud t en spring naar volgende lijn
  Serial.println(t);

  t++;
  delay(1000);
  }
```



Resultaat van de linker code op de monitor

De variabele "t" is een **geheugenplek** dewelke we voorzien in de microcontroller en die een grote heeft waar een **integer** (2 bytes = tekenbit + 15 bits) in kan. Als bit 15 (MSb) "1" is zal het een negatief getal zijn terwijl als het een "0" is geeft dit aan dat het een positief getal is.

Zie volgende tabel i.v.m. de soorten variabelen.

variabele	minimale waarde	maximale waarde
byte	0	255
int	-32.768	32.767
word [12]	0	65.535
float [13]	-3,4028235 1038	3,4028235 1038

Int = 2 bytes ((- 2 ^15) …. ((2 ^15) – 1)

Word = 2 bytes (2 ^16)

Float = 4 bytes (- 3.4028235 +38 …. 3.4028235 + 38) (32 bits = 4 bytes)

Merk op dat er ook een variabele type “string” bestaat dat dient om karakters (vb “vlag”) op te slaan en te bewerken.

Merk op dat je **globale variabelen** aanmaakt voor de setup functie. Deze kunnen dan gebruikt worden in het ganse programma, ook in andere subroutines (zie verder).

We schrijven daarom **int t;** omdat we een getal verwachten dat groter dan 255 zal zijn.

We moeten daarna de juiste **snelheid** baudrate instellen van de seriële poort. Deze moet gelijk zijn aan die van de serial monitor. We kiezen voor **9600** baud. Het commando **Serial.begin(9600)** zorgt dat de poort wordt geïnitialiseerd en de snelheid correct ingesteld wordt.

Je moet dit commando in de setup functie zetten omdat dit maar 1 keer moet uitgevoerd worden.

Serial.print(“var t = ”) gaat de tekst tussen de “ “ afprinten.

Serial.println(t) zal de inhoud van de **variabele t** afdrukken en **springen naar het begin** van een **volgende regel** (ln = linefeed). Wens je dit niet te doen dan kan je ook gewoon Serial.print(t) schrijven.

Door **t++** te doen wordt de variabele **incremented** (met 1 verhoogd) en terug in dezelfde variabele gestopt. Het is dus hetzelfde als t = t + 1 maar dan korter geschreven.

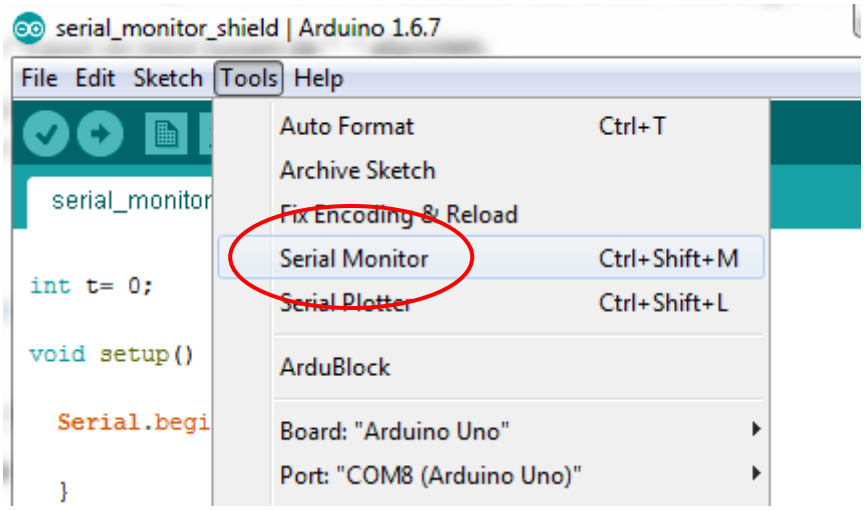
Er bestaan nog andere manieren om snelle code (afkortingen) te schrijven, zie onderstaande tabel:

Afkorting	Betekenis	Uitleg
i++	i = i + 1	Verhoog i met 1.[16]
i--	i = i - 1	Verlaag i met 1.
i+=5	i = i + 5	Tel 5 op bij i.
i-=5	i = i – 5	Trek 5 af van i.
i*=5	i = i * 5	Vermenigvuldig i met 5.
i/=5	i = i / 5	Deel i door 5.

4. Hoe stel je de Serial Monitor in?

In Arduino kan je gebruik maken van een ingebouwde Serial Monitor.

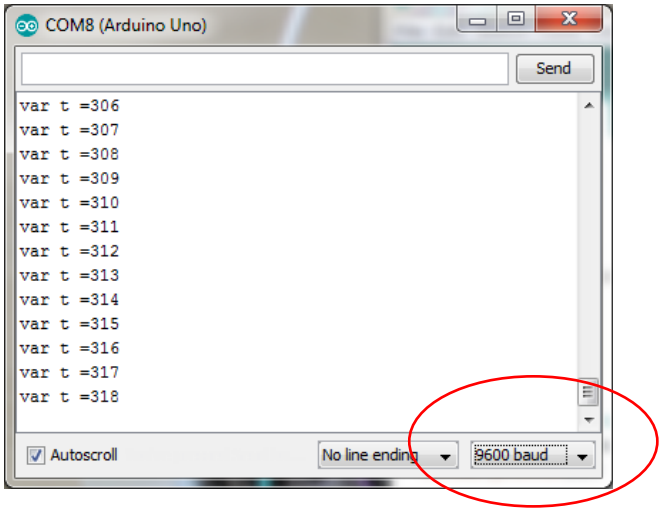
Klik maar eens in de menu **Tools -> Serial Monitor** (kan ook via de serial monitor icoon in de menu).



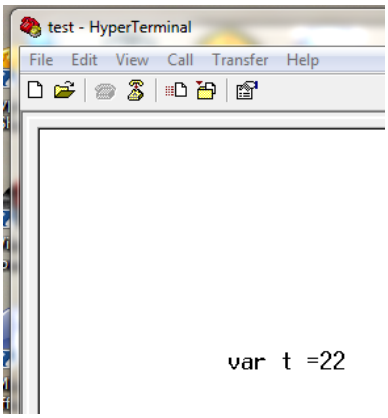
Als de Arduino juist is ingesteld is om code te uploaden zal ook de **COM poort juist** ingesteld staan van de Serial Monitor. In dit voorbeeld is dit COM8.

De PC denkt dat er een serieel apparaat hangt aan de PC maar in feite hangt er een USB apparaat aan dat aan Windows doorgeeft dat het serieel werkt. We noemen dit ook een **virtual COM poort**.

Zorg dat de baudrate van de monitor op **9600 baud** ingesteld staat.



Soms wil je tekst op en **bepaalde plaats** tonen op het PC scherm in hyperterminal. Dan kan dit het volgende resultaat zijn:



Hoe kunnen we nu de **positie verplaatsen** van onze cursor?

We maken gebruik van **VT100 commando's** (ook wel ESC commando's genoemd). Dit omdat we steeds eerst een ESC moeten sturen (ASCII waarde = 27, zie volgende tabel) en daarna pas het VT100 commando (zie volgende tabel).

ASCII	omschrijving of teken	afdrukbaar
7	Terminal bell	nee
8	Backspace	nee
9	TAB	nee
10	Linefeed (nieuwe regel)	nee
13	Carriage return (naar begin regel)	nee
27	Escape	nee
32	Spatie	ja
48	0	ja
65	A	ja
97	A	ja

Stukje uit de ASCII tabel

Wanneer we een ESC willen sturen moeten we dus de ASCII waarde 27 versturen.

Er bestaan allerlei commando's om het scherm aan te sturen van jouw serial monitor.

VT100 opdracht	uitleg
[2J	Veeg het ganse scherm uit
[1K	Wis alles weg van in't begin van het scherm tot aan de cursor
[0K	Wis de regel uit van de cursor tot aan het einde van de regel
[H	Zet de cursor in de linker bovenhoek
[x;yH	Verplaats de cursor naar positie (x,y)
[7m	Druk de tekst wit op zwart af
[?25l	Maak de cursor onzichtbaar
[?25h	Maak de cursor weer zichtbaar

Enkele handige VT100 commando's

Als we deze kennis nu eens toepassen op de volgende code:

serial_monitor

```
int t= 0;

void setup() {

  Serial.begin(9600);//baudrate instellen

  Serial.write(27);
  Serial.print("[?25l");//cursor onzichtbaar

  Serial.write(27);
  Serial.write("[2J");//wis ganse scherm

  }

void loop() {

  Serial.write(27);
  Serial.print("[10;15H");//locatie x=10, y=15

  Serial.print("var t =");
  Serial.print(t);

  t++;
  delay(1000);
  }
```

Elke **Serial.write(27)** stuurt een **ESC** in de vorm van een ASCII byte (27) naar de seriële monitor. Daarna volgt dan een VT100 commando.

Serial.print("[?25l") zorgt ervoor dat de cursor niet zichtbaar is op de monitor.

Merk op dat je de VT100 commando's als tekst moet doorsturen. Hiervoor moeten de karakters tussen " " geschreven worden. Het VT100 commando is tekst en kan **ook** gestuurd worden via een **Serial.print** commando. Omdat er een **ESC teken voorkomt zal het toch niet afgedrukt** worden. Je mag dus kiezen tussen Serial.write en Serial.print.

Serial.write("[2J") zorgt ervoor dat het **scherm wordt gewist.**

Serial.write is trouwens een commando dat bedoeld is om gegevens door te geven die enkel een computer kan lezen. Bij **Serial.print** is het de bedoeling dat de **mens de gegevens kan lezen**.

Door **Serial.print("[10;15H")** te versturen geven we aan dat de **cursor** moet gaan staan op de plaats x = 10 (horizontaal) en Y = 15 (verticaal).

Als je **Serial.print("var t=")** zonder voorgaand ESC commando zal deze **tekst afgeprint** worden op het scherm.

Tot slot wordt de **inhoud** van de variabele afgedrukt via **Serial.print(t).**

5. Uitdagingen voor de serial monitor:

    1. Maak in de volgende programma's gebruik van deze serial monitor functie om jouw code te debuggen!
    2. Maak een wiskundige berekening in jouw code en toon zowel de ingegeven waardes als het resultaat op de serial monitor en dit onder elkaar.
    3. Zorg dat er 5 variabelen in een stuk code worden teruggestuurd, maar we gaan pas verder naar de volgende variabele wanneer er **ENTER (of een ander karakter) gedrukt** is van het keyboard. Zie hiervoor de functie **Serial.available()** en **Serial.read().**
    4. Gebruik opnieuw de functies uit oef 3 maar stuur nu een **LED aan/uit telkens je op ENTER** drukt.
    5. Print op het scherm een ASCII tabel uit. Je kan hier **Serial.print(x , DEC)** voor gebruiken. Zie op www.ascii-table.com voor meer info over de ASCII tabel.
    6. Toon op het scherm of de knop S1 / S2 zijn ingedrukt of niet met de tekst "schakelaar gesloten" of "schakelaar open".
    7. Meet het **RS232 signaal met jouw logic analyzer** terwijl je een ASCII getal verstuurd. Kan jij het getal terugvinden op de analyzer?
    8. Onderzoek de werking en gebruik van de **Serial Plotter** in Arduino (zie info verder in cursus). Maak er bijvoorbeeld een **sinusgenerator** mee. Zie zeker eens op https://rheingoldheavy.com/new-arduino-serial-plotter/

Opmerking: soms wordt de ENTER ook teruggestuurd en gezien door de serial monitor als een karakter (afhankelijk van de serial monitor versie in Arduino). Als het hindert kan je bijvoorbeeld een alternatief programma zoals **Putty** gebruiken.
https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

In uitdaging 3 kan je **een karakter** proberen in te lezen tot je op ENTER drukt. Enter is geen karakter maar wel een **combinatie van een LINE FEED** (volgende lijn) **en CARRIAGE RETURN** (ga naar begin van de lijn). De ASCII waarde voor LF = 10 en CR = 13 (zie tabel).

Om deze oefening goed te kunnen begrijpen moet je eerst de gestuurde karakters eens printen op het scherm. Dit kan bijvoorbeeld met de volgende oefening. De code kan je niet in Tinkercad testen, wel op een aangesloten UNO !!! Tinkercad stuurt de ENTER niet door !

serial_mon_enter_oef

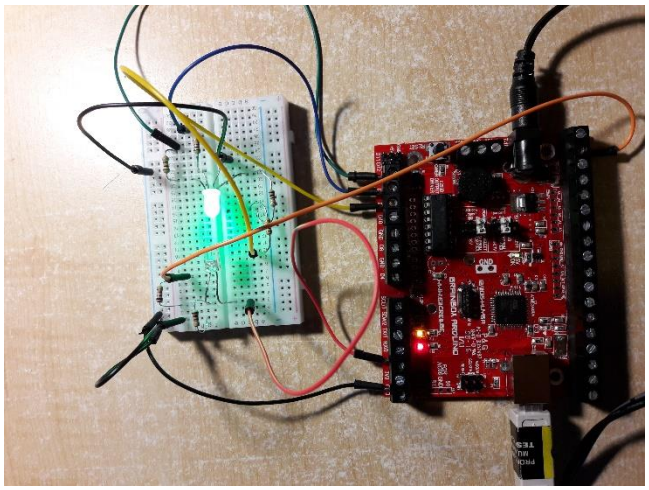
```
byte incomingByte = 0; // for incoming serial data

void setup() {
  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}

void loop() {
  // reply only when you receive data:
  if (Serial.available() > 0) {   //lees tekst en ENTER, karakter / karakter
    // read the incoming byte:
    incomingByte = Serial.read();

    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte,DEC); //print karakter als decimaal getal
                                      //check de ASCII code tabel of dit klopt

    //niet in tinkercad te testen !!!! wel op uno
    if(incomingByte == 10){  //toets enter -> LF = line feed = 10 (ascii code)

     Serial.println("enter gedrukt");
    }
  }
}
```

Hoe kan je een ganse tekst versturen?

serial_mon_oef_string §

```
String input = "";

void setup() {
  Serial.begin(9600);
}

void handleSerialInput() {  //functie die input uitprint
    Serial.print("Ontvangen tekst: '");
    Serial.print(input);
    Serial.println("'");
    input = ""; //Wis het ontvangen bericht
}

void loop() {
  while (Serial.available() > 0) {
    char c = Serial.read();
    if (c == '\n') {  // bij \n ontvangen = new line, enter gedrukt

      handleSerialInput(); //spring naar functie voor afprinten geheel

    } else if (c != '\r') {  //zolang niet \r carriage return gedrukt
                             //ga voor volgende char op te halen
                             //en negeer dit karakter (niet toegevoegd in rij)
      input += c; //verzamel de karakters van de string achter elkaar
    }
  }
}
```

8.2 Toon de resultaten in een excel sheet op de PC



Als voorbeeld hebben we een **spectrometer** gebruikt waarbij metingen van kleurintensiteit kunnen verstuurd worden naar de PC. De laatste stap bij deze opstelling is namelijk de **data aanschouwelijk maken** zodat we deze **goed kunnen analyseren.**

Tip: Om deze oefening handig zelf te testen kan je best een **reeksen met variabelen** versturen vanuit de Arduino naar de PC.

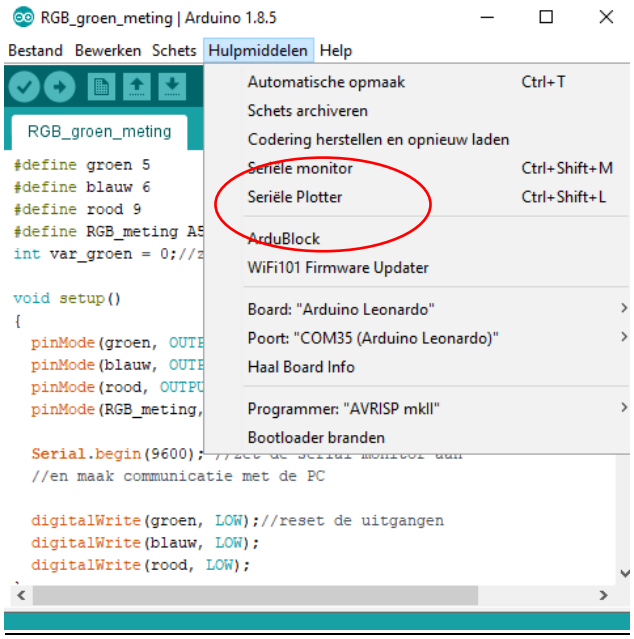
Een mogelijkheid om data makkelijker te gaan analyseren is deze **visueel** maken. Zo kunnen we heel eenvoudig **in Arduino IDE** de data voorstellen via de "**serial plotter"** in een grafiek. De curve wordt **constant aangepast** aan de laatste metingen tijdens het meten.

Nadeel van deze methode is dat we de data **niet kunnen saven**. Daar is dan de 2de meetmethode voor uitgezocht. Eerst de **seriële data opslaan in een bestand** en daarna via **Excel uitlezen** en op het scherm tonen in een grafiek.

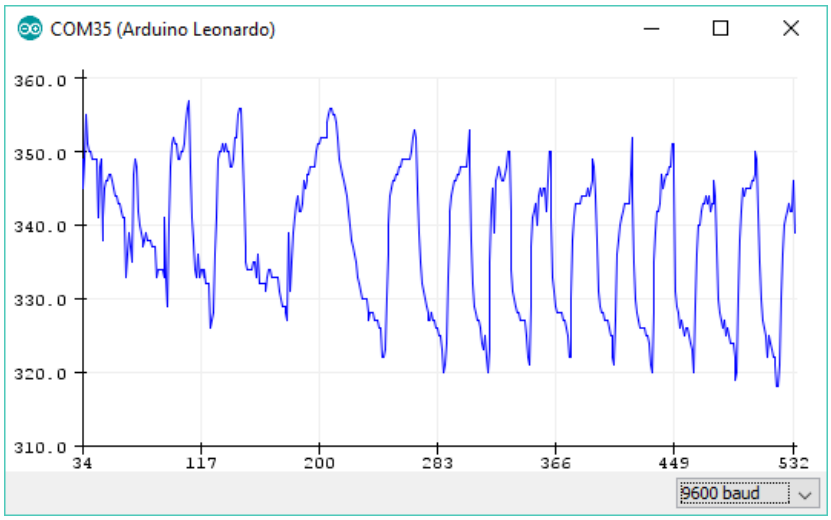
Test zeker beide methoden uit op jouw opstelling.

Methode 1: de serial plotter

1. Download de Arduino code om bijvoorbeeld enkel de lichtintensiteit van een groene LED te meten. Zie dat de Arduino schakeling op de brainbox of UNO uiteraard hiermee overeenkomt (zie vorige hoofdstukken).
2. Open nu via hulpmiddelen -> **seriële plotter**



3. Nu toont Arduino de gemeten waardes op het scherm.



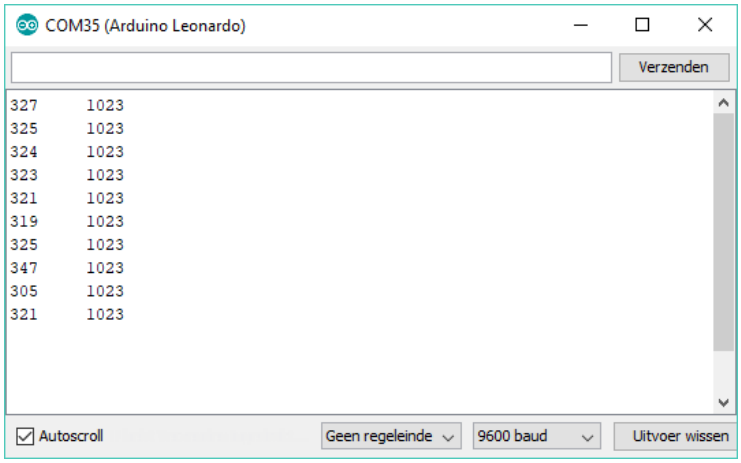
De grafiek toont nu de lichtintensiteit (tussen 0 en de maximum waarde) en de metingen in de tijd. In mijn programma werd er om de minuut gemeten.
Merk op dat de plotter automatisch de schaal aanpast.

Methode 2: file saven en dan tonen in een excel sheet.

1. Zorg er weer voor dat je een programma hebt dat via de serial monitor **enkel** **"getallen"** terugstuurt, vb van de groene meting. Als je tekst mee zou sturen moeten we straks in excel deze er eerst uitfilteren. Tekst kan je namelijk niet in een grafiek tonen 😊
2. Merk op dat je in de Arduino code **2 extra lijnen** voorziet:

```
Serial.print(var_groen);

Serial.print("\t");          // prints a tab - nodig voor uitlijning
Serial.println(1023);        // print waarde 1023 om Seriele plotter in dezelfde schaal te houden
```

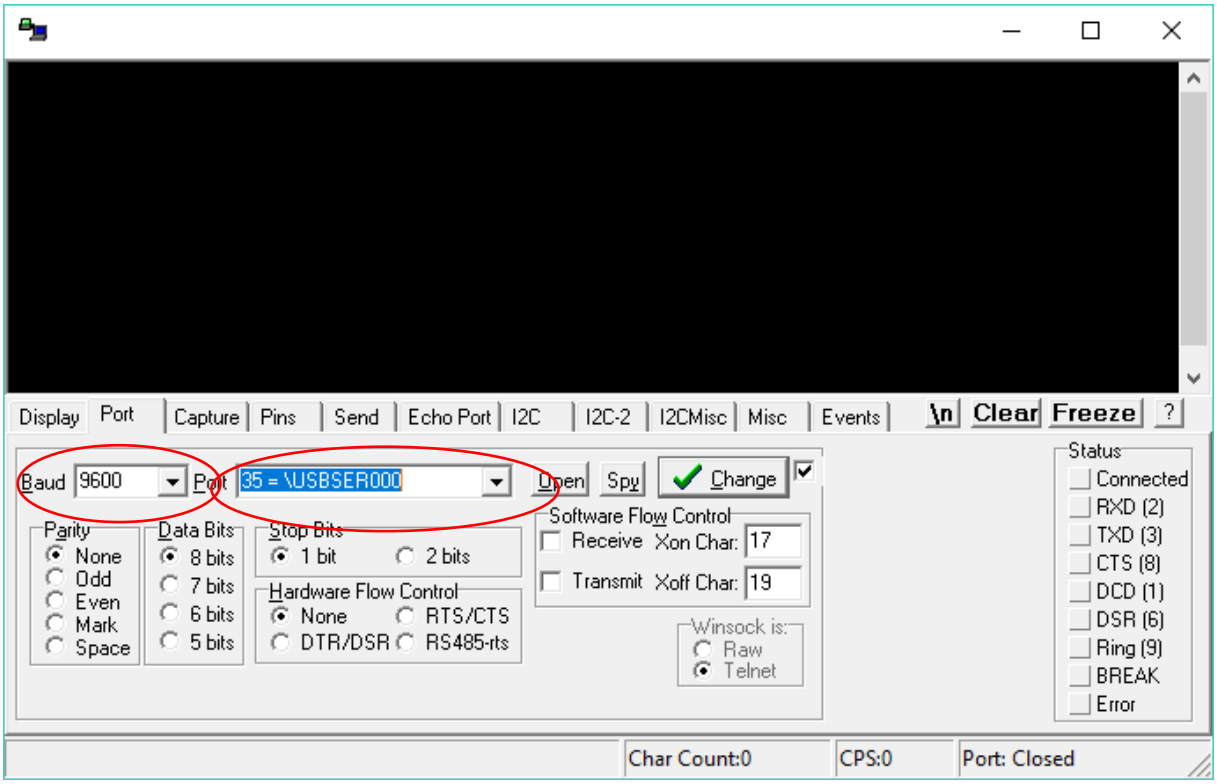
3. Zorg dat de Arduino de getallen verzendt en ze af te lezen zijn op de serial monitor.



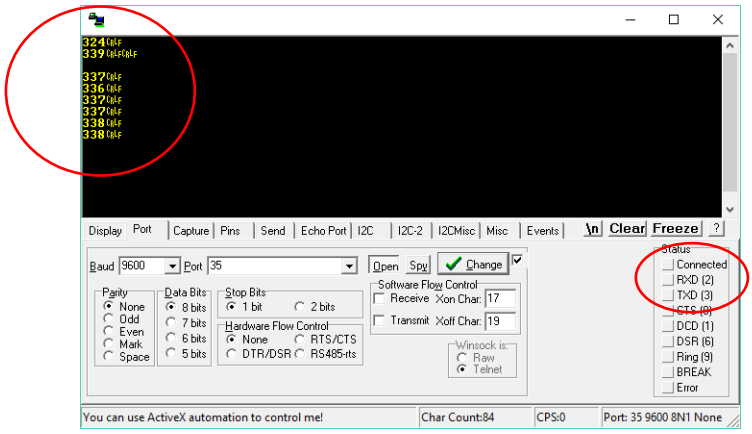
4. **Sluit** nu de seriële monitor. Je mag op dit moment niet meer gebruik maken van de COM poort via deze monitor.
5. Installeer het gratis programma "**Realterm**" op jouw PC en start dit op.
   De software kan je hier downloaden: https://sourceforge.net/projects/realterm/

6. Klik op de tab "**port**" en stel de Baudrate in (bytes per seconds).

7. Stel ook de poort juist in op de **COM poort** van jouw Arduino, hier poort 35.

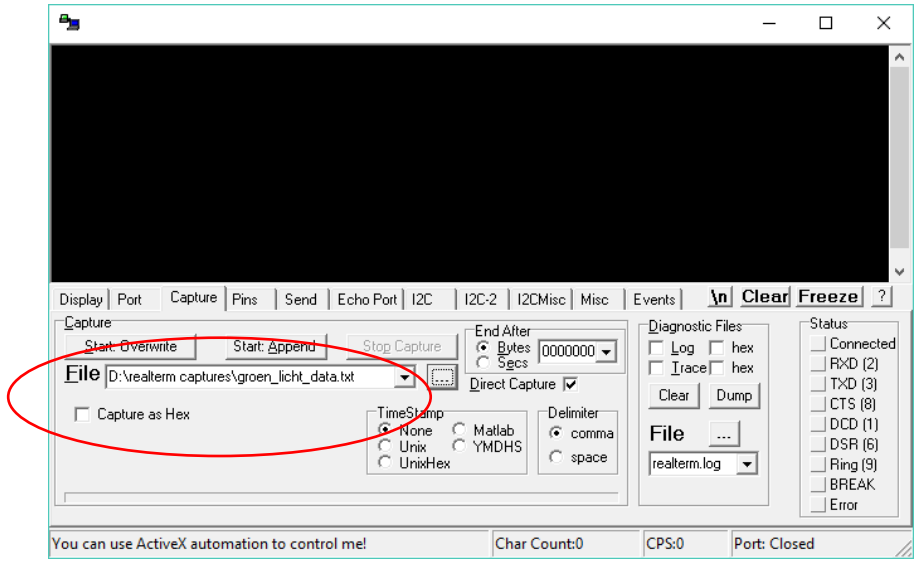


8. Klik nu op "**open**" om de communicatie met de Arduino COM signalen te starten en binnen te laten. Nu moet de **RXD pin** af en toe oplichten in het **geel, telkens er een nieuwe byte wordt verstuurd**.

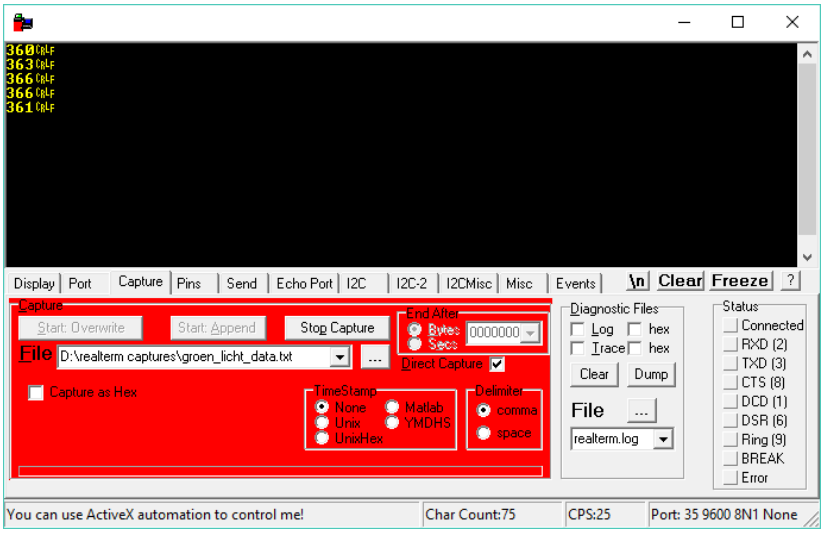


Je kan de data aflezen op het scherm. CRLF staat voor "carriage return" en "line feet". Dit wil zeggen dat de Arduino een "ln" uitvoert. Het effect is hetzelfde als op "**enter**" drukken = spring naar de volgende lijn en ga naar het begin van die lijn.

9. Zorg dat in de tab “**Capture**” de **locatie en naam van de file,** waar je jouw data wil in saven, juist staat. Voeg zelf de extensie *.**txt** toe !



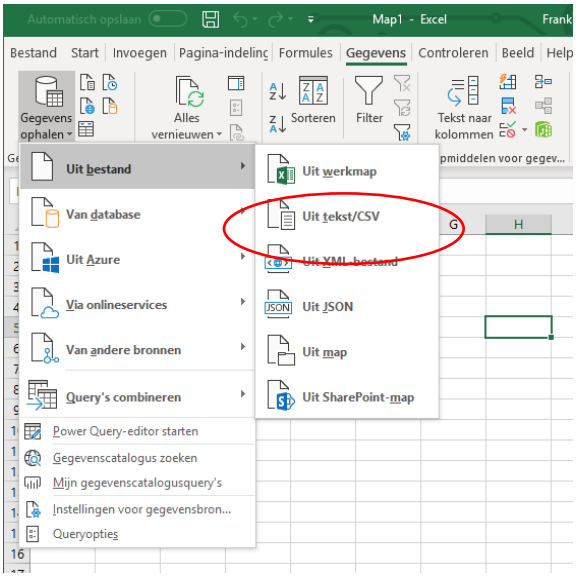
10. Je mag deze meting **constant laten lopen**, zo wordt achteraf jouw grafiek in excel ook automatisch geupdated.
11. Druk nu op de knop “**Start:overwrite**” om de data op te laten slaan in de file!



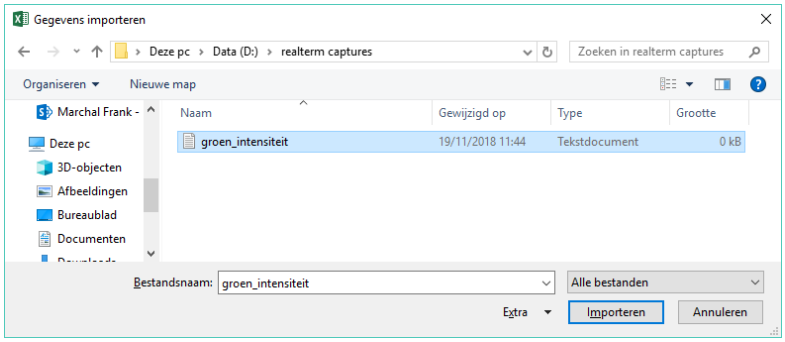
Nu wordt het toetsenbord “rood”, dus we zijn aan het opnemen.

12. Maak Realterm klein en **open nu Excel,** terwijl je dus Realterm laat open staan en de Arduino nog steeds data stuur.

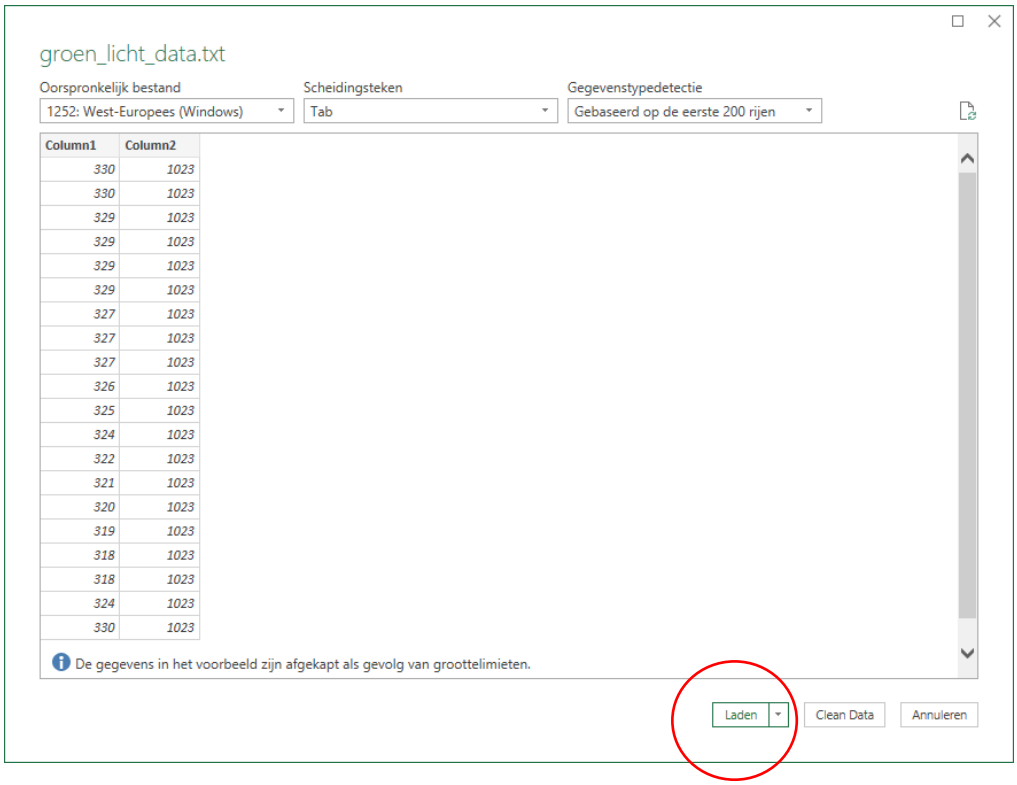
13. Start met een **lege sheet** en klik op gegevens -> gegevens ophalen -> uit bestand -> **uit tekst**



14. Selecteer de **plek waar jij jouw txt file hebt gesaved** waar de data van Realterm wordt in weggeschreven.

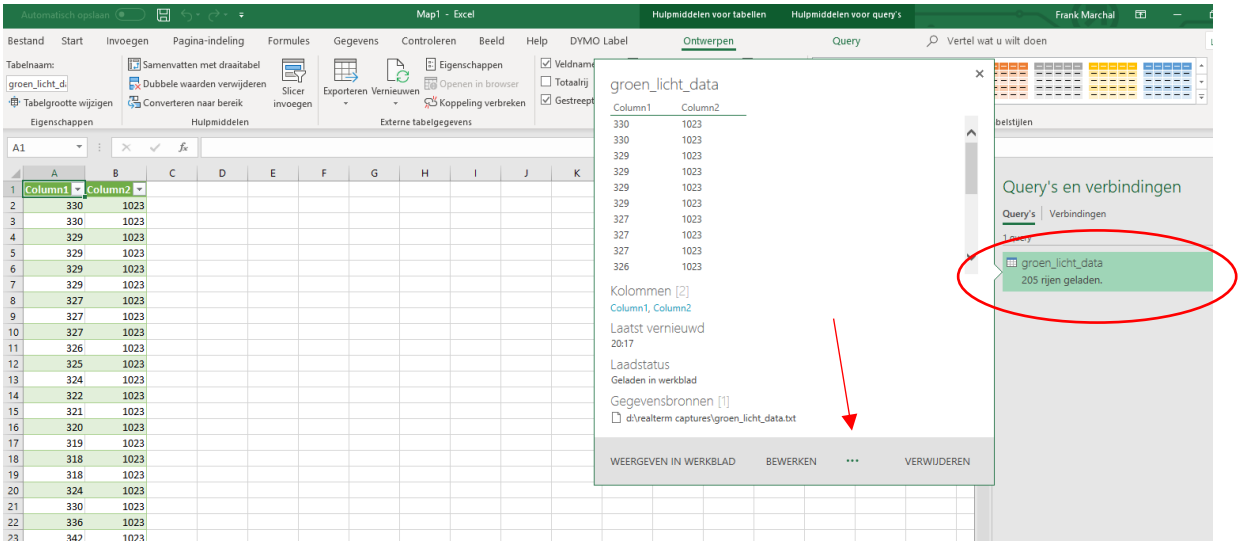


15. Nu kan je "**importeren**" en krijg je de inhoud van de file te zien:



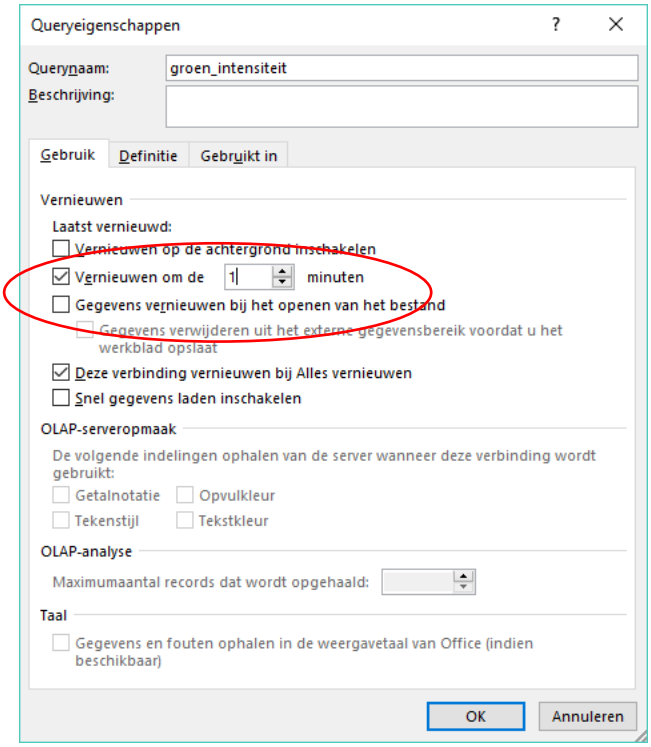
16. Klik op **“laden”**. Nu wordt de data ingeladen in Excel.

17. We moeten nu nog enkel instellen dat de data regelmatig wordt vernieuwd. Dit doen we door op de groen gemarkeerde file naam te gaan staan, en dan op de **3 puntjes** te klikken die er in het pop-up venster verschijnen.



18. Selecteer “**eigenschappen**”.

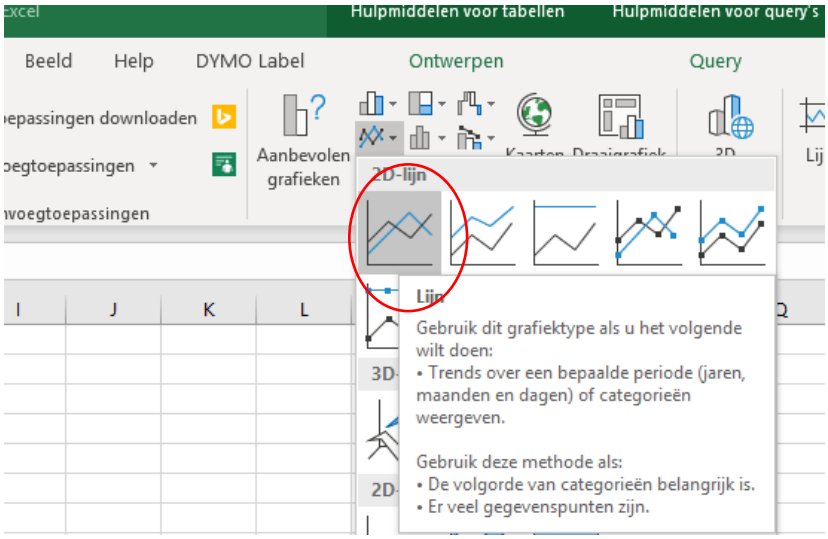
19. Vink “**vernieuwen om de …. minuten**” aan en stel dit in op bijvoorbeeld minimum 1 minuut.



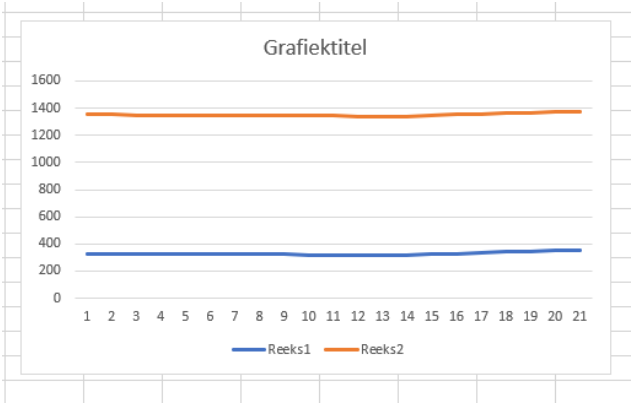
20. Klik op **OK** om het venster te sluiten.
21. **Sluit** het zijvenster door het kruisje te sluiten.
22. **Save het bestand** onder een naam.
23. Nu moeten we nog een grafiek uit de tabel halen. **Selecteer hiervoor een aantal items** uit de kolom die je wil in de grafiek zetten, samen met de 1023 kolom.

329	1023
329	1023
329	1023
329	1023
327	1023
327	1023
327	1023
326	1023
325	1023
324	1023
322	1023
321	1023
320	1023
319	1023
318	1023
318	1023
324	1023
330	1023
336	1023
342	1023
346	1023
350	1023

24. Klik dan op invoegen -> 2D lijn -> lijn

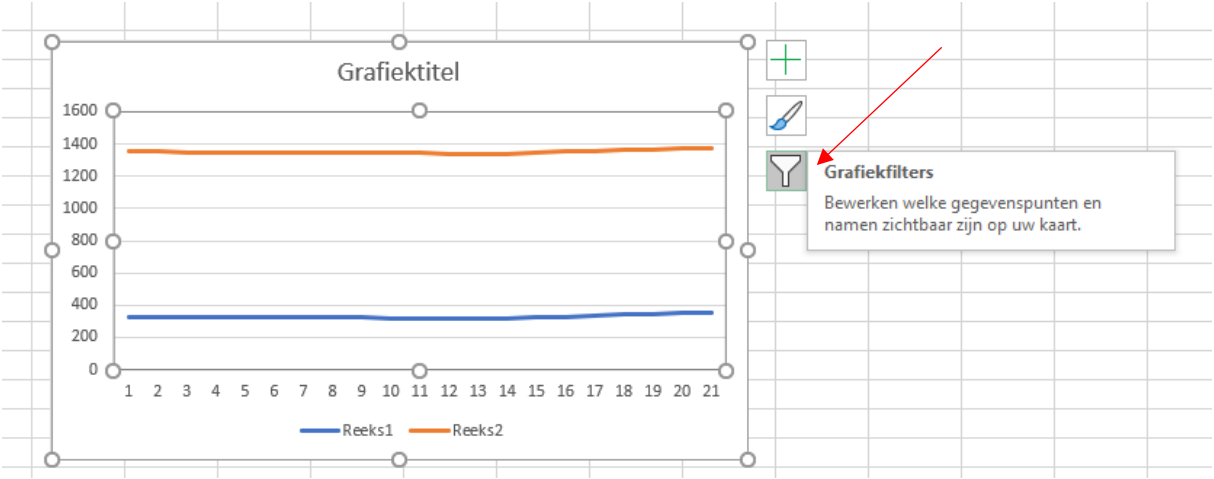


25. Nu verschijnen de volgende 2 grafieken.

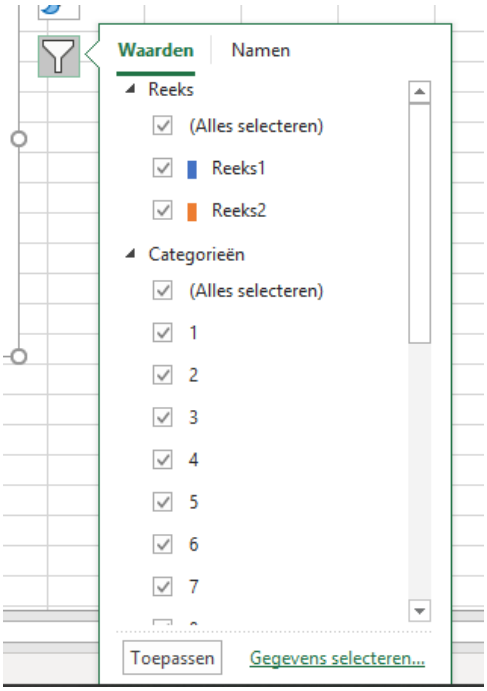


De **reeks 1 zijn de getallen die we wensen** in een grafiek te zetten. Reeks 2 is de lijn van 1023, die ons in feite niet boeit. We **moeten deze echter even meesturen** om de grafiek te kunnen verkrijgen in Excel.

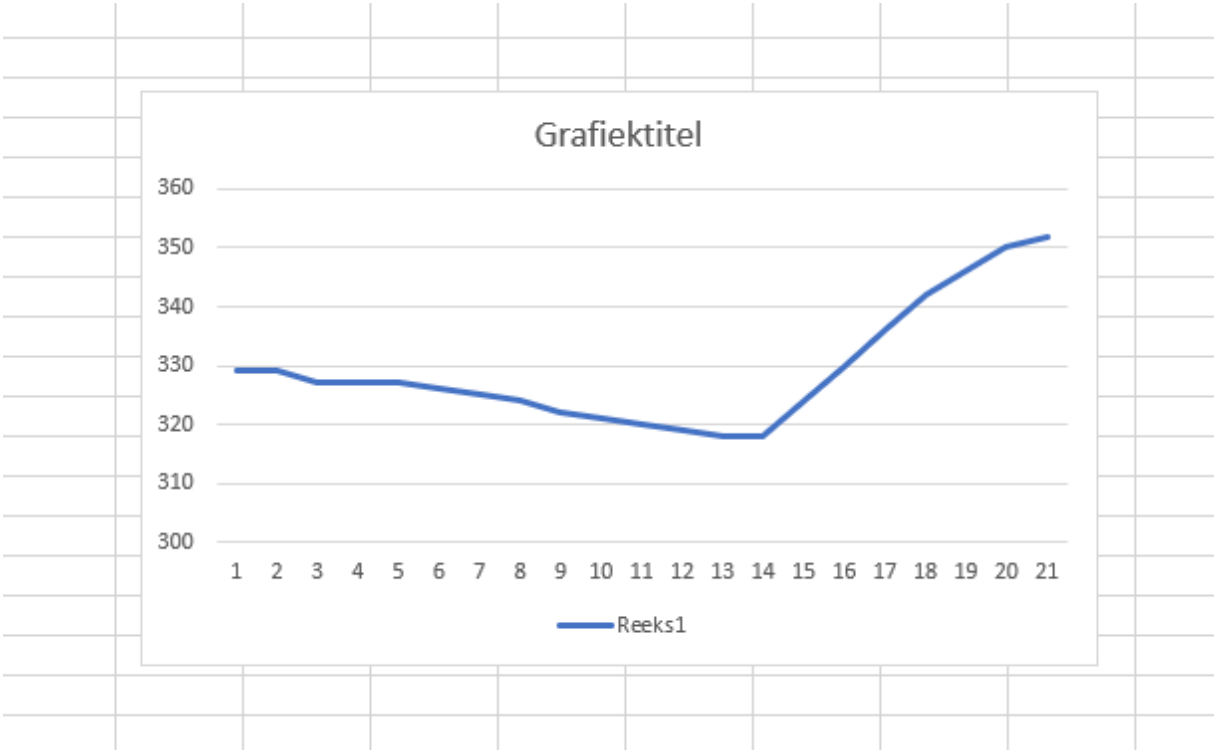
We **schakelen reeks 2 gewoon uit** door rechts op de grafiek te klikken en de filter te selecteren.



Vink nu “reeks2” uit en klik op **“toepassen”.**



Uiteindelijk krijgen we nu de **gewenste grafiek** op het scherm:



Elke minuut zal nu de laatste data op het scherm geplaatst worden en kan de verandering gevolgd worden. Uiteraard zullen algen veel trager groeien en kan je best de **refresh tijd verhogen tot 60 minuten of meer.**

9. Geluid maken met een buzzer

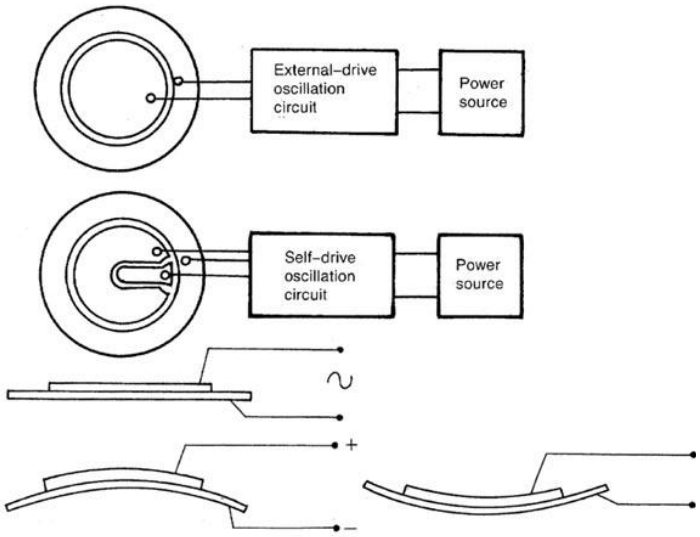
Geluid maken met de Arduino kan door extern een **buzzer** aan te sluiten.

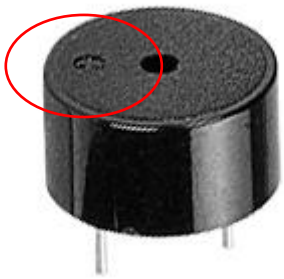
Met geluid kan je bijvoorbeeld aangeven wanneer er een error in de code zit of wanneer je op een knopje drukt. Of je kan er echte muziekstukken en alarm sirenes mee bouwen.

Hoe passen we de hardware voor geluid aan?

Er zijn 2 soorten buzzers:

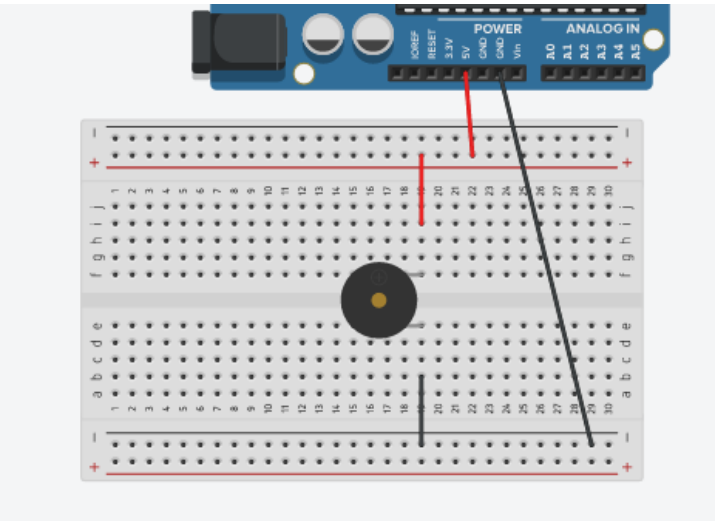
1. Een buzzer met een **piezo element <u>zonder</u> eigen oscillatie circuit**. In dit geval moeten we zelf een wisselende spanning of blokspanning aanleggen. Hiermee kunnen we alle desgewenste tonen opwekken. Deze buzzer heeft **geen polariteit.**
2. Er bestaan ook buzzers met piezo element **<u>met</u> een eigen oscillatie circuit**. Hier moeten we enkel een "1" aanleggen wanneer we geluid willen horen. Bij een "0" stopt het geluid. Andere tonen kunnen niet opgewekt worden dan de voorziene. Let op: hier is een duidelijke **polariteitsaanduiding** voorzien, dus de "+" aan de microcontroller pin en de "-" aan de massa.





Buzzer met "+" aanduiding !

Doe een test met de actieve buzzer (met eigen oscillatie circuit):

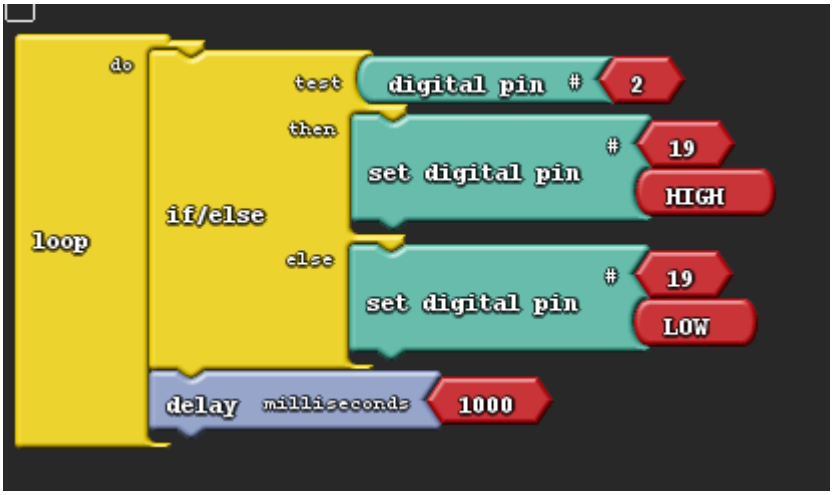


Deze buzzer heeft polariteiten(zie +)

Hij geeft 1 constante frequentie.

➔ Je kan ook code schrijven voor deze buzzer:

Als er op de knop gedrukt wordt moet de buzzer werken. Sluit deze aan op bijvoorbeeld pin A5 (pin 19).



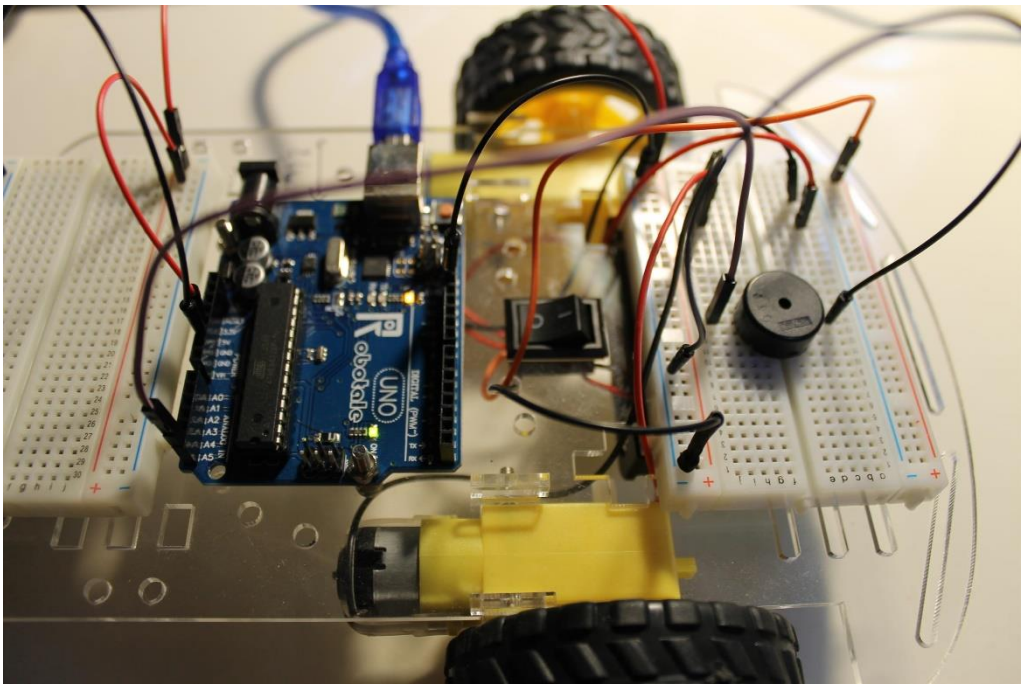
Test deze code eens uit.

Doel: maak geluid met een passieve buzzer (zonder eigen oscillatiecircuit)

Benodigdheden:

- Robot chassis met Arduino Uno
- Een passieve buzzer
- Enkele draden

1. Finale opstelling:

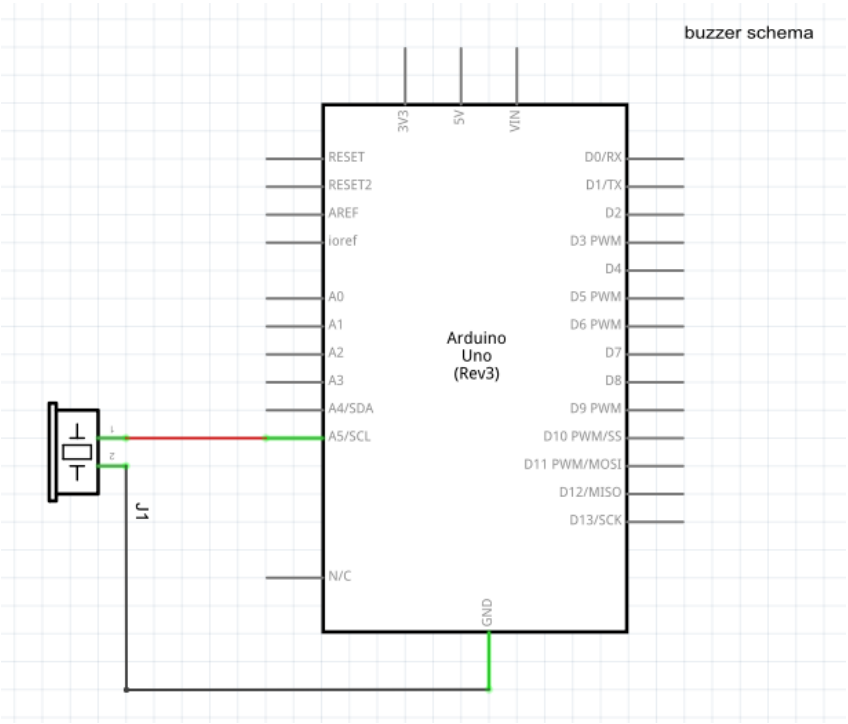


We sluiten de buzzer aan op analoge pin 5 (A5) van de Arduino. We gebruiken echter deze pin enkel **digitaal,** want we gaan met een **blokgolf** van “1-en” en “0-en” geluid produceren. Daarom kunnen we deze pin ook nummer 19 noemen.

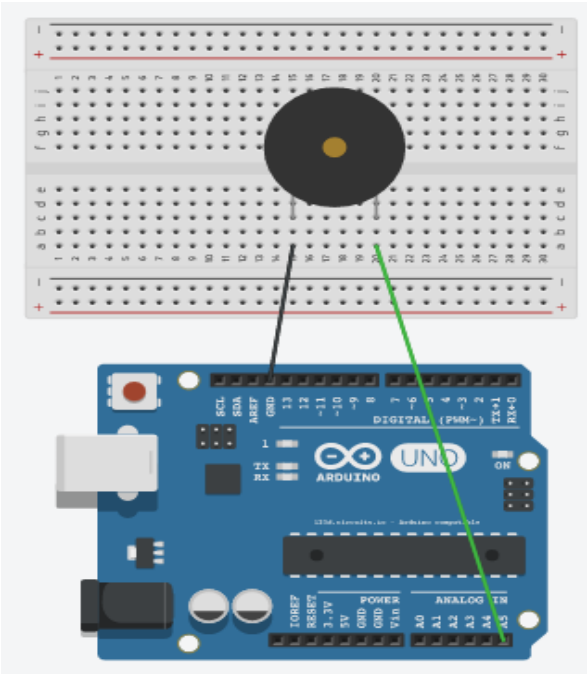
Je vind deze nummer 19 door gewoon na de digitale pin 13 aan de rechter zijde van de Arduino verder te tellen met de analoge pinnen aan de linker zijde. Sommige pinnen hebben nu eenmaal meerdere functies bij deze processor (zie technische info van de Arduino).

De **passieve buzzer heeft geen polariteit.** Dus je kan deze gewoon aansluiten met 1 willekeurige zijde aan de massa en de andere aan digitale pin 19.

2. Het schema van de buzzer:

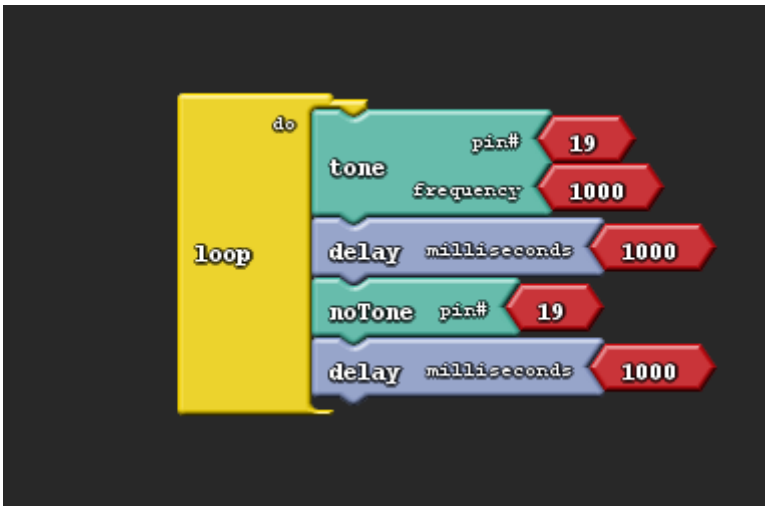


3. De breadboard opstelling:



Bouw dit schema op jouw robotchassis.

4. De volgende ARDUBLOCK code geeft 1 seconde een toon van 1KHz afgewisseld met een pauze van 1 seconde.



Ga in de “pins” menu op zoek naar de **tone** en **noTone** functie. Sluit pin 19 aan op beide functies. Stel de frequentie in op 1000Hz. Voorzie een pauze van 1 seconde na de toon. Hierbij geef je aan hoelang de toon zal duren. Daarna is er 1 seconde stilte (**noTone**) voorzien.

Merk op dat je van de tone functie niet kan gebruik maken op bepaalde pinnen, wanneer je ook PWM voor de motoren gaat toevoegen (zie meer info hierover bij de uitleg van de DC motoren).

5. Meer info over de c-code van deze tone functie:

geluid

```
#define SPEAKER 19
void setup()
{
    pinMode(SPEAKER,OUTPUT);
}

void loop()
{
  tone(SPEAKER, 1000);
  delay( 1000 );
  noTone(SPEAKER);
  delay( 1000 );
}
```

In c-code kan je perfect dezelfde functies terug vinden

De functies om geluid met Arduino te maken zijn dus eenvoudig.

Tone(pin,frequentie) gebruik je op de pin waar het geluid moet uitkomen aan te geven en welke frequentie deze toon moet hebben. De frequentie is uitgedrukt in **Hertz.**

De tijdsduur van deze toon wordt hiermee niet aangegeven. Die moet je apart met een **delay()** aangeven.

Wanneer je wil dat de toon stopt dan moet je de functie **noTone(pin)** gebruiken in jouw code.

Test deze code eens:

toon_na_drukknop

```
#define SPEAKER 19
#define S1 14
int t= 0;

void setup()
{
  pinMode( S1 , INPUT_PULLUP);
  pinMode( SPEAKER , OUTPUT);
}

void loop()
{
  if (digitalRead(S1)== LOW)
  {
    tone(SPEAKER, 600+t);

  }
  else
  {
    noTone(SPEAKER);
  }
}
```

In dit geval is er enkel geluid wanneer de laag actieve knop wordt ingedrukt. Probeer deze code eens uit en pas de hardware aan indien nodig!

politie_sirenev2

```
#define SPEAKER 19
#define S1 14
int t= 0;

void setup()
{
  pinMode( S1 , INPUT_PULLUP);
  pinMode( SPEAKER , OUTPUT);
}

void loop()
{
  if (digitalRead(S1)== LOW)
  {

    for(t=0;t<800;t++){
        tone(SPEAKER, 600+t);
    }
    for(t=800;t<0;t--){
        tone(SPEAKER, 600+t);
    }

  }
  else
  {
    noTone(SPEAKER);
  }
}
```

Voorbeeld van sirene met een buzzer. Test ook deze code eens uit.

6. Uitdagingen voor de buzzer:

1. Maak een eigen compositie van een liedje
2. Telkens je op de knop S1 of S2 drukt hoor je een andere toon
3. Als je op S1 drukt dan knipperen om beurten 2 LEDs en we horen een politie sirene
4. Maak een programma waarbij je tonen laat horen **zonder dat je de Tone() functie** gebruikt. We hebben deze functie nodig op het moment als we later ook RC6 (de afstandsbediening) willen gaan gebruiken. Dit omdat Arduino voor beide functies eenzelfde timer gaat gebruiken (meer uitleg zie gevorderde cursus).

```
void beep(byte pin) {
  byte cnt;

  //beep at 1kHz
  for(cnt = 0; cnt < 200; cnt++) {
    digitalWrite(pin, HIGH);
    delayMicroseconds(500);
    digitalWrite(pin, LOW);
    delayMicroseconds(500);
  }
}
```

Voorbeeld van 1KHz toongenerator. Merk op dat deze code reeds als een functie is geschreven (zie hierover verder in de cursus). I.p.v. **delay()** dewelke is uitgedrukt in milliseconden gebruiken we nu een **delayMicroseconds()** om de 1KHz te bekomen.

5. Meet het buzzer signaal met een **LA of een digitale scope**. Hoe ziet het signaal eruit? Stuur verschillende frequenties naar de buzzer en merk het verband op tussen frequentie en breedte van de pulsen.
6. Maak een **gehoortester**. Deze moet 2 knoppen hebben. Hiermee kan je de frequentie **hoger (UP) en lager (DOWN) instellen**, in stappen van 50Hz.
   Wanneer je op de derde knop **RESET** drukt springt de tester terug naar 1000Hz.
   Er zal een **rode LED** aan gaan als de minimum (<50Hz) en maximum waarde (>20000Hz) is bereikt.
   De frequentie wordt steeds getoond op de **serial monitor**. Als de grenzen zijn bereikt wordt dit ook getoond op het scherm.

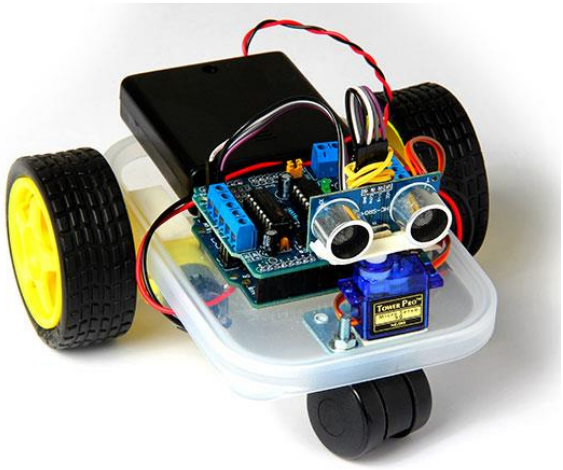
@ 4EE : bouw nu eerst jouw eigen Arduino UNO en Arduino robot. Zie hiervoor de aparte Arduino UNO bundels.

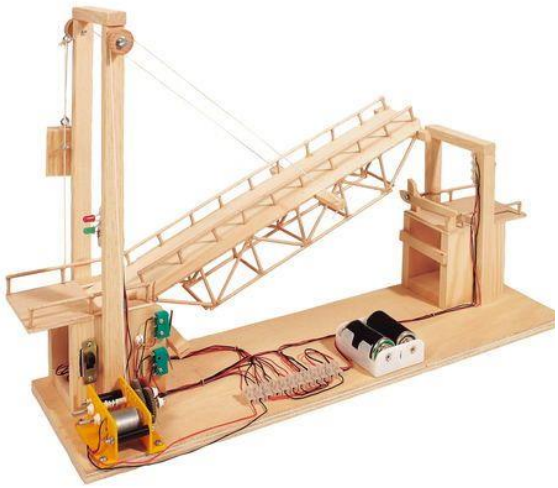
(in de gevorderden deel 2 cursus mag de cursist deze UNO zelf solderen en leren gebruiken).

10. DC Motoren aansturen (zonder/met PWM)

Tot nu toe hebben we ons vooral bezig gehouden met het inlezen van digitale schakelaars. We stuurden LEDs, een buzzer en een Serial Monitor aan.

Nu willen we graag **een DC motor** gaan aansturen. Zodoende kunnen we een beweegbare opstelling gaan controleren. Dit kan bijvoorbeeld een ophaalbrug, lift, Arduino robotje of zelfs een drone zijn.





1. Eerst sturen we een DC motor (gelijkstroommotor) aan met **enkel DC signalen**.

We zouden kunnen kiezen om **1 motor in 1 richting** aan te sturen.

Probleem is meestal dat de motor te veel stroom vraagt, zeker bij opstart, dan onze Arduino pin kan leveren (max 40mA, zie hiervoor de technische tabel van de Arduino). Een 5V DC motortje kan makkelijk **600mA** vragen wanneer hij een zwaar gewicht vooruit moet duwen (zie PIC robot).

Dus hebben we een **elektronische schakelaar** nodig dewelke deze stromen en eventueel hogere spanningen aan kan. Een motor is ook een inductieve belasting, dus we hebben hier ook te maken met inductie stromen (wanneer de schakelaar wordt afgezet wil de stroom nog voortvloeien in de schakelaar). Oplossing hier is van een **vrijloopdiode** te gebruiken. Deze diode staat omgekeerd geschakeld over de motor en zal de inductie stromen opvangen wanneer de schakelaar open gaat.

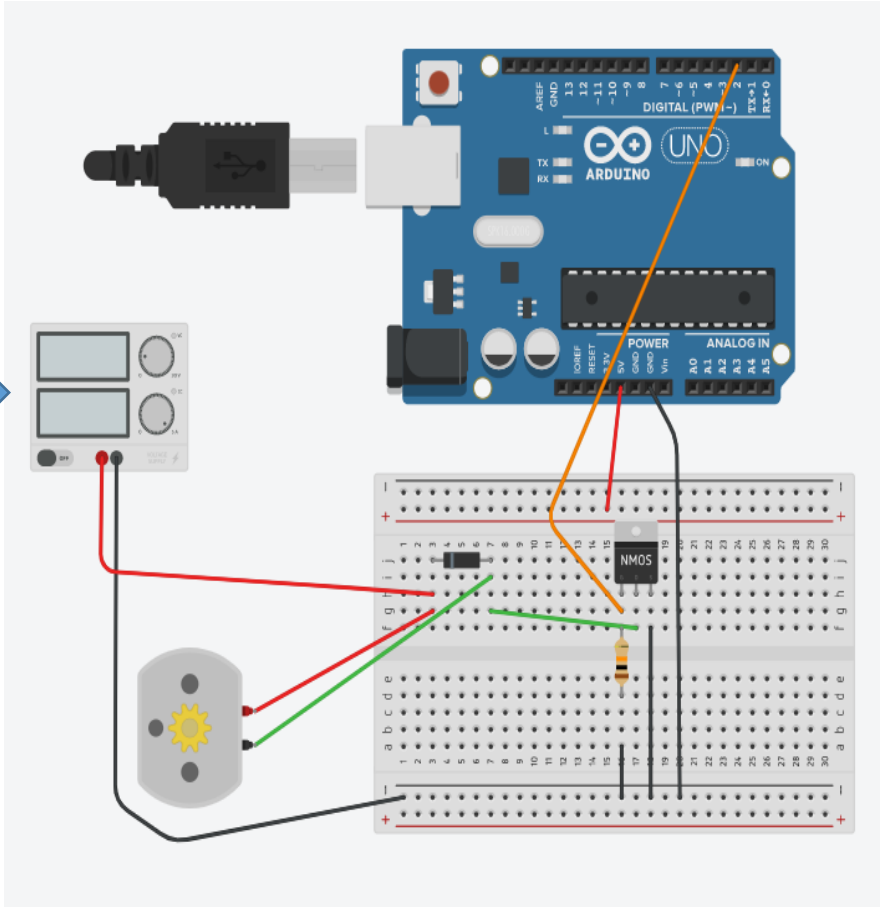
Een **N-kanaals** zelfsperrende **MOSFET** kunnen we inzetten als schakelaar. Wanneer we op de gate, een **5V spanning aanleggen** gaat de MOSFET sluiten en de **motor draaien**.

Wanneer er **geen spanning** wordt aangesloten (massa) **spert** de MOSFET en **stopt dus de motor**.

Het inwendig schema van de MOSFET toont een vrijloopdiode die antiparallel geschakeld is t.o.v. de MOSFET.

Bestudeer en bouw de volgende breadboard opstelling:

5V,
700mA



Tips:

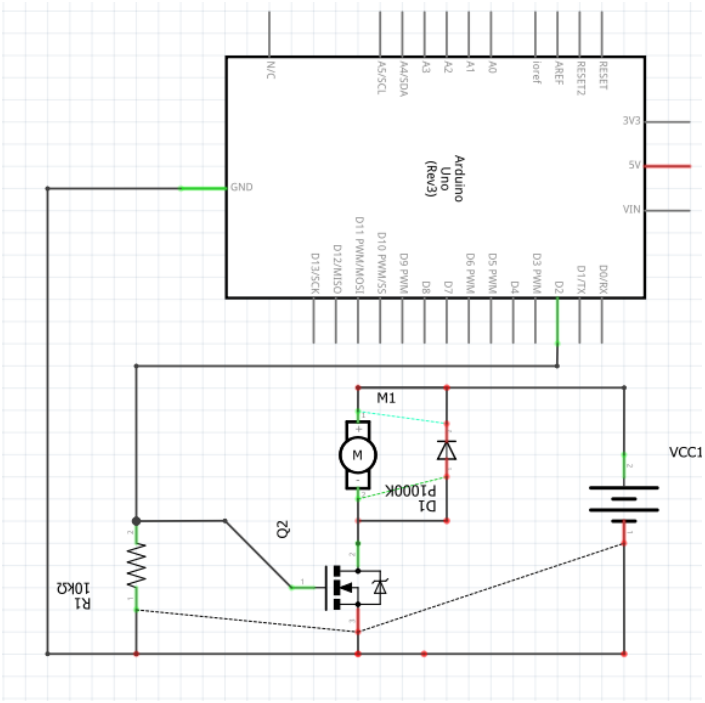
Hang de MOSFET gate (groene draad) met 10K aan de massa. Als de spanning moet wegvallen aan de Arduino, dan kan de MOSFET niet spontaan starten.

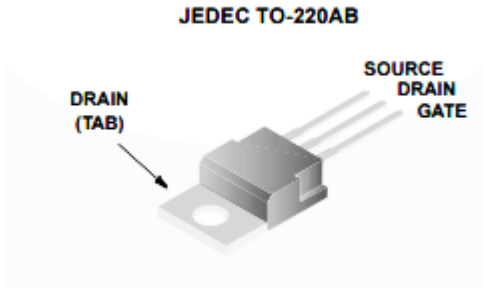
Sluit de 5V voeding van de motor aan op een externe voeding (anders trek je de stroom door de UNO voeding)

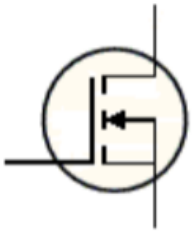
Massa's wel verbinden tussen voeding, breadboard en Arduino.

In deze opstelling is een **RFP12N10L of IRF520 MOSFET** gebruikt om een 5V DC motor aan te sturen. De gate wordt via pin 2 aangestuurd. Je kan ook eerst enkel met rechtstreeks +5V op de gate de MOSFET testen.

Schema van de opstelling:

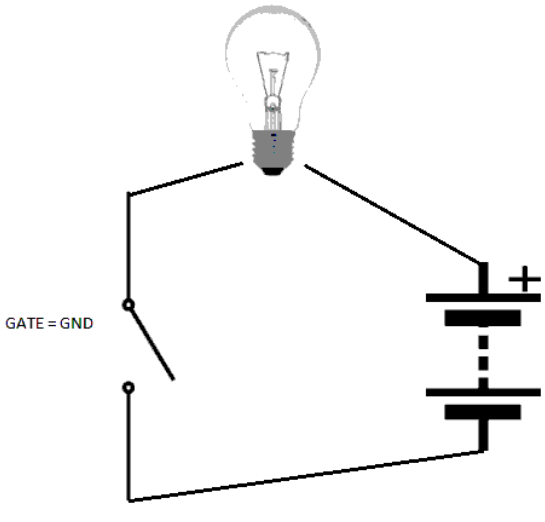


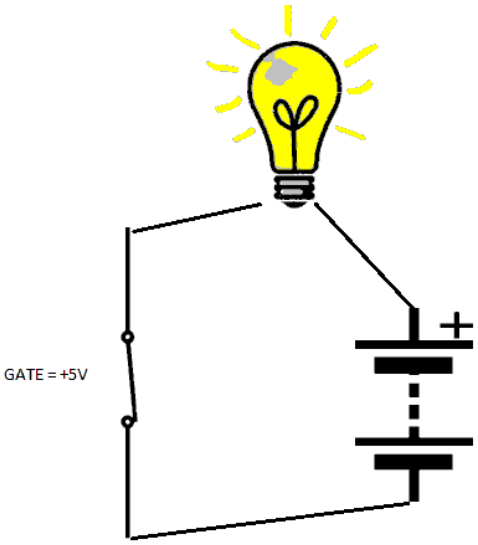




N-kanaal

Principe werking MOSFET:





We sluiten de MOSFET aan als schakelaar

Wanneer we +5V op de gate aansluiten gaat de MOSFET sluiten

(= gesloten verbinding tussen DRAIN en SOURCE)

Wanneer we GND op de gate aansluiten gaat de MOSFET open

(= open verbinding tussen DRAIN en SOURCE)

Let op de juiste aansluitingen!!!

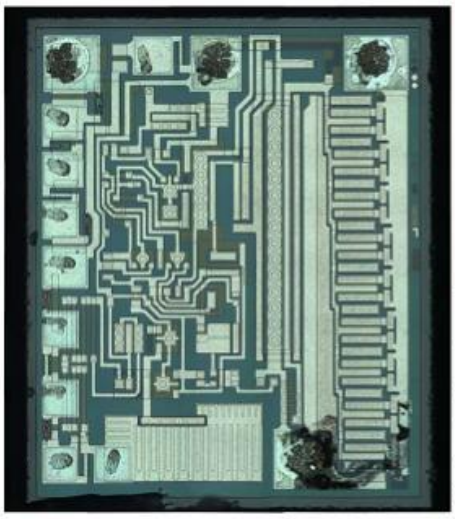
Belangrijke opmerking over ESD bij MOSFETS!

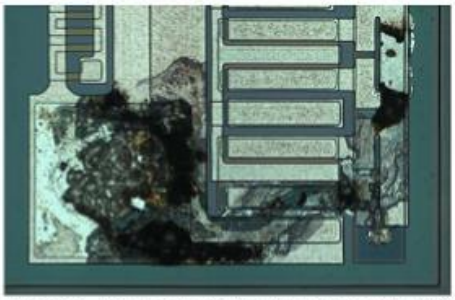
Het menselijk lichaam bezit een capaciteit van 100 tot 150pF. Deze "capaciteit" kan ladingen ophopen tot b.v. 1µC.
Indien deze lading door aanraking overgaat naar de ingangscapaciteit van de mosfet (b.v. 10pF)

dan wordt de poortspanning $U_{GS} = \frac{Q}{C_{GS}} = \frac{10^{-6}}{10 \cdot 10^{-12}} = 100.000V$

Deze componenten worden daarom in een antistatische verpakking geleverd en moeten met zorg behandeld worden als ze in de schakeling worden gebracht. Zo kunnen de elektroden uitwendig kortgesloten zijn en is het best met een geaarde laagspanningssoldeerbout te werken. De kortsluiting wordt dan verwijderd nadat de FET is aangesloten.
De ingang van de mosfet nooit „open" laten liggen in de schakeling.
Bij de meeste mosfets beschermen inwendige zenerdioden tegen ESD (electrostatic discharge). De fabrikant geeft de spanning op waarbij de mosfet beschermd is tegen ESD. Hierbij gaat men vaak uit van een "human body model" van 100pF parallel met 1,5 kΩ.

Voorbeeld van gevolgen van ESD:





Images Courtesy of Bunny Studios LCC

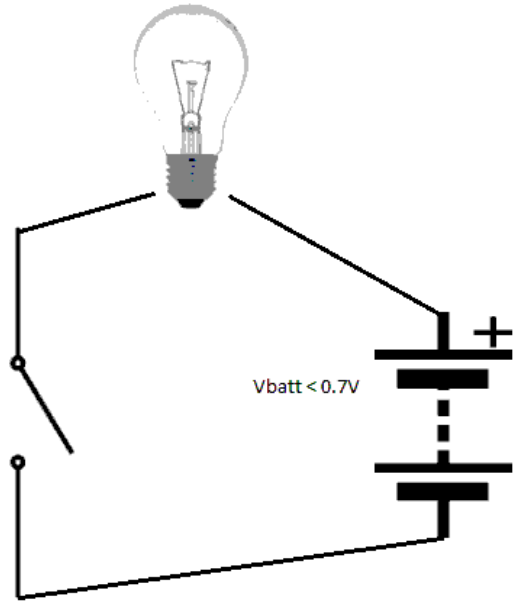
This is an example of latent ESD, also known as "Walking Wounded". This three terminal regulator IC worked about an hour after the initial ESD damage.

Bovenstaande figuur geeft aan wat ESD intern in een chip kan veroorzaken zonder dat we het gezien hebben. Dus NOOIT zomaar de pootjes van de ESD aanraken! Ofwel allemaal samen,, ofwel je aarden via een draad aan een werkbank of voeding (groene aansluiting).

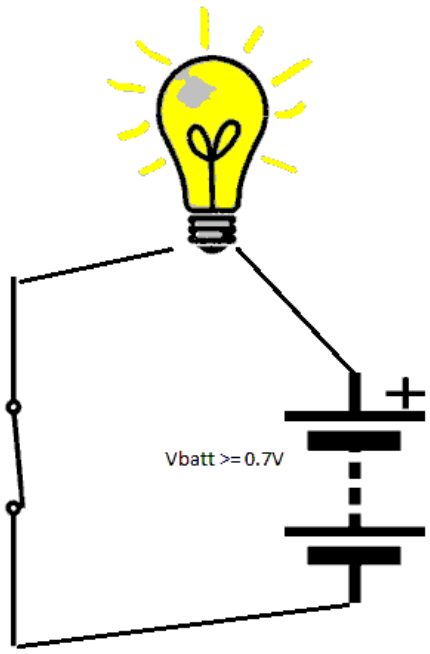
De volgende **logo's** geven aan dat de componenten ESD gevoelig zijn:

These are some of the more common logos associated with anti-static labels. They are used to inform the user that the contents are static sensitive.

Principe werking diode:



Diode in doorlaat, maar Vbatt is < 0.7V



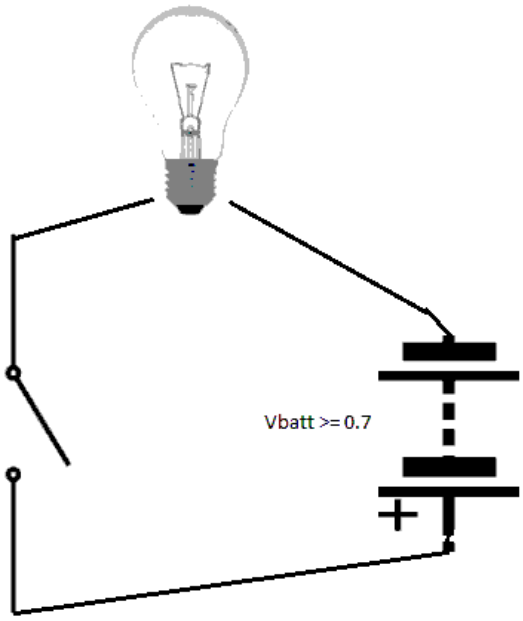
Diode in doorlaat, Vbatt >= 0.7V

Sluit de diode aan in doorlaat richting
(+ voeding aan Anode, GND aan Kathode)
Als Vbatt < 0,7V spert de diode (open)

Als Vbatt >= 0,7V geleidt de diode (gesloten)

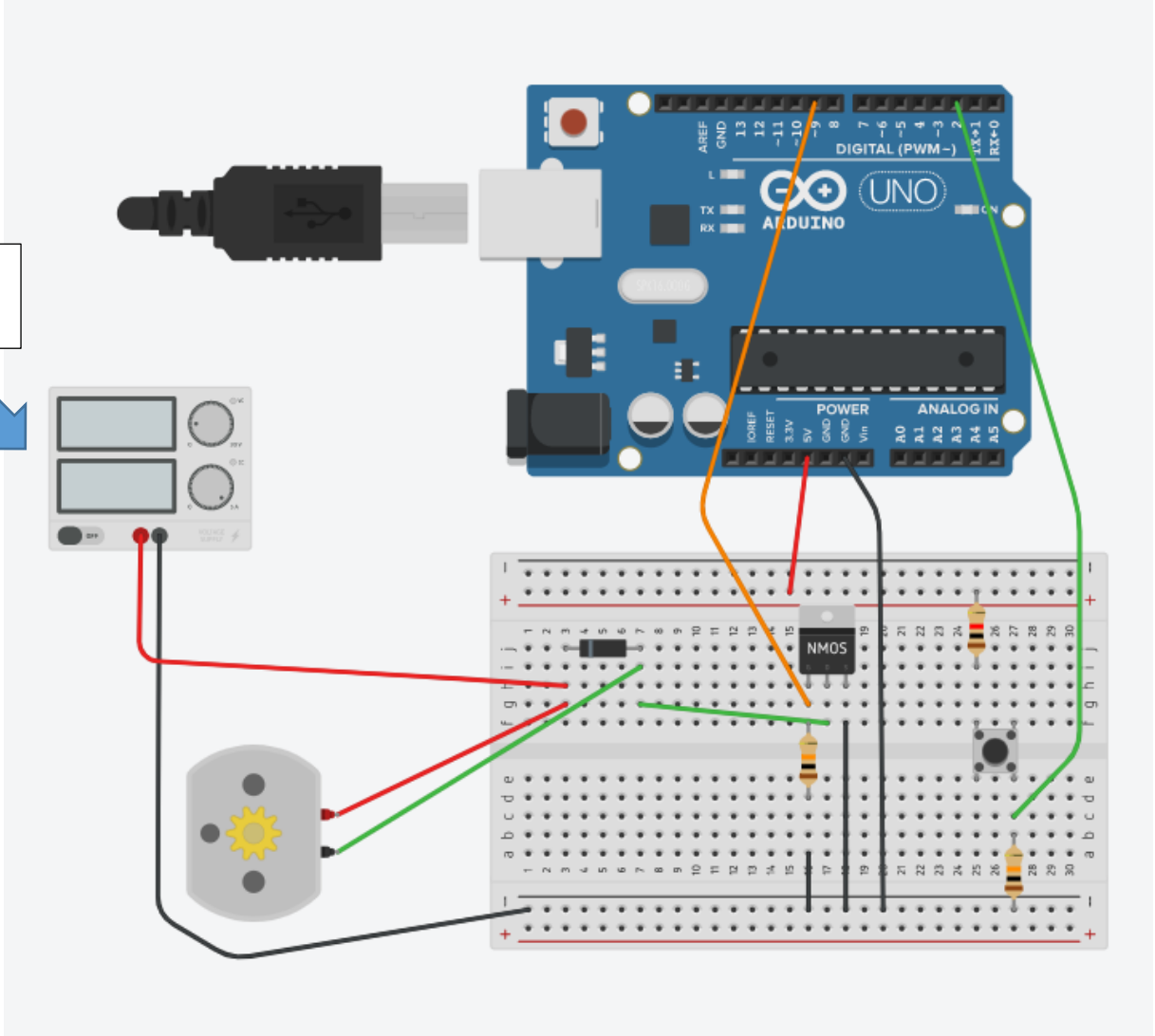
Als de diode in sper richting is aangesloten geleidt deze ook niet!
(+ voeding aan Kathode , GND aan Anode)

Een motor kan inductiespanningen/stromen opwekken. Deze moeten we kwijt via de vrijloopdiode



Diode in sper aangesloten

Voeg nu een hoog actieve schakelaar toe op pin 2 terwijl de MOSFET wordt aangestuurd via de gate op pin 9. De voeding is maximum 6V voor de motor.



Test deze opstelling uit en gebruik de volgende code:

In c-code ziet de sturing van de motor er dan als volgt uit:

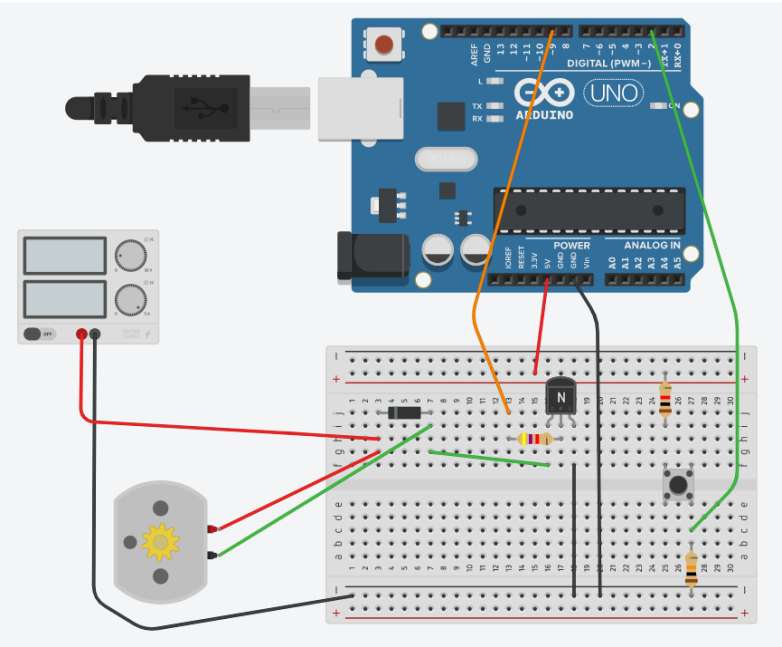
Niet gebruikte motoren steeds in de void setup() op LOW zetten!

motor_mosfet_direct1 §

```
void setup()
{
  pinMode(2, INPUT);
  pinMode(9, OUTPUT);
}

void loop()
{
  if (digitalRead(2) == 1) {
    digitalWrite(9, HIGH);
  } else {
    digitalWrite(9, LOW);
  }
  delay(10); //antidender
}
```

Uitdagingen:

1. Stuur 1 DC motor aan. Hij mag enkel draaien als **beide knoppen** tegelijk zijn ingedrukt. Enkel als aan de voorwaarde is voldaan wordt er op de **serial monitor** een tekst getoond “we zijn aan het draaien” en hoor je een **buzzer** piepen bij 1KHz. Ondertussen knipperen ook nog **2 rode LEDs.**
2. EXTRA: Stuur nu de DC motor aan met een **transistor** ipv een MOSFET. Je kan hiervoor gebruik maken van de BC547C.



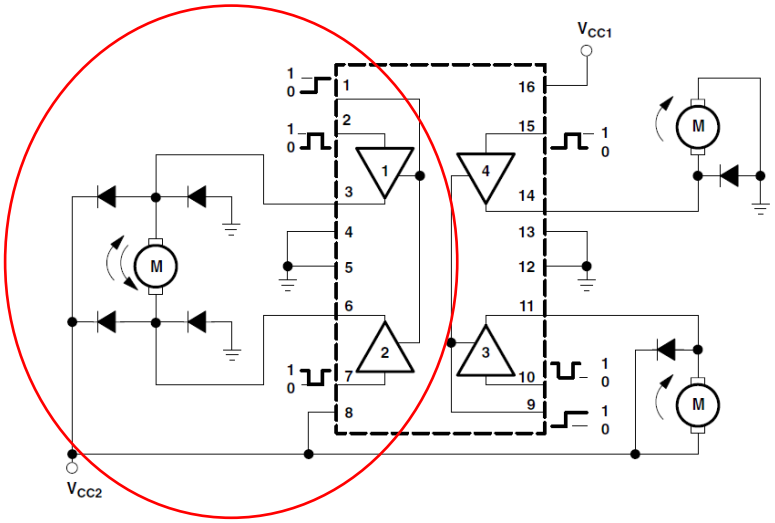
Merk op dat de transistor best warm kan worden (kan maar 100mA aan). Voor zwaarder werk best een andere transistor kiezen of voor de MOSFET gaan.

Teken het schema van de bovenstaande schakeling en berekening de weerstand van de basis transistor (5EE only):

2. Motor in 2 richtingen aansturen.

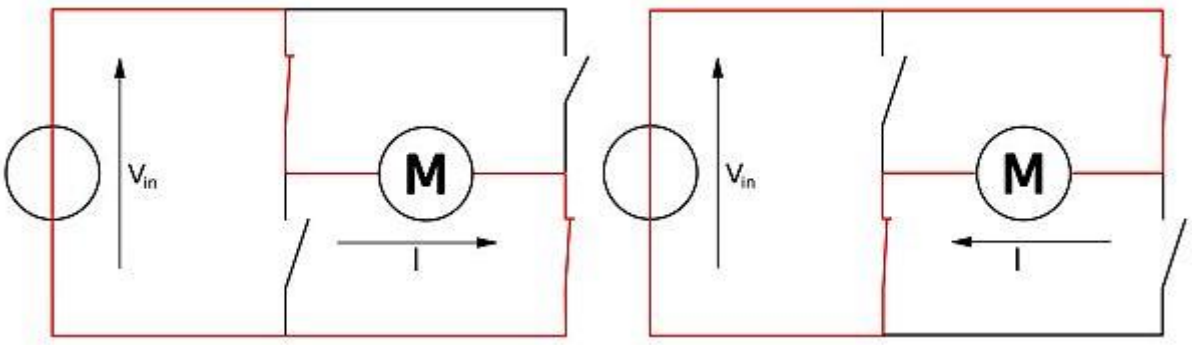
Merk op dat de motor maar **in 1 richting** kan draaien. De polariteit is m.a.w. niet omkeerbaar.

Vandaar dat we liever kiezen voor **de L293D chip**. Dit is een **motordriver** die 4 MOSFETS bevat.



We kiezen voor de H-brug opstelling, voorgesteld links op bovenstaande tekening.

Hoe werkt een H-brug?



We kunnen de MOSFETS voorstellen als schakelaars. Wanneer je 2 schakelaars sluit, voorbeeld links boven en rechts onder, dan gaat de stroom **in 1 richting vloeien** door de motor en beginnen draaien.

Wisselen we de schakelaars om (links onder en rechts boven), dan draait de motor **in de omgekeerde richting**.

Merk op dat de L293D is zo ontworpen dat je **nooit beide schakelaars aan 1 zijde** kan sluiten op het zelfde moment. Anders zou er een **kortsluiting** komen en de chip opgeblazen worden door een te grote stroom.

De aansluitingen van de driver chip **L293D** vind je hier:

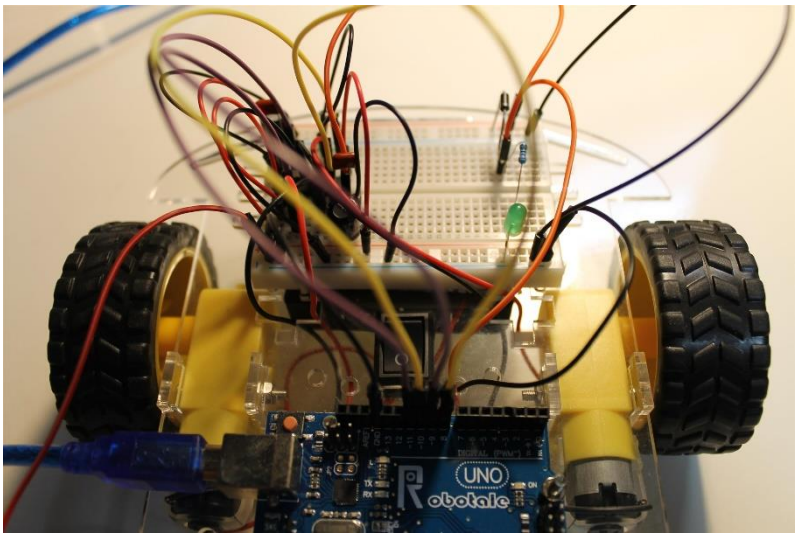
L293D pin functie	Pin nummer	Signaal
Enable motor drivers	1,9	5V
Motor Links inputs (sturing)	2, 7	5V, massa of PWM
Motor Rechts inputs (sturing)	10,15	5V, massa of PWM
Motor Links outputs	3,6	0 – 36V, 600mA
Motor Rechts outputs	11,14	0 – 36V, 600mA
VCC chip aansturingsdeel	16	5V
GND	4,5,12,13	Massa
VCC motorspanning	8	Zie motorspanning (0 – 36V)

Doel: 2 DC motoren vooruit en achteruit laten draaien met een **zelfgebouwde H-brug**

Benodigdheden:

- Robotchassis met 4AA batterijen in houder + schakelaar
- Draden
- L293D chip
- 2 x 100n condensator
- 2 x 220µF condensator
- 2 DC motoren
- 1 groene LED + 220 ohm weerstand
- 1 diode (schottky type)

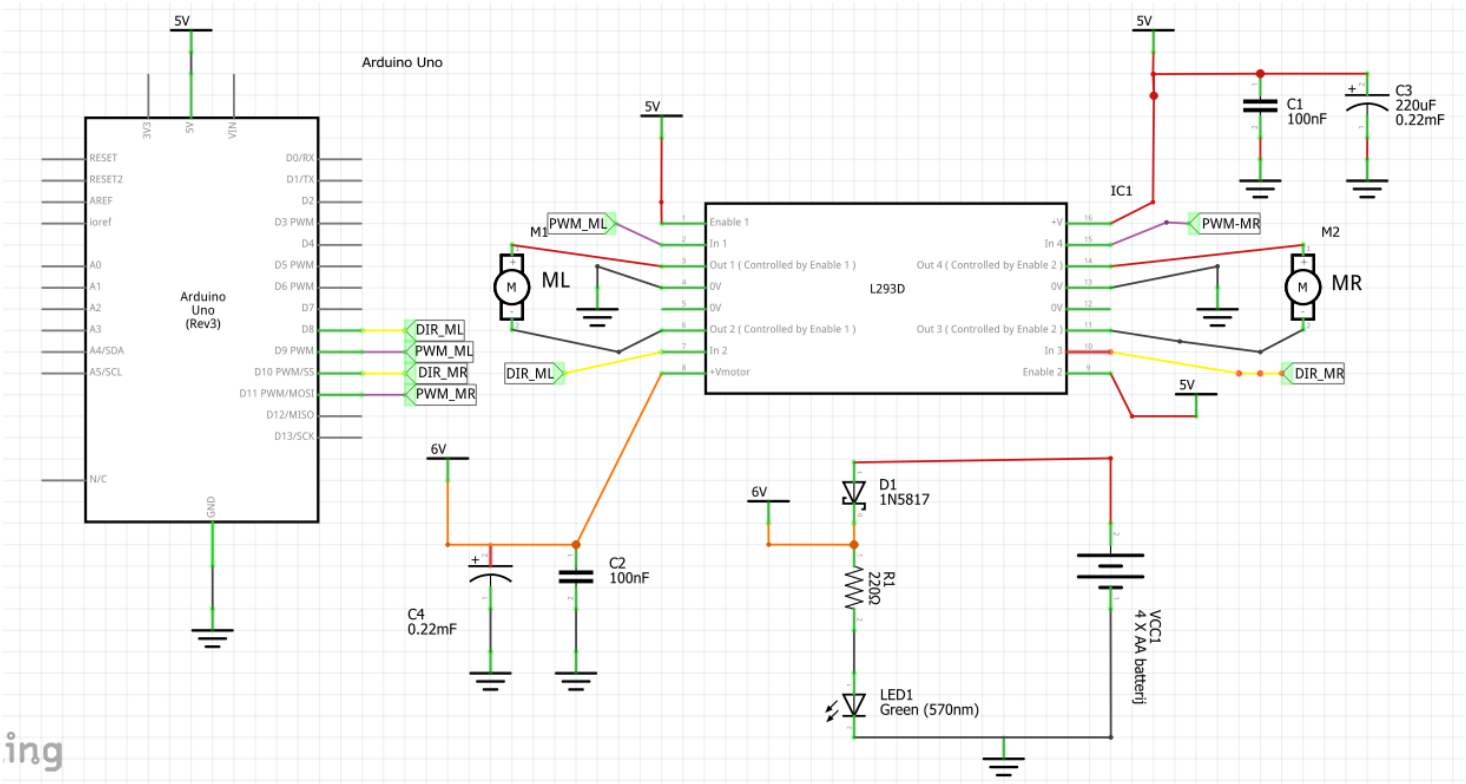
2.1 Finale opstelling van de schakeling:



Dit is de ganse motorbedrading. Schrik niet, we gaan dit stap per stap opbouwen.

Alternatief voor deze spaghetti is een motor shield gebruiken. Hierbij gebruik je een vooraf bedrade L293D op een PCB dewelke je dan enkel nog moet aansluiten (zie verder).

2.2 Schema van de L293D opstelling voor 1 DC motor:

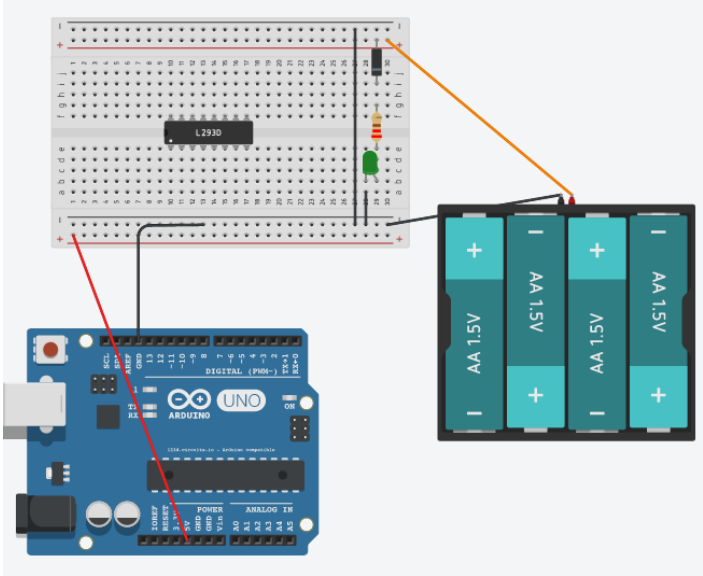


Bovenstaand schema geeft een overzicht van de aan te sluiten componenten op de Arduino en de L293D. Dit schema kan als een referentie gebruikt worden voor het volgende breadboard stappenplan.

2.3 Breadboard opstelling voor 2 DC motoren:

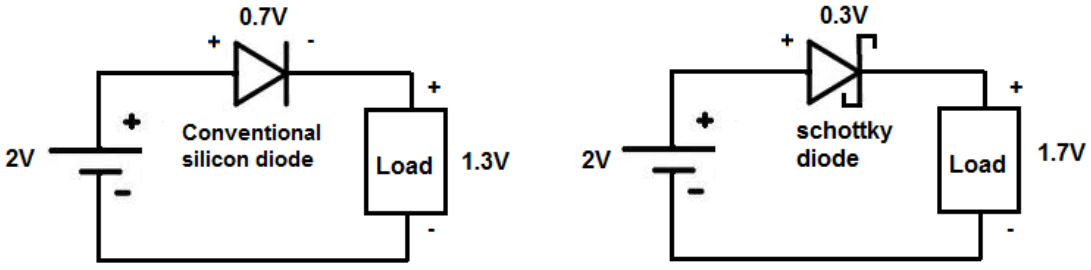
We bouwen de L293D schakeling op in verschillende fases.

2.3.1 Voorzie het breadboard van de **juiste voedingsspanningen**

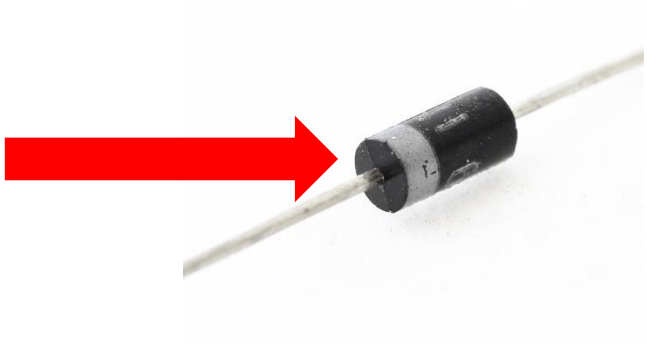


Via de 4 AA batterijen in serie kunnen we het breadboard voeden met een 6V (4 x 1.5V). Om onze batterijen te beschermen tegen verkeerd aansluiten (+ en – omkeren = verkeerde polariteiten aansluiten) moeten we eerst een **diode** voorzien.

Een diode is net als een LED. Ze heeft **polariteiten** en kan dus slechts in 1 richting maar geleiden. Ze kan uiteraard geen licht geven. We zijn over de 1N5819 een 0.3V kwijt. Dit omdat het een type **Schottky diode** is. Standaard diodes hebben een spanningsval van 0.7V.



De **min** op de diode wordt aangeduid met een **grijs streepje**!



Sluit de diode aan met zijn **"+" kant** aan de **positieve kant** van de 6V batterij pack. Dit is de draad die na de schakelaar in het robotchassis volgt.

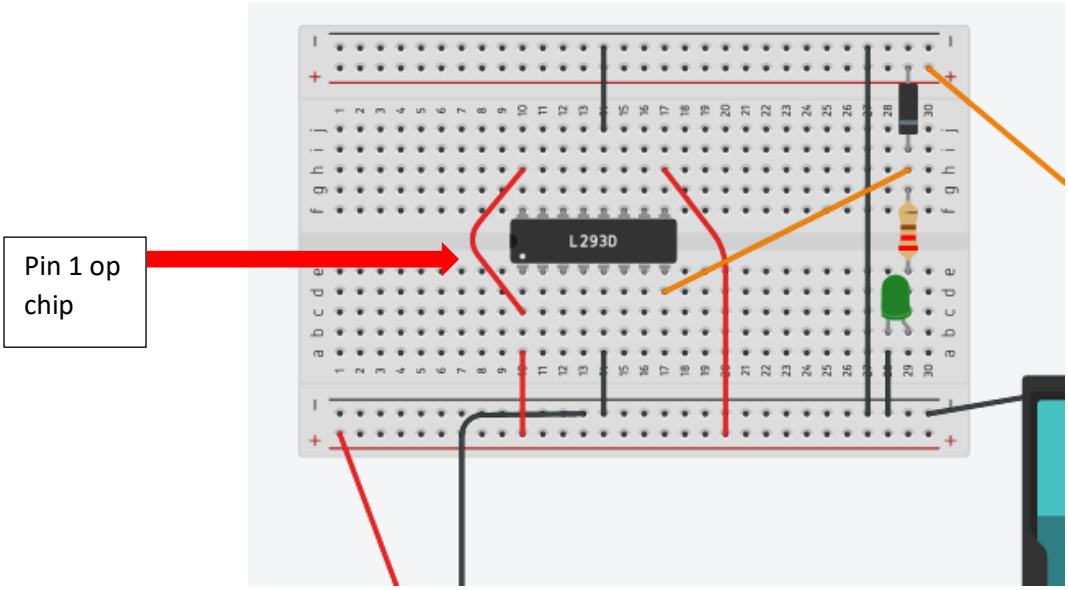
Om te controleren of de batterij spanning juist is aangesloten en nog voldoende werkt sluiten we een **groene LED met voorschakelweerstand** aan op deze spanning.

Merk op dat je de **massa van de Arduino verbindt met de massa van de batterij**. Dit is een **veel voorkomende fout als je dit vergeet.** Wanneer je dit vergeet gaat straks de schakeling niet correct werken. De referentiespanning is namelijk zwevend als je de massa's niet verbindt met elkaar.

De **+5V van de Arduino** voorzien we op de andere kant van het breadboard. Mogelijk wordt deze afgetapt van het andere breadboard.

Test de groene LED door de schakelaar te bedienen. Deze moet branden als de schakelaar op "I" staat.
Wanneer we gaan werken aan het breadboard moet de **spanning steeds eerst afgezet** worden!

2.3.2 De L293D van de **juiste voedingsspanningen en enables** voorzien

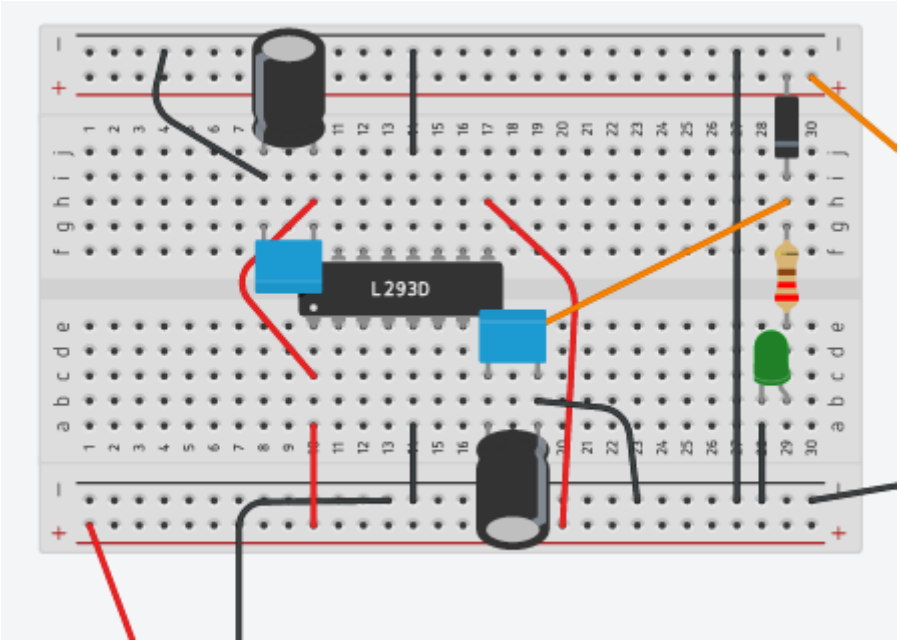


Plaats de L293D chip op het breadboard. De chip moet je plaatsen over het middengedeelte van het breadboard. Zorg dat **pin 1** (het bolletje of inkeping op de chip) op dezelfde plaats staat als op bovenstaande figuur!

Sluit alle 5V en massa draden aan op de L293D zoals boven aangegeven. De **batterijspanning** wordt **na de diode afgetapt** en verbonden met **pin 8** van de chip.

Ook beide enable pinnen 1 en 9 (pinnen die toestaan dat de driver gaat werken) moeten aan +5V hangen.

2.3.3 De voedingsspanning van **extra condensatoren** voorzien



We gaan een **electrolytische condensator** van 220µF plaatsen over de voeding van de L293D logica (pin 16) en de voeding voor de motoren (pin 8).

Deze elco doet dienst als **buffer** en vangt de grote stroompieken op bij het starten van de motoren. Zodoende wordt de batterij niet direct platgetrokken.

Merk op dat deze condensator **polariteiten** heeft. De **min** kant, dewelke aan de massa moet aangesloten worden, is voorzien van een **grijze streep**. De min kan heeft ook een **korter pootje**.

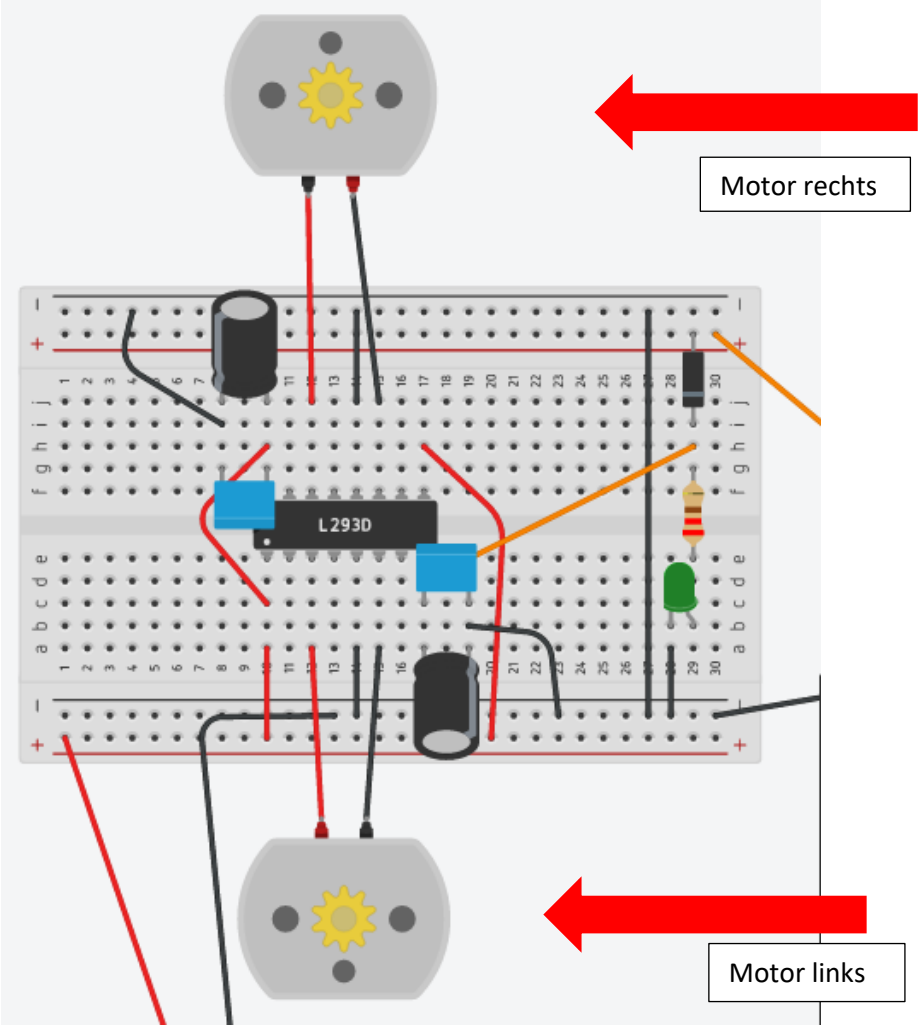
Respecteer dit polariteitsverschil! Anders zou de elco kunnen ontploffen wanneer je deze onder spanning zet!

Het is handiger als je de **pootjes gelijk knipt** alvorens de elco op het breadboard te plaatsen.

We voorzien ook een **ceramische condensator** van 100n op dezelfde voedingspinnen van de L293D. Deze dienen om **hoog frequent storingen** af te leiden van de voeding naar de massa. Deze condensators hebben **geen polariteit**. De waarde wordt aangegeven op de condensator met 104 (100 000pF = 100nF).

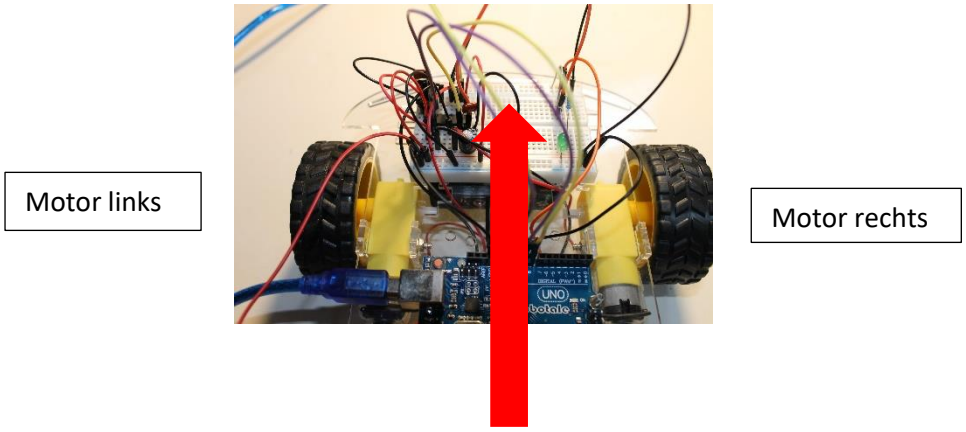
2.3.4 De DC motoren aansluiten

Nu moeten we de **2 DC motoren** nog toevoegen aan de L293D. Normaliter zijn deze al voorzien op het breadboard tijdens de opbouw van het robotchassis. Check na of de 4 draden op de juiste plaats zijn aangesloten!



Motor links sluiten we aan op pin 3 (zwart) en pin 6 (rood).

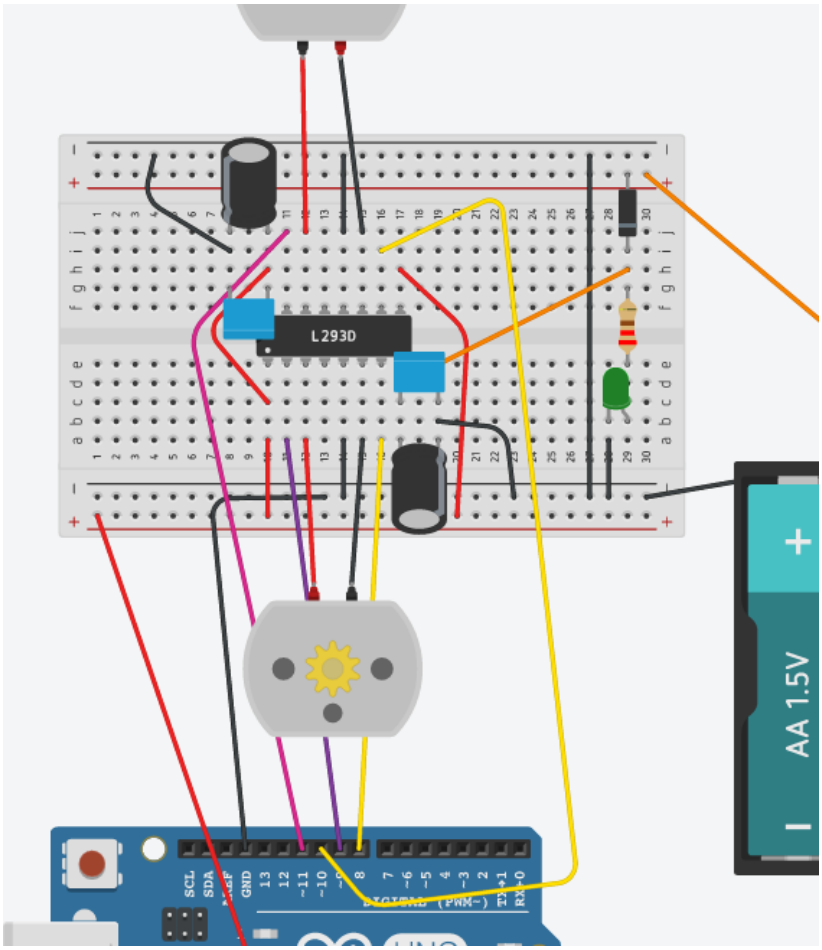
Motor rechts sluiten we aan op pin 14 (zwart) en pin 11 (rood).



2.3.5 De aansturing van de Arduino op de driver voorzien

Tot slot moeten we in deze ganse spaghetti ook **nog 4 draden toevoegen** om de motoren te kunnen aansturen via de Arduino.

De aansluitingen op de L293D vind je terug in de L293D tabel.



Arduino	Pin	L293D	pin
PWM_ML	9	Input PWM_ML	2
DIR_ML	8	Input DIR_ML	7
PWM_MR	11	Input PWM_MR	15
DIR_MR	10	Input DIR_MR	10

Als alles goed is aangesloten kunnen we nu van start gaan met het programmeren.

Merk op dat een slecht contact op het breadboard het resultaat snel kan beïnvloeden. Dus alle **draadjes goed nakijken of ze voldoende klemmen** in het breadboard.

De volgende fase hierna is in de elektronica een PCB ontwikkelen waar deze elektronica op gesoldeerd zit of een PCB met driver kopen als shield (zie eerder aangegeven).

3. Maak het volgende ARDUBLOCK programma om de motoren full speed vooruit en achteruit te laten draaien met een druk op de schakelaar.

Op pin 2 is een hoog actieve schakelaar aangesloten (zie dit hoofdstuk voor de hardware aansluiting).

Download de code in de Arduino.

Zet nu de batterijspanning van de robot aan door de **on/off knop op "1"** te zetten. De groene LED moet nu branden op het breadboard.

De robot rijdt nu **achteruit.** Wanneer je op de **schakelaar** aangesloten op pin 2 drukt zullen de motoren **vooruit** draaien.

Merk op dat we op dit moment de motoren full speed (de volle +5V) krijgen.

Je kan de motoren ook langzamer laten draaien. Dan moet je aan de slag met PWM (Pulse Width Modulation). Zie hiervoor verder in de cursus bij tips voor programmeren.

Tip: Wanneer je **slechts 1 motor nodig** hebt moet je er altijd voor zorgen dat **beide pinnen van de andere motor op LOW** staan (aan de massa liggen). Anders gaat deze motor onwillekeurig draaien en geeft dit niet het gewenste effect.

4. Hoe ziet de c-code eruit voor de motoraansturing?

motor_vooruit_achteruit.abp

```
void setup()
{
pinMode( 2 , INPUT);
pinMode( 10 , OUTPUT);
pinMode( 11 , OUTPUT);
pinMode( 9 , OUTPUT);
pinMode( 8 , OUTPUT);
}

void loop()
{
if (digitalRead( 2))
{
digitalWrite( 8 , LOW );
digitalWrite( 9 , HIGH );
digitalWrite( 10 , LOW );
digitalWrite( 11 , HIGH );
}
else
{
digitalWrite( 8 , HIGH );
digitalWrite( 9 , LOW );
digitalWrite( 10 , HIGH );
digitalWrite( 11 , LOW );
}
}
```

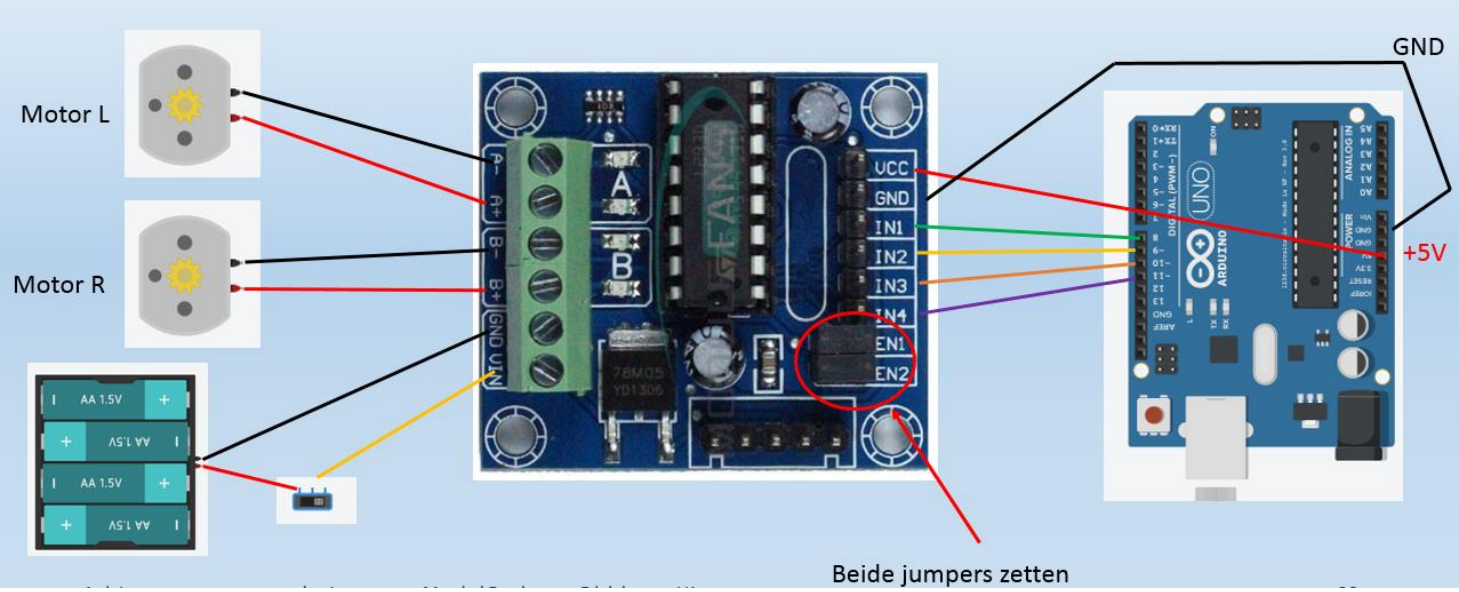
Je kan heel makkelijk uit de code aflezen wat er gebeurd met de motoren.

Ook hier wordt de functie **digitalWrite (pin number, value)** gebruikt om de motoren aan (HIGH = +5V) of uit (LOW = 0V) te zetten.

Merk op dat als je de MicroArdubot van EDULAB gebruikt, je pin 11 moet vervangen door pin 13 !!!

5. Motor in beide richtingen aansturen met **een L293D drivershield (chinese versie)**.

Wanneer je het bedraden van het breadboard te moeilijk vind kan je ook gebruik maken van een kant en klare oplossing via een drivershield.

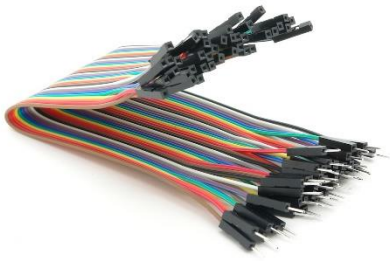


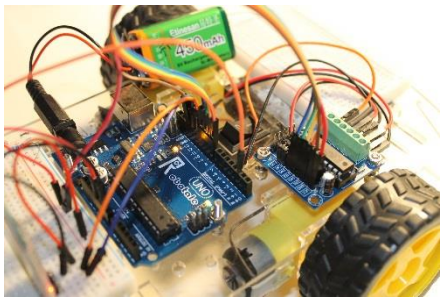
Deze figuur geeft aan hoe de shield moet worden aangesloten.

De motoren en batterijspanning moeten we met een kleine schroevendraaier vastzetten (let op de juiste aansluitingen.

De +5V en GND voor de sturing van de L293D chip tappen we af van de Arduino voeding.

De 4 draden voor het aansturen van de motoren zijn speciale kabeltjes (draad met mannelijke naar vrouwelijke connectoren).

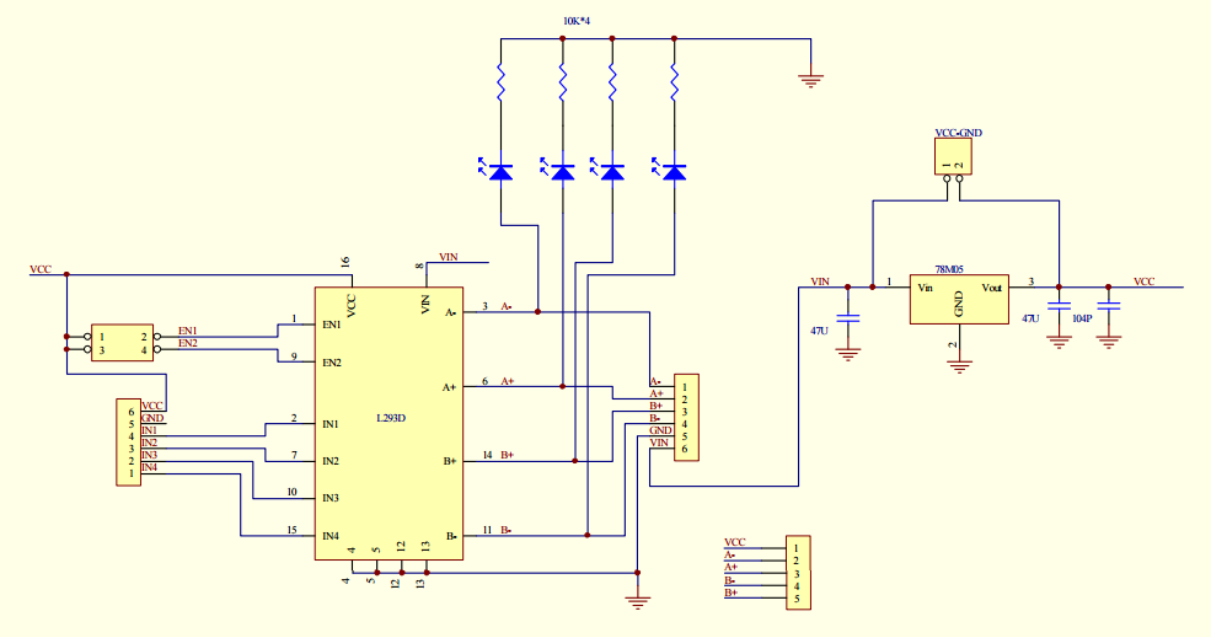




De aansluitingen van de Arduino op de L293D zijn de volgende:

L293D shield	Arduino
IN1	8
IN2	9
IN3	10
IN4	11

Detail schema van het motorshield:



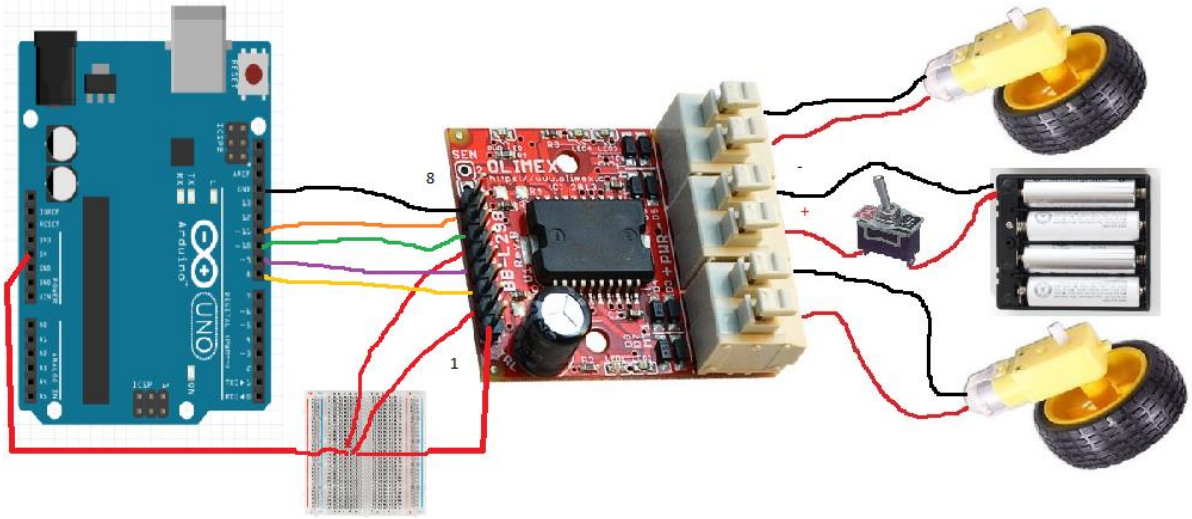
Merk op dat je zowel de schakelaar van de batterij op de robot (Vin), als de VCC moet uittrekken (+5V van de Arduino) om de motoren tot stilstand te houden en de L293D dus niet te voeden, terwijl je de +9V of de USB wil laten insteken in de Arduino.

Wanneer je enkel de schakelaar zou afzetten van wordt, eigenaardig genoeg, de L293D toch nog via de Arduino +5V gevoed en trekt de chip de stroom uit de +9V batterij of de USB van de Arduino.

6. Uitdagingen met motoren:

1. Voeg een schakelaar toe aan de hardware. Wanneer je op deze knop drukt rijdt de robot naar links. Laat je hem los dan stoppen de motoren. Druk je op de reeds aanwezige knop dan rijdt de robot naar rechts.
2. Voorzie 4 knoppen (elk voor 1 richting = vooruit, achteruit, links, rechts) en sluit ze allemaal digitaal aan. Telkens je op 1 van de knoppen drukt gaat de robot die richting uit. Druk je geen knop in, dan stopt de robot.
3. Voeg bij opdracht 2 nog 4 LEDs toe. Plaats 2 gele LEDs vooraan de robot en 2 rode achteraan. Wanneer de robot vooruit rijdt gaan de beide gele LEDs branden. Gaat hij achteruit gaan beide rode LEDs branden. Gaat de robot naar links, dan branden de linkse gele en rode LED. Gaat de robot naar rechts, dan branden de rechtse gele en rode LED.

Extra info: motor in beide richtingen aansturen met een **L298D drivershield van Olimex**.



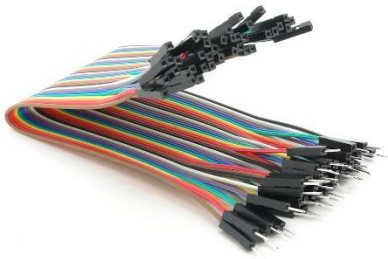
Wanneer je het bedraden van het breadboard te moeilijk vind kan je ook gebruik maken van een kant en klare oplossing via een drivershield.

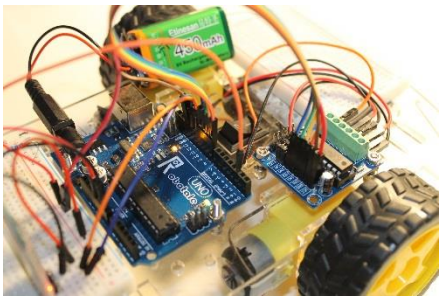
Deze figuur geeft aan hoe de shield moet worden aangesloten.

De motoren en batterijspanning moeten we klemmen in de aansluitingen.

De +5V en GND voor de sturing van de L298D chip tappen we af van de Arduino voeding. Ook de enable aansluitingen worden +5V. Deze zorgen ervoor dat dat deel van de chip aangestuurd wordt.

De 8 draden voor het aansturen aan Arduino's zijde zijn speciale kabeltjes (draad met mannelijke naar vrouwelijke connectoren).

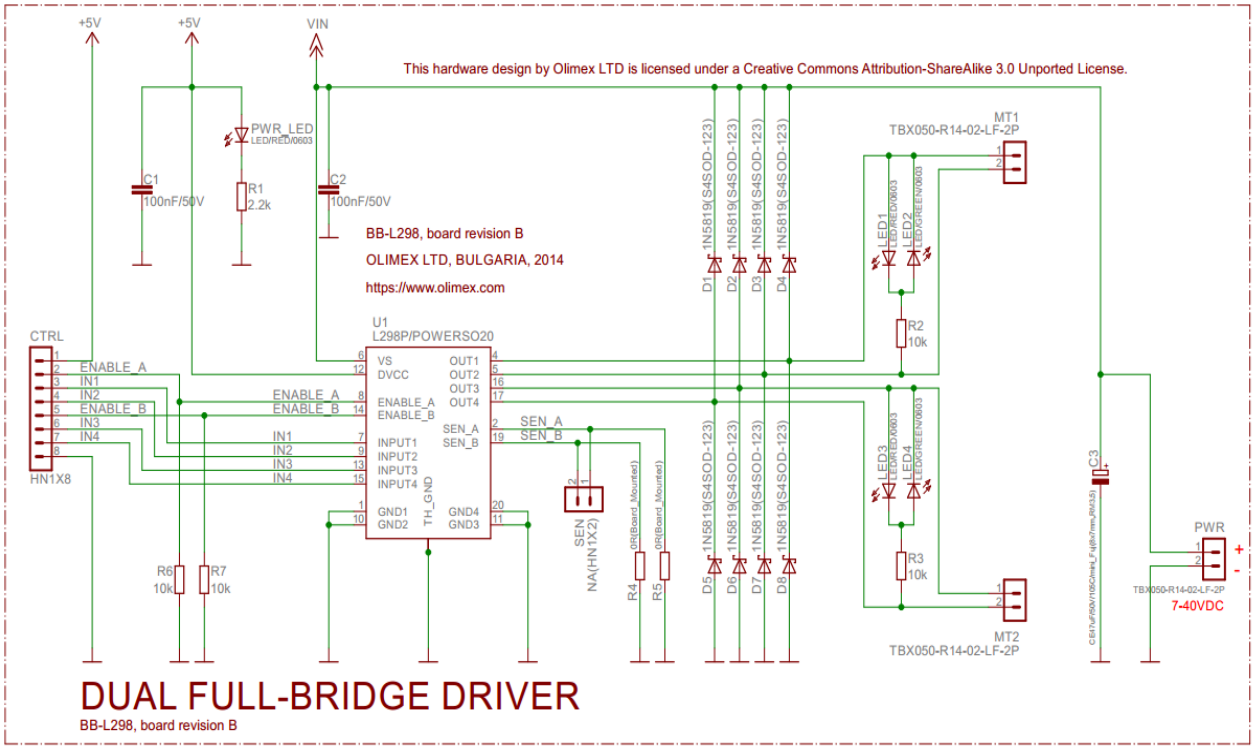




De aansluitingen van de Arduino op de L298D zijn de volgende:

L298D shield	Arduino
3	8
4	9
6	10
7	11
1, 2 en 5	+5V
8	GND

Detail schema van het motorshield:



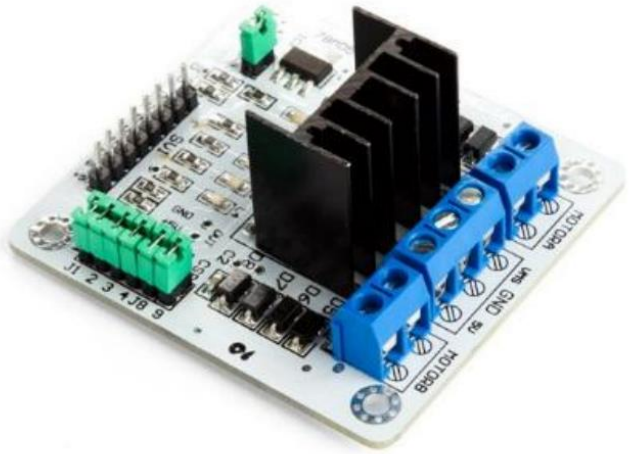
Merk op dat je zowel de schakelaar van de batterij op de robot (Vin), als de VCC moet uittrekken (+5V van de Arduino) om de motoren tot stilstand te houden en de L298D dus niet te voeden, wanneer je de +9V of de USB wil laten insteken in de Arduino.

Wanneer je enkel de schakelaar zou afzetten wordt, eigenaardig genoeg, de L298D toch nog via de Arduino +5V gevoed en trekt de chip de stroom uit de +9V batterij of de USB van de Arduino.

Extra info: motor in beide richtingen aansturen met een **VMA409 drivershield van Velleman**

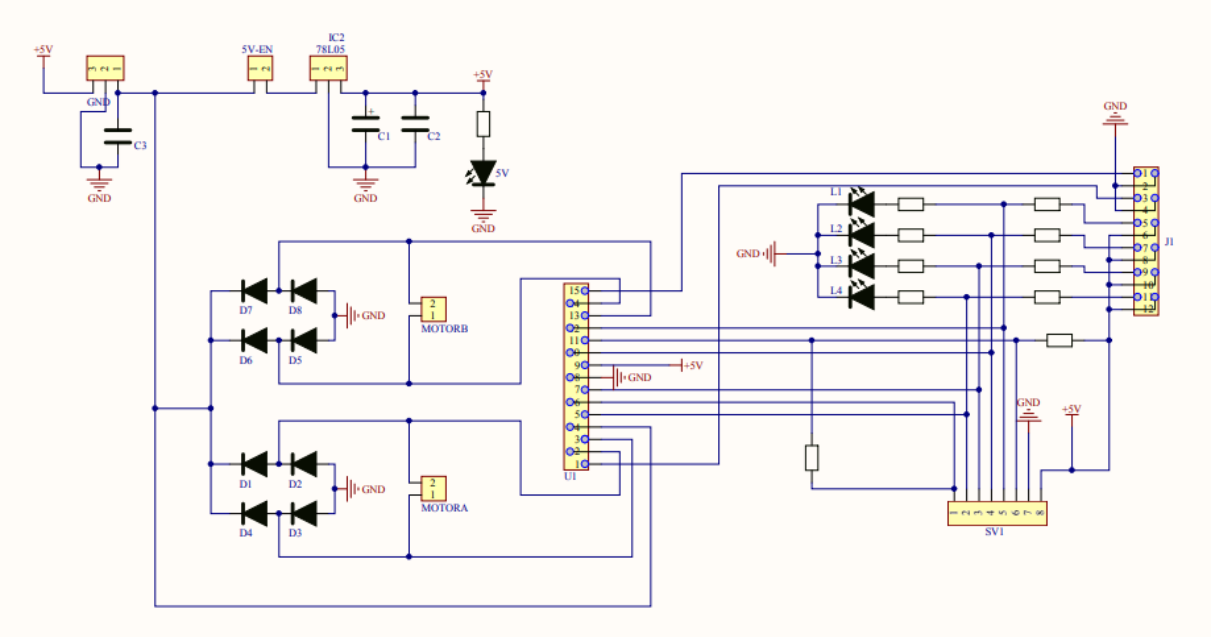
Ook hier is een L298 driver gebruikt.

De handleiding kan je vinden op https://www.velleman.eu/downloads/29/vma409_a4v01.pdf



Pin layout:

Pin Name	Description
MOTORA	motor 1
MOTORB	motor 2
VMS	5 VDC to 35 VDC
GND	ground
5V	power input for the logic circuit on the board
ENA	enable pin for motor 1
IN1	control pin motor 1
IN2	control pin motor 1
IN3	control pin motor 1
IN4	control pin motor 1
ENB	enable pin for motor 2
5V	5 V output
GND	ground
CSB	current test pin for motor 1; can be wired to a resistor for current testing or tied to a jumper to disable it
CSA	current test pin for motor 2; can be wired to a resistor for current testing or tied to a jumper to disable it
UR1	pull-up resistor
UR2	pull-up resistor
UR3	pull-up resistor
UR4	pull-up resistor
5V_EN	5 V source jumper; supplies power from the VMS port when the jumper is enabled; the power is supplied by the 5 V port when the jumper is disabled



Schema van de VMA409

VMA409

This module allows full control of two DC motors or one stepper motor.

driver	L298N
driver power supply	+5 V to +35 V
driver output current (max.)	2 A
logic power output Vss	+5 V to +7 V (internal supply +5 V)
logic current	0-36 mA
controlling level	low: -0.3 V to 1.5 V, high: 2.3 V-Vss
enable signal level	low: -0.3 V to 1.5 V, high: 2.3 V-Vss
max. power	25 W
working temperature	-25 °C to +130 °C
dimensions	69 x 56 x 36 mm

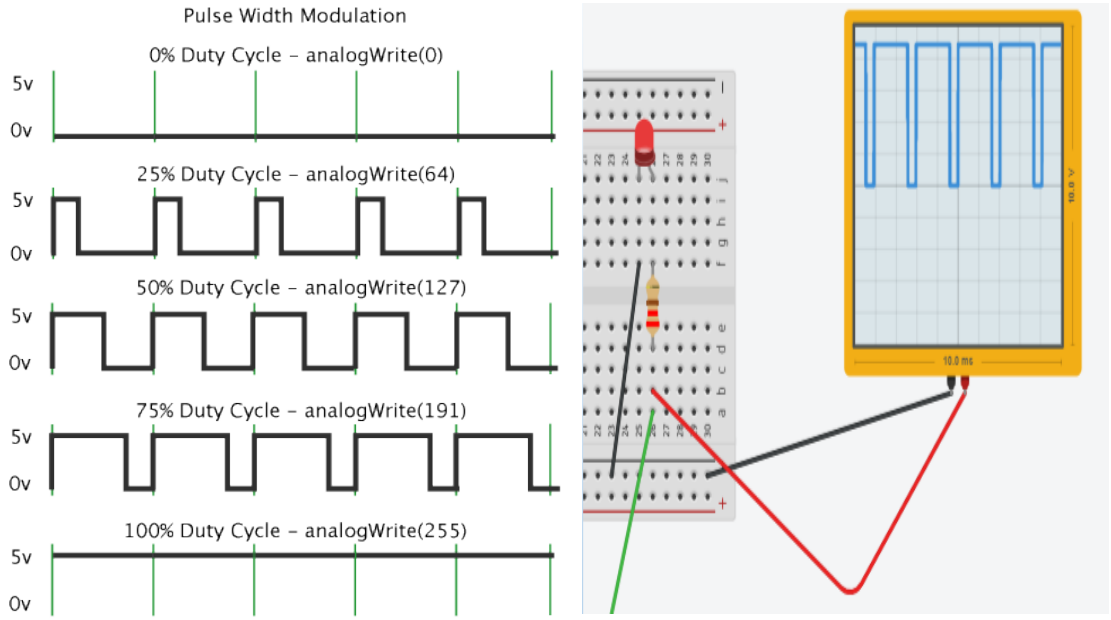
Aansluitingen?

VMA409	**Arduino**
VMS	5V van batterij voor motor
GND	Aan Arduino en batterij verbinden
IN1	8
IN2	9
IN3	10
IN4	11
5V	5V voor logic LM298D aansluiten
ENA	5V
ENB	5V
MOTORA	MOTOR aansluiten
MOTORB	Andere motor aansluiten

7. Tips bij het programmeren

Hoe een motor trager laten draaien?

Hiervoor moeten we de motor aansturen met PWM (Pulse Width Modulation = puls breedte modulatie). D.w.z. dat we geen constante 5V spanning sturen naar de motor maar een blokspanning. De breedte van de pulsen kan je regelen met een programmeer functie. **Hoe breder de pulsen**, hoe **hoger de gemiddelde spanning**, dus hoe **sneller de motor vooruit** gaat draaien. Je kan dit makkelijk nameten met een digitale voltmeter op de motorpin van de Arduino.



De PWM pinnen bij Arduino zijn aangeduid met een **~** op de PCB**.** 

We hebben voorzien bij de bedrading van de L293D dat pin 9 en pin 11 zeker een PWM uitgang hebben.

We leggen aan deze pinnen de pulsen aan terwijl de pinnen 8 en 10 de **richting** (DIRECTION) bepalen. Als bijvoorbeeld de **pinnen 8 en 10 LOW** zijn, dan zal de robot **vooruit** rijden en telt de **breedte van de puls** om de snelheid te bepalen. Zijn deze pinnen **HIGH** dan rijdt de robot **achteruit** en telt de **ruimte tussen de pulsen** om de snelheid te bepalen. Want je moet altijd in het oog houden dat de stroom van een hoog naar laag potentiaal wil bewegen.

Met de functie **analogWrite(pinnumber, value)** kunnen we de PWM waarde ingeven. Dit is een getal tussen 0 en 255 (zie bovenstaande figuur).

In ARDUBLOCK noemt deze functie **“set analog pin”**.

Een voorbeeld van deze PWM code kan je hier vinden:

We zetten alle pinnen laag van de rechter motor (pin 10 en 11). Zo kan deze niet uit zichzelf onwillekeurig gaan draaien.

Via pin 9 gaan we met de "**set analog pin**" functie de pulsen instellen op een breedte van 150. Dit is iets meer dan 50% van de breedte als je weet dat we maximum 255 kunnen instellen (dus ongeveer de halve snelheid).

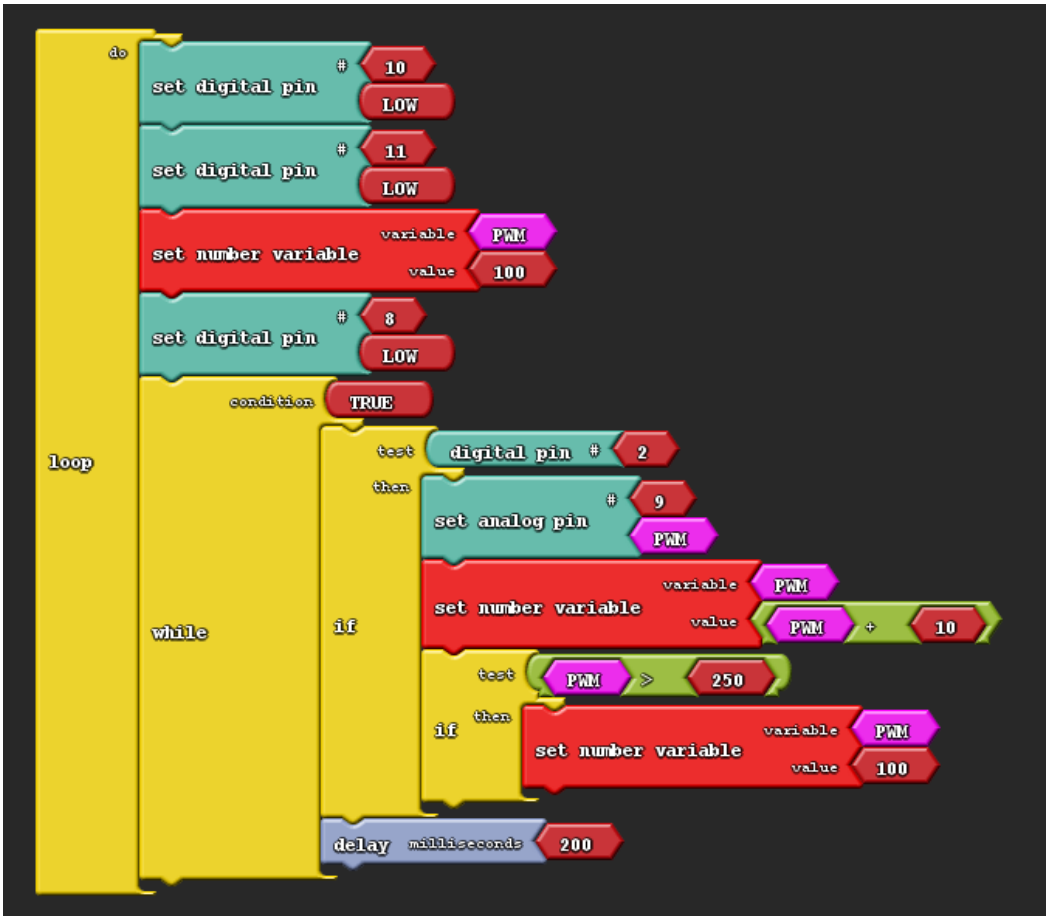
In c-code ziet het er als volgt uit:

PWM_eenvoudig

```
void setup()
{
pinMode( 10 , OUTPUT);
pinMode( 11 , OUTPUT);
pinMode( 9 , OUTPUT);
pinMode( 8 , OUTPUT);
}

void loop()
{
digitalWrite( 10 , LOW );
digitalWrite( 11 , LOW );
analogWrite(9 , 150);
digitalWrite( 8 , LOW );
}
```

Test de code eens uit. Pas de waarde "150" voor de PWM aan en test uit bij welke getal jouw motor nog kan draaien.

Voor de **gevorderden** kan je de **PWM regelbaar maken met een knop**. Telkens je op de knop drukt gaat de linker motor iets sneller vooruit draaien. Wanneer hij op het snelste is (PWM = 255), en je drukt nog eens, dan gaat de motor weer van de waarde 50 starten. We kiezen 50 omdat de motor een **minimum waarde nodig** heeft om te kunnen starten en de **wrijvingskrachten** te overwinnen van de mechanica.
Test zeker deze code eens uit en pas hem aan voor de rechter motor.



Merk op dat we hier gebruik maken van een variabele “**PWM**”. Dit is een geheugenplaats waar je een waarde kan in bewaren, ook na een optelling.
We moeten **ook vooraf de pinnen 10 en 11 laag maken**, want anders gaat de rechtermotor willekeurig gaan draaien, en dat is niet de bedoeling.

De c-code voor deze PWM oefening ziet er als volgt uit:

motoren_PWM.ino

```
int _ABVAR_1_PWM;

void setup()
{
pinMode( 2 , INPUT);
pinMode( 10 , OUTPUT);
pinMode( 11 , OUTPUT);
_ABVAR_1_PWM = 0;
pinMode( 9 , OUTPUT);
pinMode( 8 , OUTPUT);
}

void loop()
{
digitalWrite( 10 , LOW );
digitalWrite( 11 , LOW );
_ABVAR_1_PWM = 50 ;
digitalWrite( 8 , LOW );
while ( true )
{
if (digitalRead( 2))
{
analogWrite(9 , _ABVAR_1_PWM);
_ABVAR_1_PWM = ( _ABVAR_1_PWM + 10 ) ;
if (( ( _ABVAR_1_PWM ) > ( 250 ) ))
{
_ABVAR_1_PWM = 50 ;
}
}
delay( 200 );
}

}
```

De c-code is vergelijkbaar met ARDUBLOCK. Op het begin wordt een variabele van het type **integer** (int) aangemaakt. Dit is een **geheugenplaats** met de naam PWM (maar de omzetter van ARDUBLOCK voegt er "_ABVAR_1_" aan toe). Hier kan een getal < 2^15 in gestopt worden (zie eerder getoonde variabele tabellen).

De variabele wordt standaard in de setup functie op 0 gezet. Wij zetten deze achteraf op 50 in het begin van de loop. De waarde wordt steeds met 10 verhoogt (dus de puls wordt steeds een beetje breder) tot 250 is bereikt. Dan gaan we terug naar 50.

Dit levert een mooie **meting op met een oscilloscoop** op als je meet op de PWM pin.

8. Uitdagingen voor PWM motoren:

    1. Meet met de Logic Analyzer het PWM signaal na. Onderzoek het verband tussen de ingestelde waarde in de software en de breedte van de puls op de analyzer.
    2. Laat de motor achteruit draaien terwijl je de PWM regelt.
    3. Laat beide motoren met PWM vooruit en achteruit draaien.
    4. Laat de robot een vierkantje rijden van 50cm op 50cm.
    5. Maak een LED dimmer (merk op dat je hier de MAP() functie moet gebruiken als je de LED met een potmeter analoog wil aansturen). Je kan ook gewoon rechtstreeks met PWM aansturen.

11. Afstandssensor

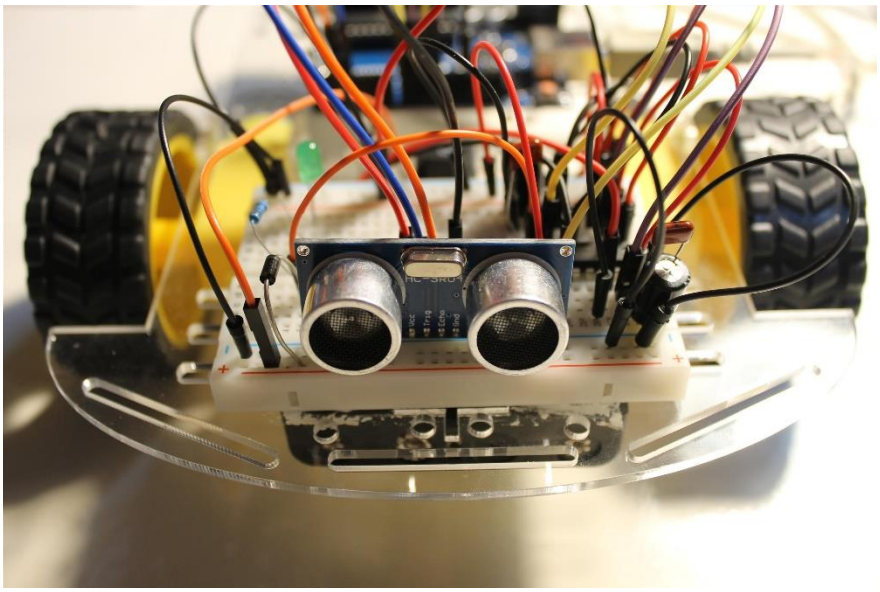
Nu we de motoren kunnen aansturen en de robot kunnen laten rijden moeten we ervoor zorgen dat deze robot **nergens tegen botst**. Hiervoor heeft men een handige afstandssensor bedacht met de naam **HC-SR04**.

Doel: laat een robot vooruit rijden en tijdig omkeren wanneer er zich een hindernis voordoet.

Benodigdheden:

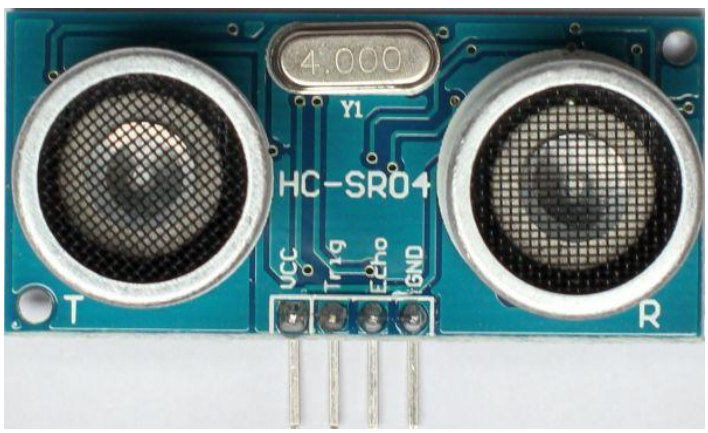
- Robotchassis met aansluitingen van de 2 DC motoren (zie vorig hoofdstuk)
- HC-SR04 afstandssensor
- 4 extra draden

1. Finale opstelling:



2. Uitleg over de sensor:

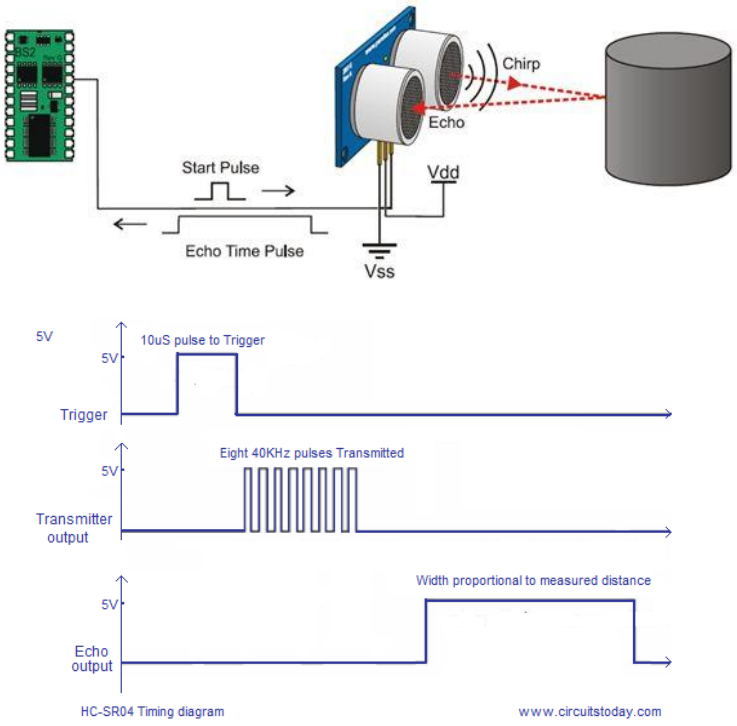
Deze sensor heeft 4 aansluitingen: VCC, Trigger, Echo en GND.



In de software gaan we eerst een **Trigger** of startpuls geven aan de sensor.

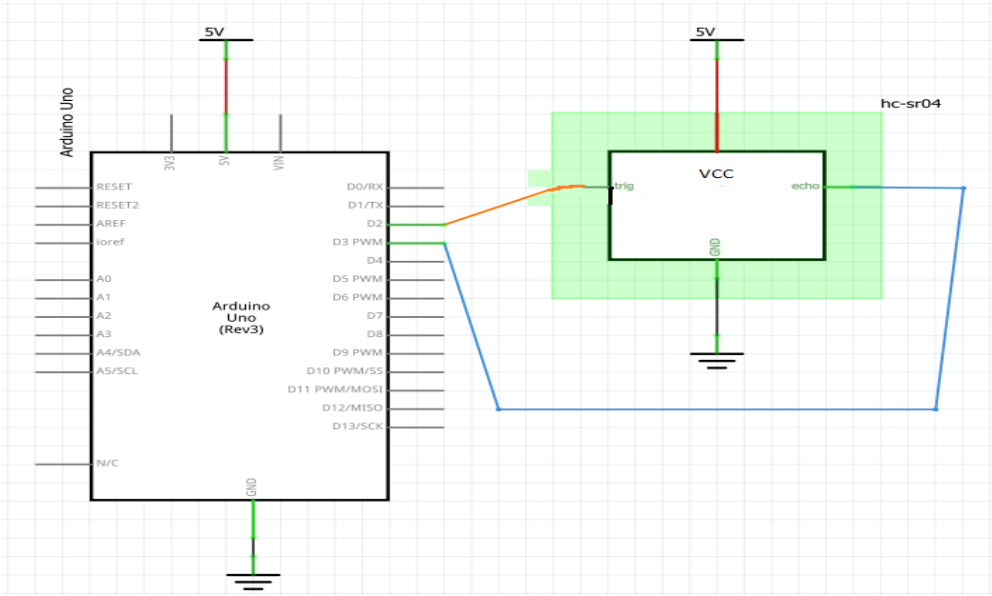
Nu stuurt de sensor geluidspulsen uit. Deze worden **weerkaatst** op een voorwerp of hindernis.

De **echo** wordt na een tijdje **ontvangen.**



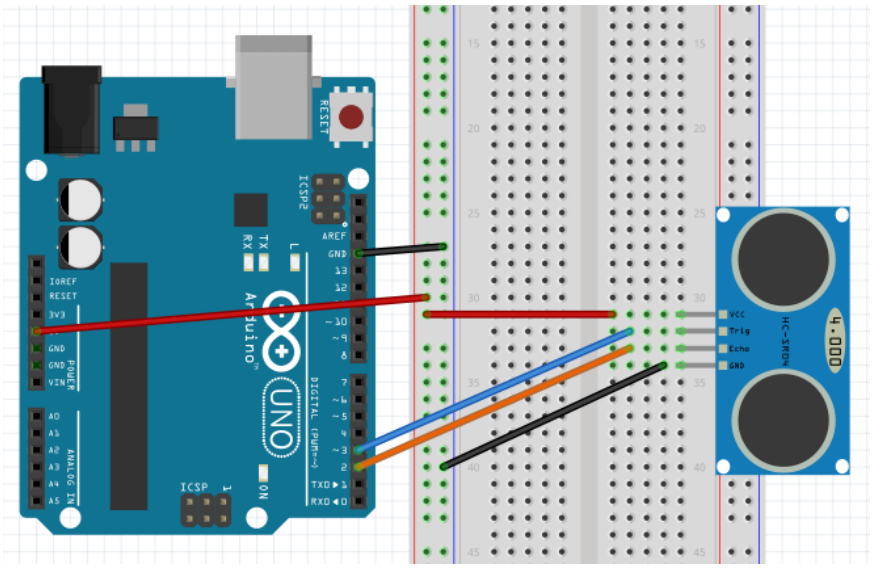
De tijd die verlopen is tussen de start van de trigger en het aankomen van de echo wordt nu terug gestuurd in de vorm van een **puls** naar de microcontroller. De **lengte van de puls** is een maat voor de gemeten afstand. Hoe verder de sensor staat van een voorwerp, hoe langer de puls wordt.

3. Het eenvoudige schema van de opstelling van de sensor: we gaan ervan uit dat de motoren en 1 schakelaar op pin A3 (nr 17) reeds aangesloten zijn, daarom staan deze niet op deze figuur aangegeven.



Trigger op digitale pin 2 en echo op digitale pin 3 aansluiten.

4. De breadboard opstelling van de sensor: we gaan ervan uit dat de motoren reeds aangestuurd zijn op het breadboard door de L293D opstelling (zie vorig hoofdstuk).
   Om de code te testen is er ook een hoog actieve schakelaar aangesloten op pin A3 (of nr 17).
   Ook deze is weggelaten op de volgende tekening.



Merk op dat in realiteit de sensor omgekeerd staat (deze kijkt naar wat er voor de robot gebeurd) t.o.v. hoe dit is aangegeven in de figuur.

5. De ARDUBLOCK code van de afstandssensor:

Wat zien we allemaal in de code?

Nadat de knop ingedrukt is, gaat de afstandssensor een meting doen. De trigger van de sensor is aangesloten op pin3 en de echo op pin 2.

Het resultaat van de meting, omgezet in een afstandswaarde en uitgedrukt in cm, wordt als waarde teruggegeven van deze functie.

De afstand wordt verstuurd naar de serial monitor en er verschijnt de tekst “afstand in cm xxx” op het scherm.

Wanneer de knop nog ingedrukt is zal er na 100ms weer een nieuwe meting gedaan worden.

6. De c-code van de afstandssensor (voor de specialisten):

afstandsmeting

```
int ardublockUltrasonicSensorCodeAutoGeneratedReturnCM(int trigPin, int echoPin)
{
  int duration;
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(20);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  duration = duration / 59;
  return duration;
}

int _ABVAR_1_afstand;

void setup()
{
Serial.begin(9600);
pinMode( 17 , INPUT);
_ABVAR_1_afstand = 0;
digitalWrite( 3 , LOW );

}
```

Je kan in de speciale afstandssensor (Ultrasonic sensor) code zien dat er eerst een puls van 20µs (0.000 020 seconden) zal worden verstuurd als trigger van de sensor. Daarna wacht men tot via de **pulseIn functie** de **afstand**, in de vorm van de **lengte van de puls in microseconden**, wordt teruggegeven. Om tijd om te zetten naar afstand gaat men deze gewoon **delen door 59.** Deze waarde is bekomen uit de volgende berekening, rekening houdend met de **snelheid van het geluid**:

distance [m] = pulse [s] * 340 [m/s] / 2

distance [cm] = pulse [us] * 0.017

distance [cm] = pulse [us] / 59 *// only 0.3% error and no floating point math needed*

Deze **afstandswaarde** wordt teruggegeven door de functie aan de microcontroller.

In de setup wordt de serial monitor klaar gezet, de juiste pinnen ingesteld en de variabele op nul gezet.

Hoe komen we nu exact aan / 59?

m = s . 340 (m/s) / 2

m = s . 170 (m / s)

m = µs . 0,00017

cm = µs . 0,017

cm = µs . 1 / 59

cm = µs / 59

```
void loop()
{
if (digitalRead( 17))
{
_ABVAR_1_afstand = ardublockUltrasonicSensorCodeAutoGeneratedReturnCM( 3 , 2 ) ;
Serial.print( "afstand in cm =" );
Serial.print( _ABVAR_1_afstand );
Serial.println("");
}
delay( 100 );
}
```

In de loop gaat de Ultrasonic sensor functie uitgevoerd worden nadat er op de knop is gedrukt (aangesloten op pin 17 = A3). De bekomen afstandswaarde wordt op de serial monitor getoond.

7. Uitdagingen voor de afstandssensor:

    1. Meet de afstand. Wanneer deze **kleiner is dan 10cm en groter dan 5 cm** dan hoor je een piepgeluidje. Dit kan je vergelijken met de sensoren in de bumper van de auto's.
    2. Voeg aan oefening 1 nog LEDs toe. Te korte afstand is een rode LED aan. Ander gaat een groene LED aan. Zo kan je bijvoorbeeld detecteren of er een auto aan een verkeerslicht staat of niet.
    3. Laat de robot vooruit rijden tot deze op 5cm van een voorwerp staat. Laat hem dan 90 graden omkeren en weer vooruit rijden.
    4. Laat enkele robots in een houten bak rijden. Laat hen elkaar detecteren met een afstandssensor. Wanneer de afstand kleiner wordt als 5cm tussen 2 robots rijdt de robot sneller vooruit en geeft de andere robot een duw.
    5. Meet met een **digitale scope of logic analyser** op het trigger en echo signaal. Onderzoek of de gemeten puls (in µs) x 0.017 gelijk is aan de afstand (in cm) getoond op de serial monitor. Toon het verband aan in jouw verslag met de nodige tekeningen en metingen. Merk op dat je deze meting ook in **Tinkercad kan doen met de simulatie scope**.

Belangrijke opmerkingen:

Zorg dat de batterijen **goed geladen** zijn (9V, 450mAh of meer) en de 4 x AA (2600mAh of meer).

Koppel de robot af van de USB spanning en **connecteer de 9V batterij** om vrij te kunnen rondrijden.

Rijd met de robot op een **plank met een rand of op de grond**. Dit om te voorkomen dat de robot van de tafel rijdt.

Zorg dat de draadjes op het breadboard **goed contact** geven en **kort geplaatst** zijn waar de spanning hoort aan te komen (voorbeeld kort bij de IC pinnen van de L293D). Eventueel de waardes eens nameten met de voltmeter. Wanneer deze te laag zijn, omwille van een te grote belasting, dan werkt de schakeling zoals voorspelt.

Zorg dat de draadjes **steeds zo kort mogelijk** gekozen worden. Elke draad vormt een antenne en kan storingen van andere draden opvangen.

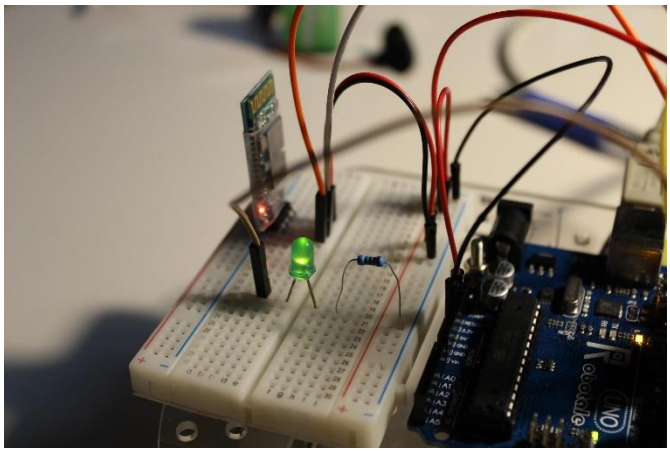
Wanneer je een **negatieve waarde** krijgt op de serial monitor ben je in feite een **te grote afstand** aan het meten: vb 36ms = 36000ns, PulsIn kan deze waarde wel meten maar is een integer variabele. In deze functie wordt dan een getal groter dan 32767 gemeten, vb 36000 – 32767 = 3233 gemeten. Dit zal dan als een overflow of negatief getal aangegeven worden op de serial monitor. Test zeker uit!

12. Bluetooth leren gebruiken (5IW enkel bij veel tijd)

Nu we geleerd hebben hoe we motoren moeten aansturen kunnen we de robot niet enkel met knoppen aansturen maar kunnen we dit ook draadloos proberen. We gaan in dit hoofdstuk de **bluetooth draadloze communicatie** technologie inzetten.

Iedereen heeft tegenwoordig wel een **GSM met bluetooth**. Dus wanneer we via een GSM applicatie de juiste commando's kunnen sturen naar de robot met zijn bluetooth ontvanger, dan kunnen we de robot zo aansturen.

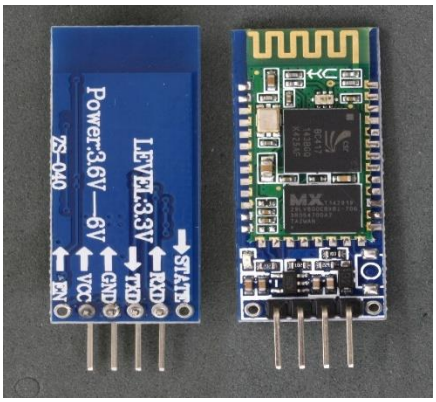
1. Finale opstelling:



2. Benodigdheden:
   - Robotchassis met aangesloten motoren via de L293D
   - Bluetooth module HC-06
   - 5 extra draden
   - Groene LED met voorschakelweerstand van 220 ohm.

3. Uitleg over de HC-06 module

Als hardware gaan we gebruik maken van de **HC-06 bluetooth module**. Merk op dat de module op de achterkant het label ZG…. of FC 114 bevat als het een Chinese module is. Voor een tot nog toe onbekende reden werken de andere modules voorlopig niet. Merk op dat er in bluetooth meerdere standaarden in omloop zijn (BLE voor Apple (v4) bij de HM10 , bluetooth v2 bij de HC06)



Merk op dat er maar 4 aansluitingen nodig zijn om deze communicatie te krijgen met een Arduino.

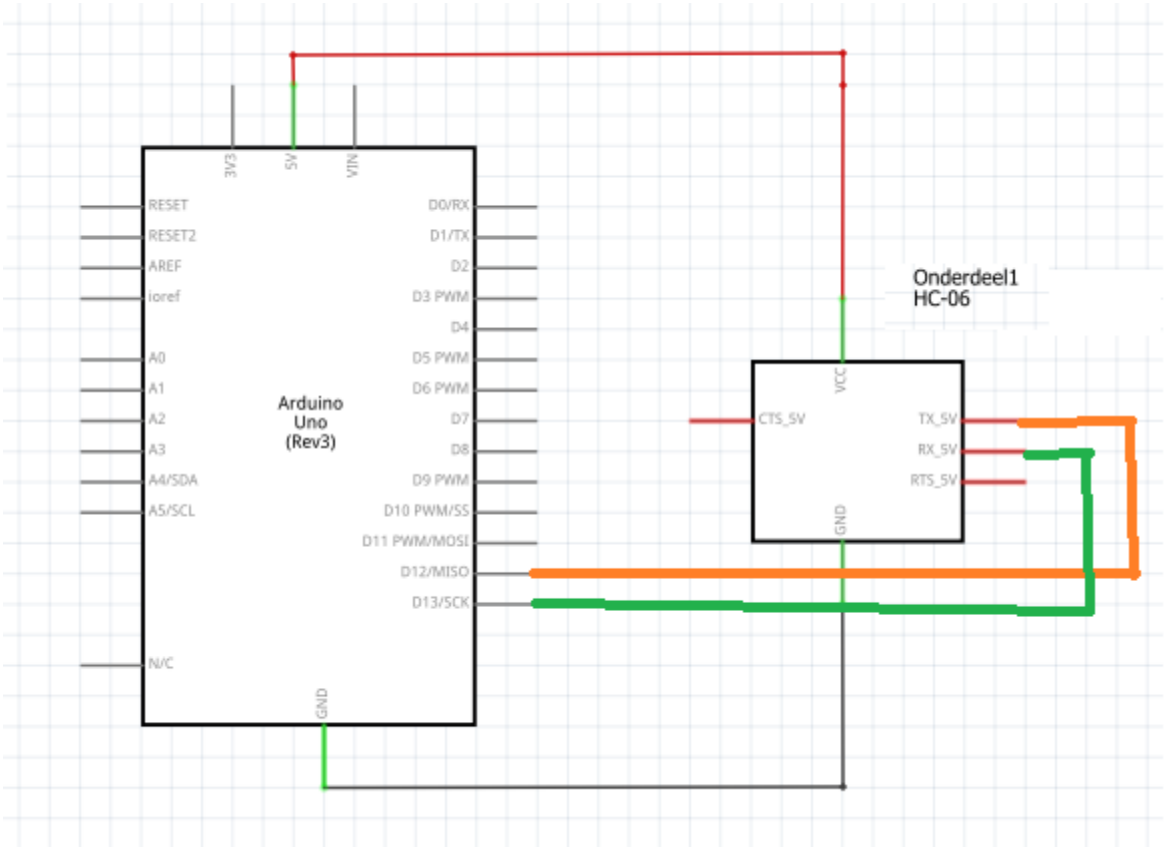
We sluiten de VCC en GND aan voor de 5V module, terwijl er ook een **seriële ingang RXD en uitgang TXD** aanwezig zijn. Hierlangs wordt de data verstuurd naar de Arduino.

De data bestaat uit **ASCII karakters** die we heel makkelijk via een bluetooth app kunnen versturen vanuit de GSM. Wanneer we nu deze karakters in onze code decoderen kunnen we hardware laten aansturen.

Wil je graag meer info over de module, dan kan je de datasheet bekijken op

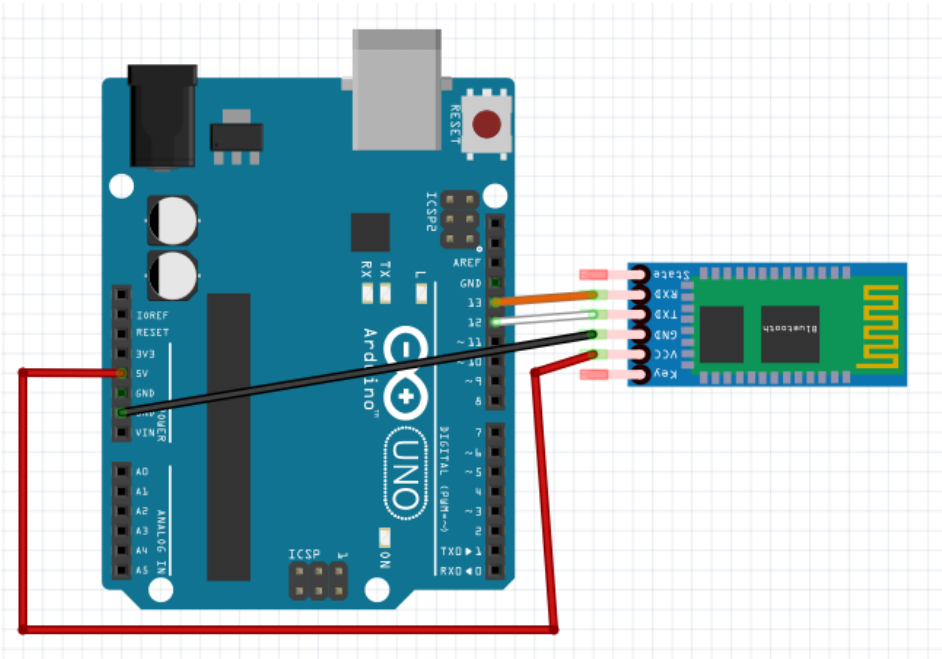
https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06

4. Het schema is eenvoudig: we gaan ervan uit dat de motoren reeds aangesloten zijn, samen met 1 LED op pin 4. De LED sturen we aan wanneer er een bluetooth signaal is binnengekomen.



Opmerking: Wanneer je wil gebruik maken van de zelfbouw bluetooth stuur applicatie **removexy.com** dan moet je de module aansluiten op de Arduino als volgt:
TX = 3, RX = 2 ipv TX = 12, RX = 13.

5. De breadboard opstelling met de HC-06: ook hier zijn, voor de duidelijkheid, de motoren en de LED niet extra voorzien.



We sluiten de massa = GND en +5V = VCC aan op de module. Daarnaast sluiten we ook pin 12 (RX UNO) aan op TXD HC06 (de zenderpin) en pin 13 (TXD UNO) aan op RXD HC06 (de ontvangerspin).

6. De ARDUBLOCK code:

   Vermits er in het ARDUBLOCK programma **geen bluetooth blokje** is voorzien kunnen we hiermee niet aan de slag. We zullen dus in c-code moeten werken.

7. De c-code aansturing voor 1 LED met bluetooth: een **groene LED** is aangesloten via een 220ohm voorschakelweerstand op pin 4.

De code ziet er dan als volgt uit:

bluetooth_demo §

```
//zorg dat eerst de vorige HC06 devices zijn verwijdert uit de GSM,
//want deze zoekt naar een andere HC06 module
//dan via de bluetooth terminal app connectie leggen
//dan de IDE downloaden in de arduino en de text verschijnt
//op de gsm

#include <SoftwareSerial.h>
SoftwareSerial BT(12, 13); //geeft de RX en TX pin aan van de UNO
// creates a "virtual" serial port/UART
// connect BT module TX to D12 -> digitale pin 12 RX UNO
// connect BT module RX to D13 -> digitale pin 13 TX UNO
// connect BT Vcc to 5V, GND to GND

int LedGL = 4;

void setup()
{
  // set digital pin to control as an output
  pinMode(LedGL, OUTPUT);
  // set the data rate for the SoftwareSerial port
  BT.begin(9600);
  // Send test message to other device (onze GSM)
  BT.println("Hello from Arduino");
}
char a; // stores incoming character from other device

void loop()
{
  if (BT.available())
  // if text arrived in from BT serial...
  {
    a=(BT.read());
    if (a=='1')
    {
      digitalWrite(LedGL, HIGH);
      BT.println("LED on");
    }
    if (a=='2')
    {
      digitalWrite(LedGL, LOW);
      BT.println("LED off");
    }
    if (a=='?')
    {
      BT.println("Send '1' to turn LED on");
      BT.println("Send '2' to turn LED off");
    }
    // you can add more "if" statements with other
  }
}
```

Test deze code eens uit. Download deze file uit de voorbeelden in de Arduino.

Belangrijk! Plaats de ino file van deze code eerst in een mapje in de root (c:\...) van jouw PC. Open dan de file en upload deze. Dit voorkomt een error.

Standaard kent onze Arduino de bluetooth code en aansturing niet. Deze moeten wij toevoegen aan het begin van de code. Vandaar de toevoeging van de **#include <SoftwareSerial.h>** header file. Deze vertelt de compiler hoe de bluetooth functie werkt.

Met het commando **SoftwareSerial BT(12,13)** stellen we de pinnen van de BT module in. RX = 12, TX = 13 bij de UNO.

We stellen in de setup() de datasnelheid van de BT module in op default 9600 baud via de functie **BT.begin(9600)**

Willen we graag tekst op het scherm van onze GSM tonen, gestuurd vanuit de Arduino, dan gebruiken we het commando **BT.println("tekst")**

We lezen in de char variabele a de characters in.

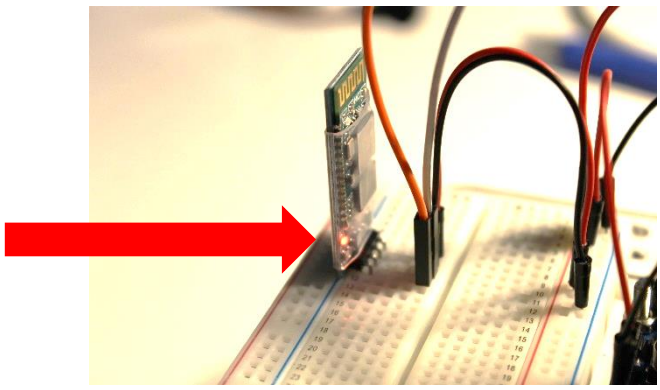
Eerst moeten we steeds testen of er een nieuw BT character is ontvangen. Dit doen we met **BT.available()**. Wanneer er iets is ontvangen dan gaan we dat character inlezen via **BT.read()**

In deze code kijken we nu na of er een 1, 2 of ? is ontvangen en sturen a.d.h.v. het character de LED aan of sturen een tekst terug naar de GSM.

Welke **app** moet ik gebruiken om de **bluetooth module te kunnen aansturen**?

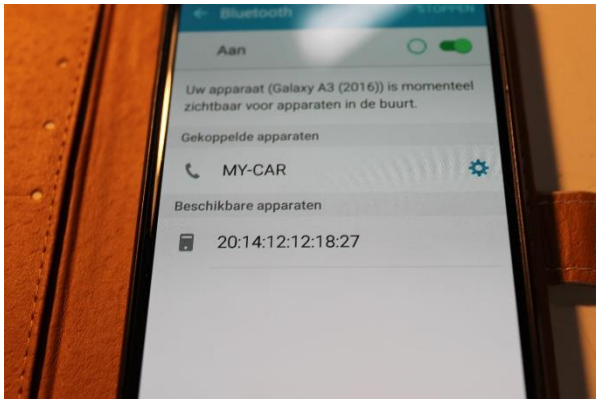
1. Voorzie de Arduino van voeding via de download kabel.

2. Nadat je de code hebt gedownload in de Arduino merk je dat de **rode LED** op de module blijft **knipperen**. Op dit moment is de +5V goed aangesloten maar hebben we **nog geen verbinding** gelegd met de module.



3. Neem jouw GSM en zet de **bluetooth verbinding** aan (uitleg is gegeven voor Andriod systemen).
4. Merk op dat er **best 1 robot tegelijk van spanning** wordt voorzien op dit moment om een bluetooth verbinding te leggen met de module. **Anders ziet de GSM meerdere modules** en weet de tester niet welk device hij nu moet koppelen. Spreek dit dus goed af in de klas. Je kan ook de unieke BT ID (of enkele letters ervan) noteren op de module. Zo weet je makkelijker welke module bij jou hoort.

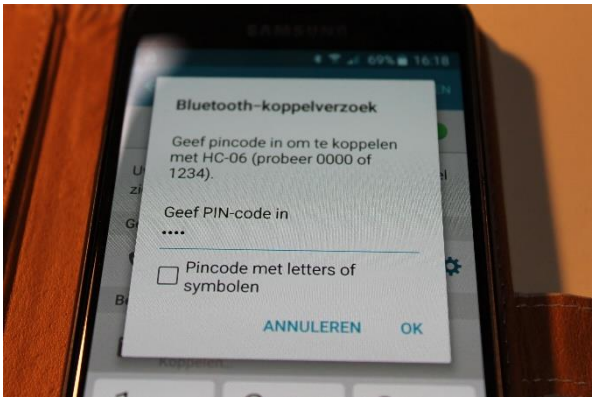
5. Als alles goed gaat merkt jouw GSM nu een **beschikbaar apparaat** op met een lange code.



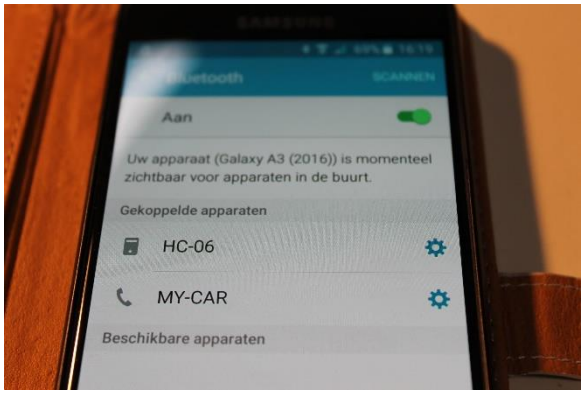
Noteer deze nummer op de BT module of op jouw blad.

Stel dat je reeds een HC-06 apparaat hebt staan van een vorige keer met een andere module, zorg er dan eerst voor dat je deze “koppeling opheft” (druk hiervoor op het blauwe tandwiel). Anders neemt de telefoon steeds de verkeerde HC-06 module, en lukt connecteren straks niet meer!

6. Probeer een verbinding te leggen met deze module. De eerste keer moet je de **pincode** ingeven. Type “**1234**” en bevestig de verbinding. De modules hebben standaard deze code.



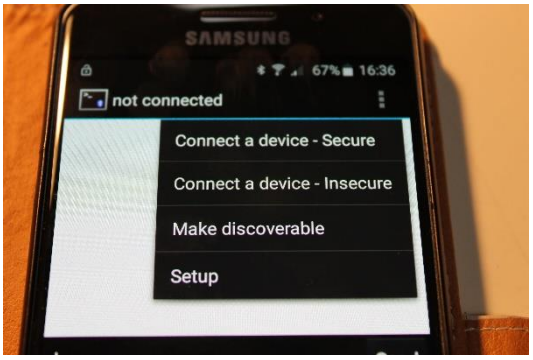
7. Als de koppeling goed gegaan is ziet nu jouw GSM de module als een **gekoppeld apparaat**. Je kan ook de module een **unieke naam geven**, vb HC-06. Druk op het tandwiel en **hernoem** het BT apparaat.



8. Nu gaan we de bluetooth app installeren op onze GSM. Ga hiervoor naar jouw **PLAY STORE** (voor Android operating systems) of de plek waar jij jouw apps download.

9. Zoek naar de app **bluetooth terminal** en installeer deze app op jouw GSM.

10. Open deze app en probeer een connectie te leggen met de HC-06. Doe dit door op de 3 puntjes de drukken en "**Connect a device – Secure**" te selecteren.

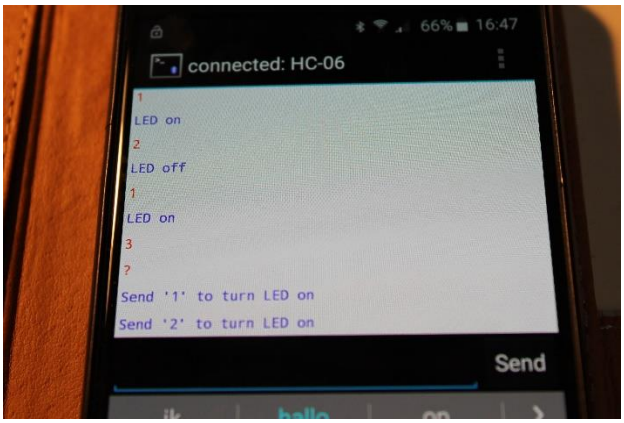


11. Selecteer nu het HC-06 device.

12. Als alles doet is gegaan moet nu "**connected: HC-06**" aangegeven zijn op de GSM en gaat de **rode LED op de HC-06 constant branden**.

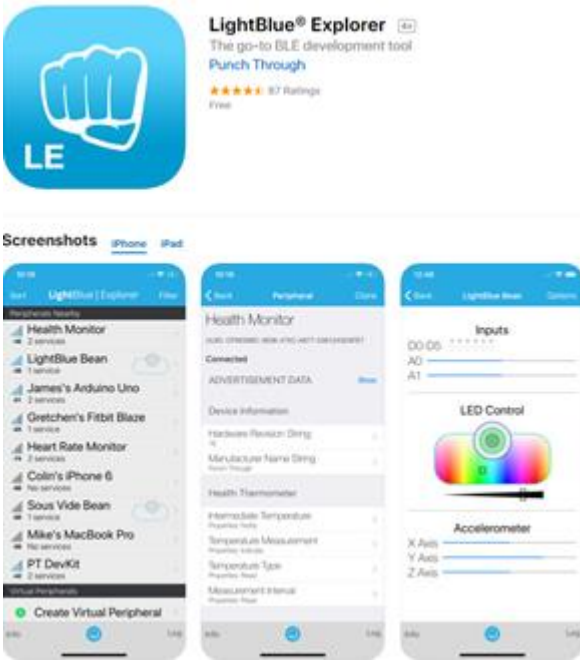
13. Nu kan je beginnen met het **sturen van characters** zoals je deze in de code hebt geprogrammeerd:

    1: LED aan op pin 4
    2: LED uit op pin 4
    ?: geef tekst op de GSM (soort help)



14. Zet na het gebruikt van de bluetooth terminal zeker jouw bluetooth verbinding terug af op je GSM. Dit spaart energie voor de GSM.

Met **de iPhone volg je best de volgende stappen**: **HM-10** toont zich als **JDY-09-V4.3 of HC05** op het scherm.



1) Download de app
2) Maak verbinding met de bluetooth module
3) Kies de laatste optie op dit scherm (onderste)
4) Je komt nu op een nieuw scherm druk rechts vanboven op hex
5) Druk nu op de "min" en het aantal veranderd naar ∞
6) Druk op null
7) Ga terug naar het vorige scherm
8) Druk op "write value" hier kan je de gewenste Commando's ingeven.

8. De c-code aansturing van de motoren via bluetooth

De aansturing van de verschillende motoren kan je terugvinden in de **voorbeeld oefeningen (zie lesgever)**. Ook hier weer de code eerst in een mapje in de root zetten (c:\...) alvorens aan te passen en te uploaden.

Kijk de werking van de code na en test deze uit op jouw robotchassis.

Merk op dat er ook 4 LEDs zijn aangesloten om de richting van de robot aan te wijzen (pin 4 tot 7). Pas hiervoor jouw hardware aan.

Op dit moment is de code van een start/stop voorzien. We kunnen met deze code enkel nog maar naar voren en naar achteren rijden.

Zorg steeds dat je na het downloaden eerst een **goede bluetooth verbinding** hebt tussen GSM en robot voordat je de code gaat uittesten.

Zorg ook voor een **goed opgeladen 9V en 4 x AA batterijen**.

Check de connecties van de motoren na voor de test. Bij slechte connecties en rare reacties van de motoren kan je best de **chip eens verplaatsen** op het breadboard en dan de draden eens opnieuw inpluggen. Wanneer breadboards meerdere keren worden gebruikt gaan de verbindingen in het board wel eens open staan en dus voor een **slechte connectie** zorgen.

9. Uitdagingen voor de HC-06 module:

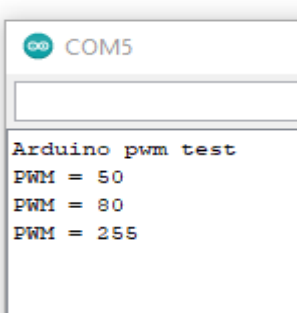
    1. Voeg characters toe en zorg dat je 4 LEDs kan aansturen op de robot.
    2. Pas de motoren code aan zodat we ook links en rechts kunnen rijden via bluetooth.
    3. Laat de robot allerlei kunstjes doen nadat je hem via bluetooth aanstuurt
    4. Zoek de apps **remotexy, blynck of Arduino bluetooth controller** uit en probeer zelf een app te bouwen dewelke jouw arduino hardware aanstuurt (enkel bestaande knopjes met nummers benoemen).
    5. Je kan ook gebruik maken van **Appinventor** maken om een **eigen app** van nul te bouwen. Werkt als scratch van MIT. http://appinventor.mit.edu/
    6. Meet het RS232 signaal op de TX, RX pin van de Arduino na tijdens het aansturen van een LED. Kijk eens wat er precies wordt gestuurd op de lijn. Kan je het getal via de decoder van de analyzer terugvinden?
    7. EXTRA: test de BT module ook eens rechtstreeks uit op de TX en RX van de UNO. Dus **geen SoftwareSerial** maar gewoon **Serial** gebruiken!

8. EXTRA: mogelijk kan je het ook wel handig vinden als je een **waarde** kan doorsturen via **BT of Serial monitor** naar de Arduino ipv tekst (vb je wil via een **slider** op jouw APP een stappenmotor, PWM van de RGB led of servo aansturen). Hiervoor moeten we dan eerst vanuit de APP de waarde versturen als tekst, dit als tekst inlezen in de Arduino en daarna de **tekst terug omzetten in een bruikbare integer variabele**. Voor deze laatste truc gebruiken we de **Serial.parseInt()** functie.

serial_mon_parseint

```
int pwm = 0;     // onthoud pwm waarde
#define PWM 9    //Rode led op pin 9

void setup()
{
  Serial.begin(9600);  // initialize serial:
  pinMode(PWM,OUTPUT);
  Serial.print("Arduino pwm test");
  Serial.print("\n");//ga naar volgende regel (ipv ln)
}

void loop()
{
  // if there's any serial available, read it:
  while (Serial.available() > 0) {

    // look for the next valid integer in the incoming serial stream:

    pwm = Serial.parseInt(); //haal serieel karakter/tekst binnen en zet om in cijferwaarde
                             //stop dan in pwm  (moet een integer getal zijn tussen 0 en 255!)

    // look for the newline. That's the end of your  sentence:
    if (Serial.read() == '\n') {

       Serial.print("PWM = ");
       Serial.println(pwm);

       analogWrite(PWM, pwm);      // verander de PWM waarde op pin 9
       delay(15);
    }
  }
}
```

Voorbeeld van een tekst die via de Serial monitor wordt verstuurd en omgezet in een variabele waarde.



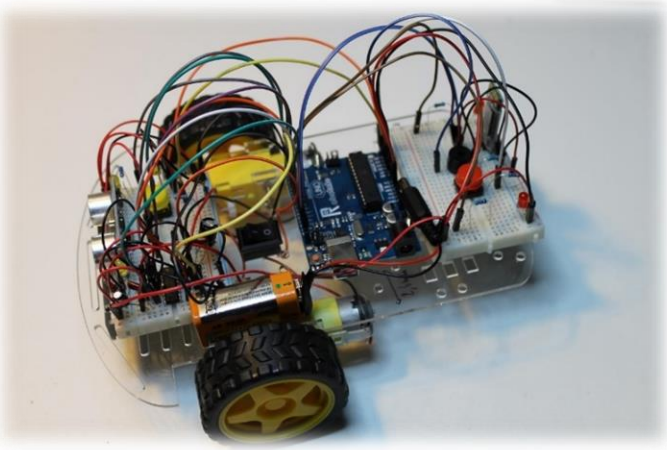
➔ **Uitdaging:** Pas nu de code aan zodat deze ook reageert op een BT signaal.

13. De Arduino robot

Nu we LEDs kunnen sturen, een zoemer laten horen, motoren aansturen en een bluetooth besturing kennen is het tijd om dit alles te integreren in 1 robot. Vergeet ook niet van de schakelaars en de afstandssensor te integreren.

Je kan bijvoorbeeld de volgende **uitdaging** eens proberen te programmeren:

1. Druk op 1 knop op de robot om de bluetooth verbinding aan/uit te zetten
2. Laat hierbij een groene LED branden
3. Telkens je drukt op de knop klinkt er een korte pieptoon
4. Voorzie 4 extra LEDs (2 geel en 2 rood) om de voor en achterlichten van de robot te voorzien
5. Stuur via bluetooth de motoren aan zonder PWM (4 richtingen)
6. Stel via een knop de PWM waarde in van de motoren en toon deze waarde op de serial monitor
7. De LEDs tonen de richting aan waarnaar er gereden wordt
8. Als de robot een voorwerp tegenkomt in tijdens het naar voren rijden dan draait hij 180 graden om en rijdt weer verder
9. Bij detectie van het voorwerp hoor je een piep toon



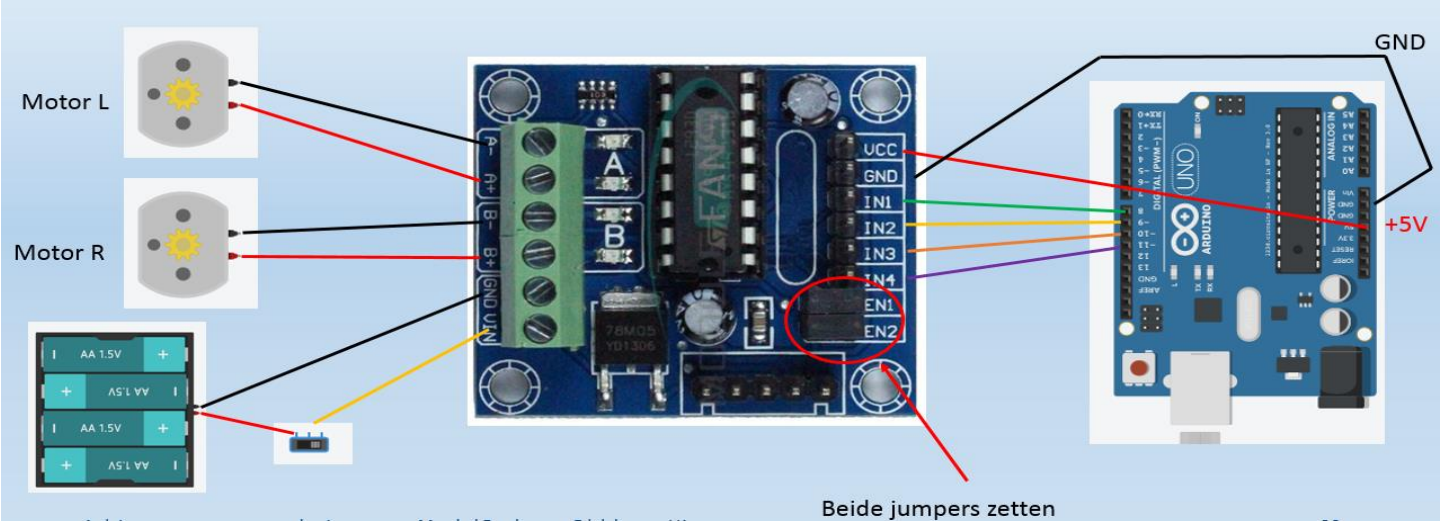
14. De lijnvolgers

1. Motoren klaar voor gebruik?

We hebben reeds geleerd hoe o.a. de motoren werden gestuurd van de robot.

Zorg, voordat je aan dit hoofdstuk begint, dat de motoren van jouw robot werken (oefeningen zie **hoofdstuk 9).**

De volgende figuur toont nog eens de aansluitingen van de motoren.



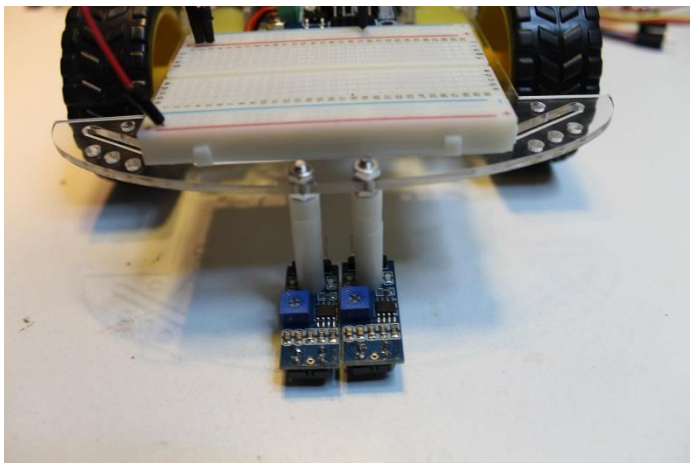
Aansluitschema met L293D motorshield

2. Lijnvolgers klaar voor gebruik?

Tijdens de **montage** hebben we 2 lijnvolgers voorzien op de robot.

Het type van lijnvolger PCB kan al eens verschillen, maar de aansluitingen blijven dezelfde.

Indien je de lijnvolgers nog niet monteerde of aansloot, kijk dan even in het **montage doc** van de Arduino robot en volg de instructies van **stap 31**.

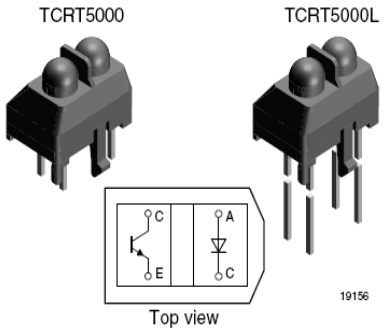


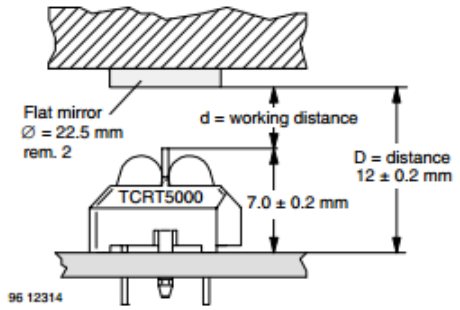
3. Meer info over de lijnvolgers:

We maken steeds gebruik van Infrarood Sensoren (IR sensors) waarbij we de afstand gaan meten tussen de sensor en de oppervlakte. Het type dat wij meestal gebruiken is de TCRT5000 van Vishay.

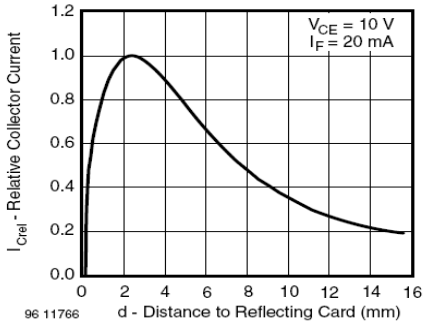
Belangrijk is dat de **afstand** gerespecteerd wordt tussen de **sensor en de oppervlakte**.

Zorg ervoor dat de **afstand "d" tussen de lijnvolger en het tafeloppervlakte maximum 15mm** is. Meet de afstand na. Eventueel moet je een moertje bijvoegen of weglaten aan de kant van de afstandsbussen.





Onder de robot zijn 2 van deze sensoren geplaatst. Wanneer zij aangezet worden en op de juiste afstand (1 tot 5mm) van het parkoer gehouden worden, dan leveren deze sensoren de volgende meting op:

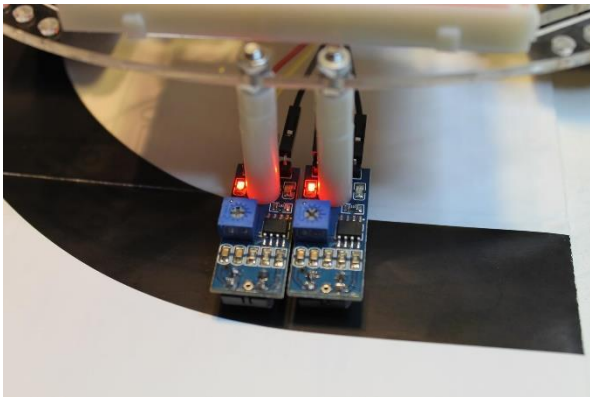


Merk op dat we bij 2.5mm de beste meting hebben!

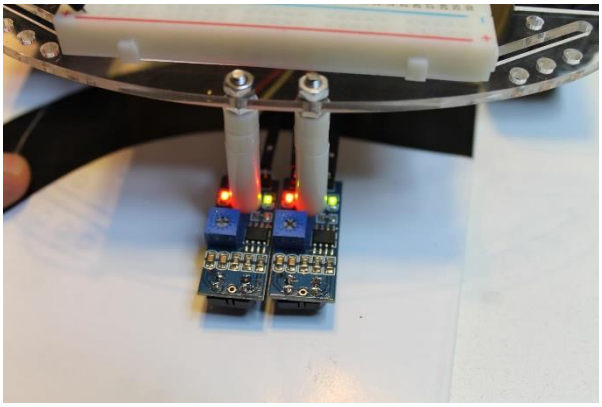
Je kan de **sensoren afstellen** via de potentiometers.

Sluit eerst 9V aan op de Arduino via de 9V batterij.

Draai dan met een kleine schroevendraaier aan beide potmeters zodanig dat je het volgende krijgt:



Zwarte lijn = groene LED OFF



Witte oppervlakte = groene LED ON

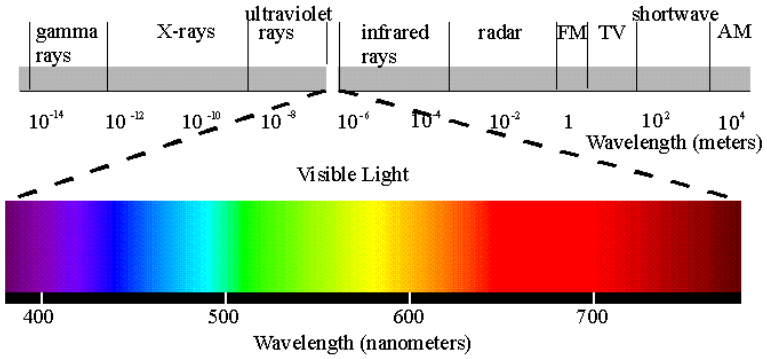
Bij een zwarte lijn onder de sensor wordt het IR signaal slecht weerkaatst.

Op een wit oppervlakte zal het IR signaal goed weerkaatst worden.

- Daylight blocking filter
- Emitter wavelength: 950 nm
- Lead (Pb)-free soldering released

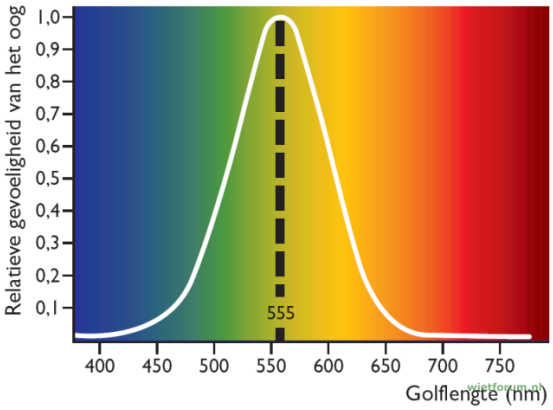
De blauwe **IR zend-LED** stuurt het IR uit terwijl de zwarte **fototransistor behuizing** het IR filtert uit het omgevingslicht.

Het lichtspectrum ziet er als volgt uit: wij gebruiken het **infrarood gebied**.



Merk op dat het oog de IR niet kan zien. Ken jij een mogelijkheid om bijvoorbeeld toch het IR signaal van een afstandsbediening te zien? …….

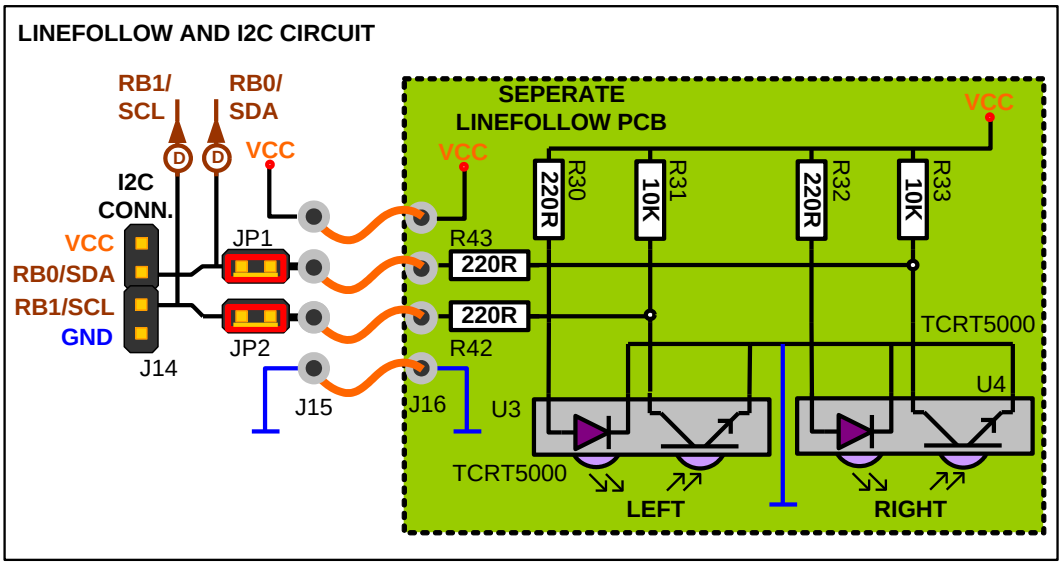
Het zichtbaar licht zit dus heel kort tegen het IR licht aan.



Ons kleurenspectrum van het zichtbare licht.

De bovenstaande tekening geeft de **gevoeligheid van het menselijk oog** weer t.o.v. het kleurenspectrum. Geel kunnen we het beste detecteren. Het rode, dat aanleunt tegen het IR zien we het minst goed.

Aansluitingen en werking lijnvolgers:



Schema van een FFrobot waar de werking van de lijnvolgers wordt aangetoond.

De werking van de schakeling is eenvoudig. De IR zend-LEDs zijn **steeds aangesloten** en sturen dus **constant een IR signaal** uit. Voor de batterij is dit natuurlijk niet de beste optie.

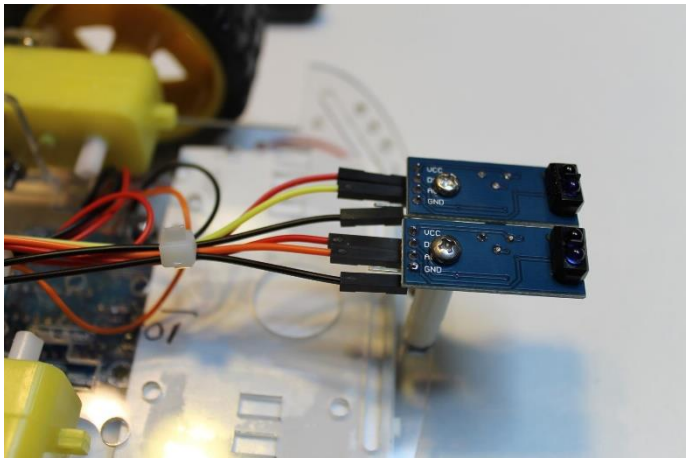
De **fototransistor** is geschakeld als een transistor als schakelaar.

Rijdt de robot over een **zwarte lijn** dan weerkaatst de IR slecht en zal de fototransistor weinig open gestuurd worden.
Dit zorgt ervoor dat de collector van de transistor hoog is.
Dit geeft een **“1”** op de digitale uitgangen van de sensoren.

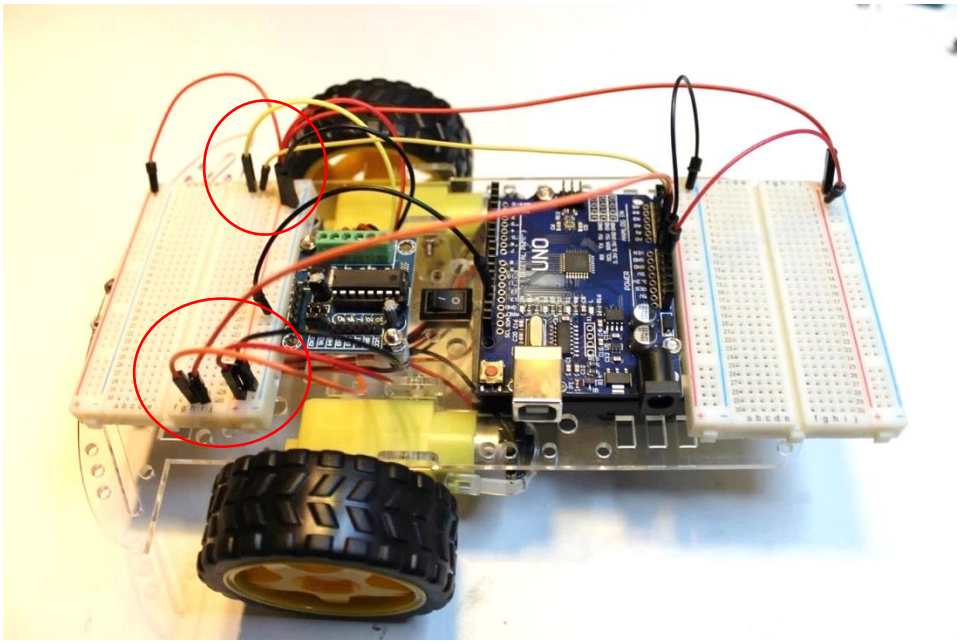
Bij de Chinese lijnvolger gaat nu de **groene LED op de lijnvolger uit.**

Wanneer er nu gereden wordt over **een wit vlak** zal het IR signaal wel goed weerkaatst worden.
De transistor schakelt helemaal open (gaat in saturatie) en de collector wordt aan massa gehangen.
We zien nu een **“0”** op de uitgang.

Bij de Chinese lijnvolger gaat nu de **groene LED op de lijnvolger aan.**



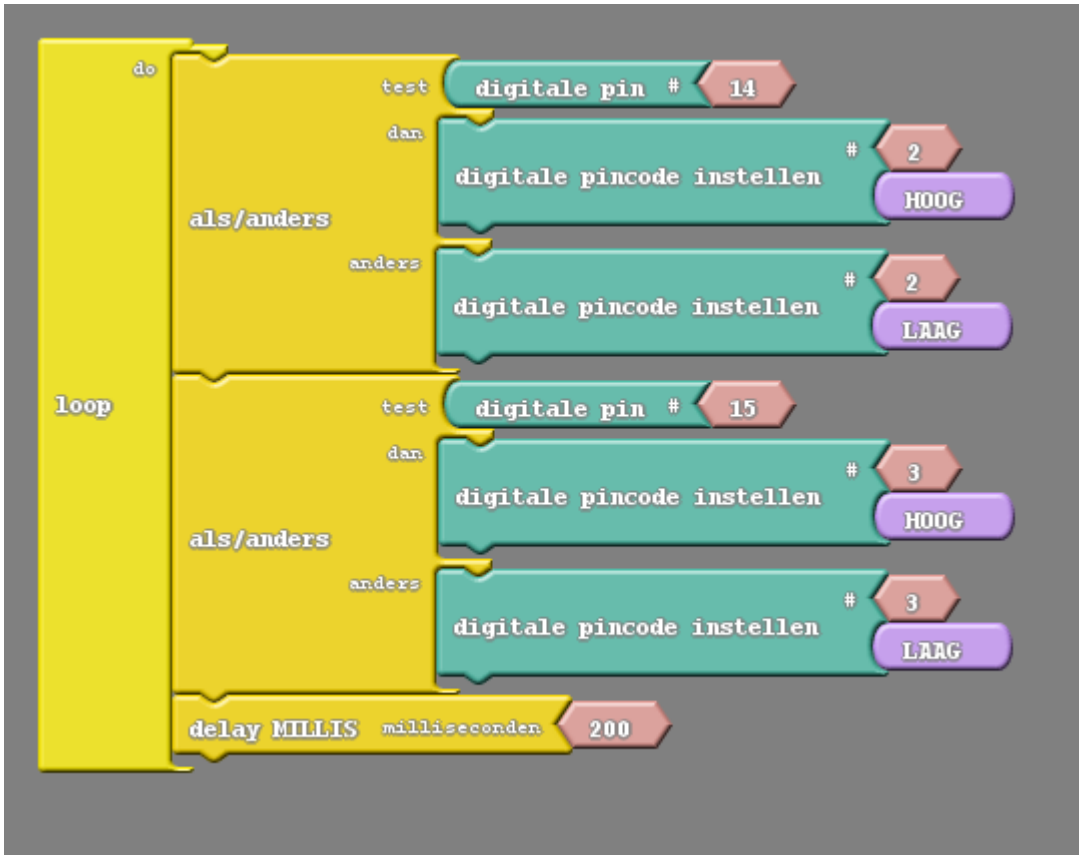
Elke sensor heeft een VCC (+5V), GND en een digitale / analoge uitgang.
Wij gebruiken voor het gemak de digitale uitgangen in onze code.



Deze foto geeft nog eens weer hoe de lijnvolgers worden doorverbonden met extra draden via het voorste breadboard naar de analoge Arduino ingangen A0 en A1.

Uitdagingen:

- Test bij jouw lijnvolger uit bij welke kleur jouw lijnvolger hoog of laag gaat. Hoe ga je dit meten?
- Indien er een LED voorzien is op de lijnvolger test je ook uit wanneer dit gebeurd.
- Is er geen LED voorzien, dan plaats je LEDs (met voorschakel weerstanden) op jouw breadboards. Verbind deze met 2 uitgangen naar keuze op jouw Arduino.
- Programmeer nu in Arduino/Ardublock de LEDs zodanig dat ze aan gaan bij zwart en uit gaan bij wit. Je kan hiervoor digitale inputs gebruiken (zie eerder in de cursus).



Lijnvolger testvoorbeeld in Ardublock

lijnvolger_digitaal

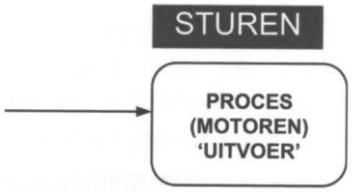
```
//test blauwe regelbare A/D chinese lijnvolgsensors -> digitale uitgang!
//regel met schroevendraaier eerst de lijnvolgers af
//groene led aan = wit

void setup()
{
  pinMode(14  , INPUT);
  pinMode( 15 , INPUT);
  pinMode( 2 , OUTPUT);
  pinMode( 3 , OUTPUT);
}

void loop()
{
  if (digitalRead(14))
  {
    digitalWrite( 2 , HIGH );//zwart = groene LED uit => 1 terug (spanning hoog bij zwakke weerkaatsing)
  }
  else
  {
    digitalWrite( 2 , LOW );//wit = groene LED aan => 0 terug (spanning laag bij sterke weerkaatsing)
  }
  if (digitalRead(15))
  {
    digitalWrite( 3, HIGH );
  }
  else
  {
    digitalWrite( 3 , LOW );
  }
  //delay( 200 );
}
```

Voorbeeld van het digitaal testen van lijnvolgers op pin 14 (A0) en pin 15(A1) in c-code.

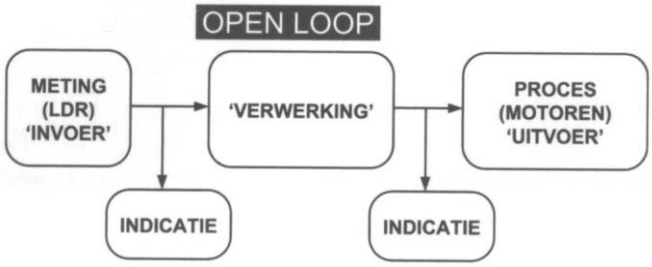
Info meet- en regelkringen:

1. Wanneer we nu moeilijker oefeningen willen maken met de microcontroller moeten wij gaan gebruik maken van de meet-en regeltechniek. Hierin onderscheiden we 3 indelingen:

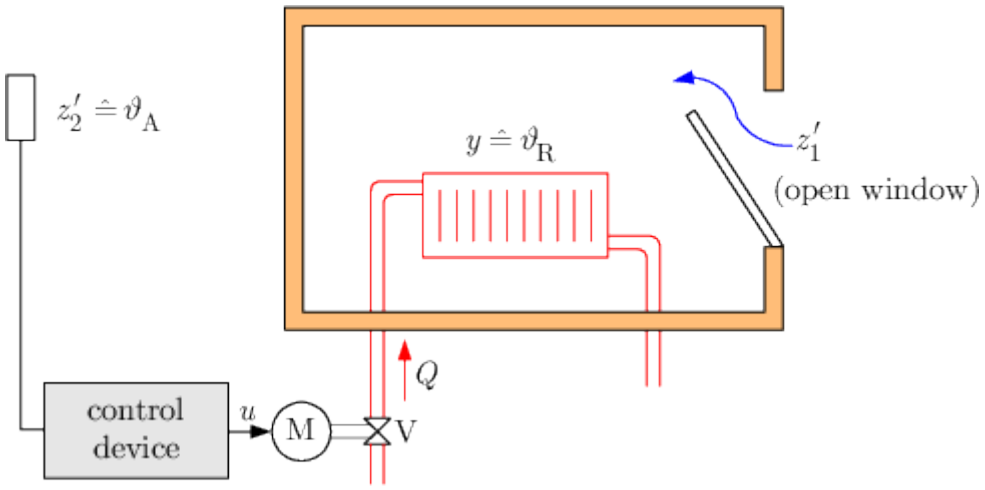
2. De eerste is gewoon **sturen**. Dat is wat we tot nu toe gedaan hebben. We hebben gewoon leds laten aan/uitgaan, tonen gegenereerd en we hebben wielen laten draaien. Vermits het draaien van de wielen bijvoorbeeld geen gevolg was van een bepaalde meting, noemen we dit eenvoudig weg een **STURING**.



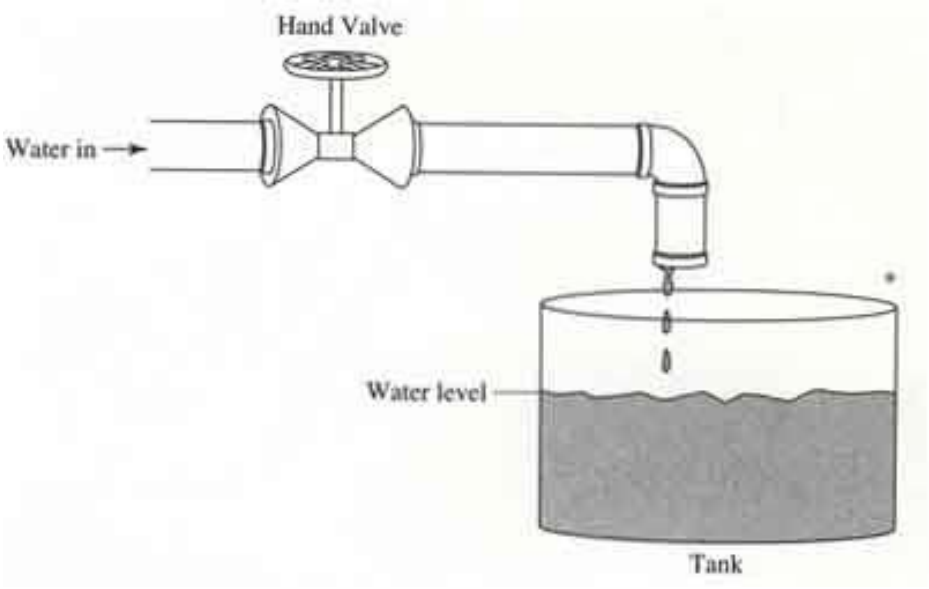
3. De tweede noemt men **open loop**. Hier is de actie wel een gevolg van de meting, maar hier heeft de actie geen gevolg terug naar de meting toe. M.a.w. de actie kan de meting niet meer beïnvloeden.
4. Een voorbeeld zou kunnen zijn dat de FFrobot moet beginnen te rijden als er met een zaklamp geschenen wordt op de LDR. Het rijden is een gevolg van de lichtmeting, maar het rijden heeft geen rechtstreeks effect op het te meten licht.



Dit is een voorbeeld van een open loop systeem:

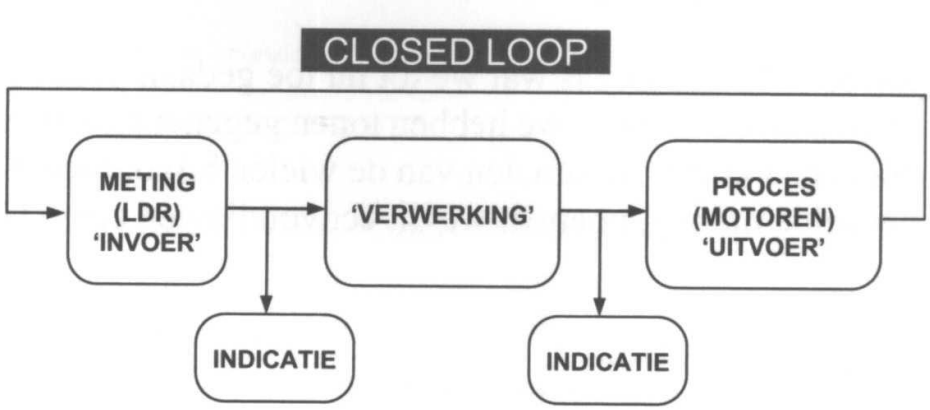


De warmte sensor meet niet binnen de ruimte die moet opgewarmd worden. De gebruiker moet dit zelf in de hand houden of er wordt a.d.h.v. de meting buiten de ruimte een actie genomen. De warmte verandering binnen de ruimte heeft geen terugslag op de sensor buiten de ruimte.

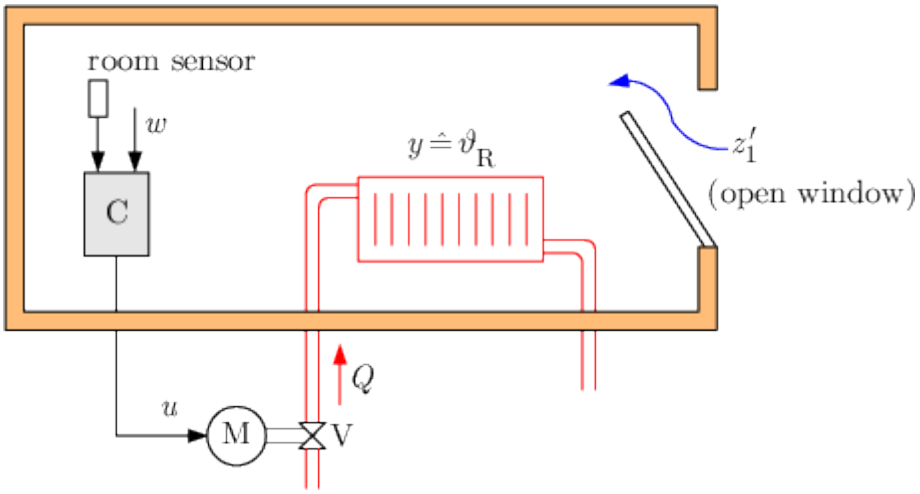


Ander voorbeeld van een open loop systeem. De operator doet hier de verwerking en opent de kraan als er water toegevoegd moet worden. Is er voldoende doet hij manueel de kraan toe en kijkt of er voldoende is. Anders vult hij bij.

5. De derde noemt men **closed loop.** De naam zegt het zelf: de lus is gesloten. De actie, die een gevolg was van de meting, heeft op haar beurt weer een rechtstreekse invloed op die meting.
6. Stel u de verwarmingsinstallatie bij uw thuis voor. Als het te koud is springt de verwarming aan. Het huis wordt nu warmer. De verwarming moet terug uitspringen voordat het te warm wordt in huis. Dit is een eeuwig durende lus, of closed loop.



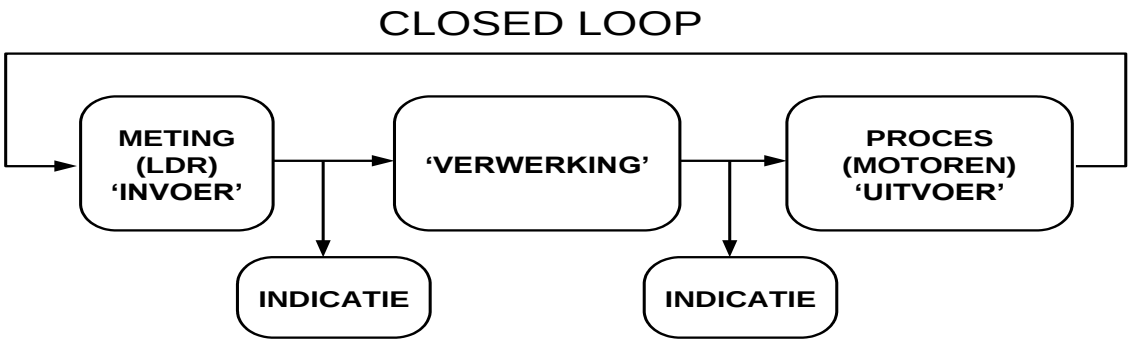
Dit is een voorbeeld van een closed loop systeem:



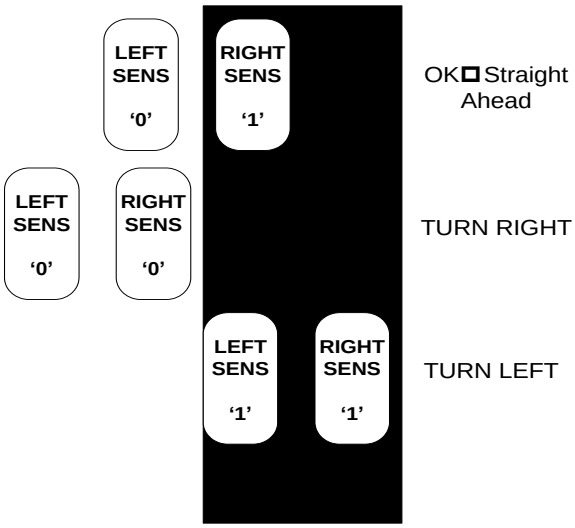
Dit keer wordt de warmte in de ruimte gemeten en wordt direct de toevoer van warm water beïnvloed. Hierdoor kan in een gesloten loop de warmte automatisch bijgeregeld worden.

4. Welke strategie gebruiken voor de lijnvolgers?

Bij de lijnvolger gaan we moeten gaan werken volgens het **closed loop systeem**. Weet jij waarom?



Dit is een **mogelijke strategie** voor de lijnvolger robot.



We gaan de **rand** van de zwarte lijn volgen.

Wanneer **beide sensoren wit** zien, dan gaat de robot **naar rechts** gestuurd moeten worden via PWM aangestuurde motoren.
Ziet de robot met **beide sensoren zwart** dan moet de robot **naar links** rijden.
Ziet hij **links wit (0) en rechts zwart** (1) dan mag hij **rechtdoor** rijden.

Je kan de robot **ook** zo programmeren dat je **de zwarte streep zelf** volgt.

Bedenkt hier zelf maar eens een strategie bij.

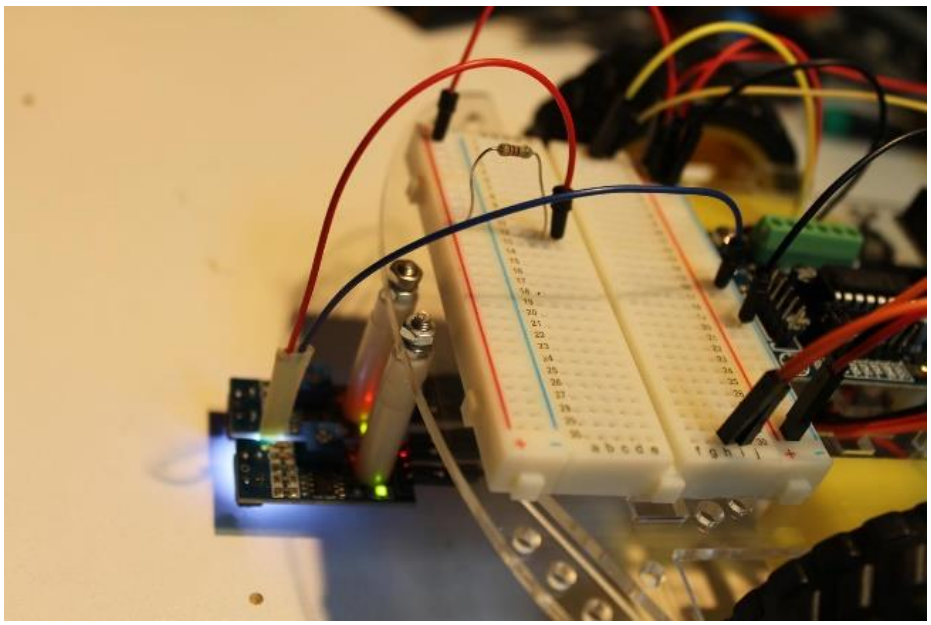
5. Extra voorzorgen om de invloed van omgevingslicht weg te werken.

Zoals eerder gezegd moeten we er mee rekening houden dat in ons omgevingslicht ook IR zitten. Ook zijn de lijnvolgers gevoelig voor weerkaatsingen en schaduwen.

Om dit probleem voor een groot stuk van de baan te helpen gaan we met een witte LED zelf een constant licht voorzien op de plek waar er gemeten wordt. Daarom is er tijdens de montage van de robot rekening gehouden met het plaatsen van een **extra witte LED** tussen de IR sensors.

Zie het **montage document stap 37** indien de LED nog niet voorzien is!

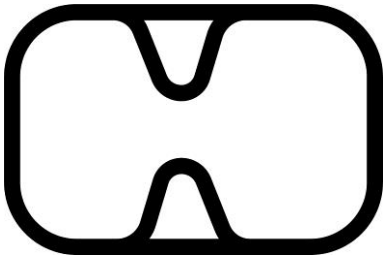
Vergeet niet van een **220 ohm weerstand in serie** te zetten met deze LED. Sluit dan de Anode (lange pootje van de LED) via de weerstand aan op de +5V en de Kathode aan de GND van de Arduino.



Witte LED om invloed van het buitenlicht te elimineren.

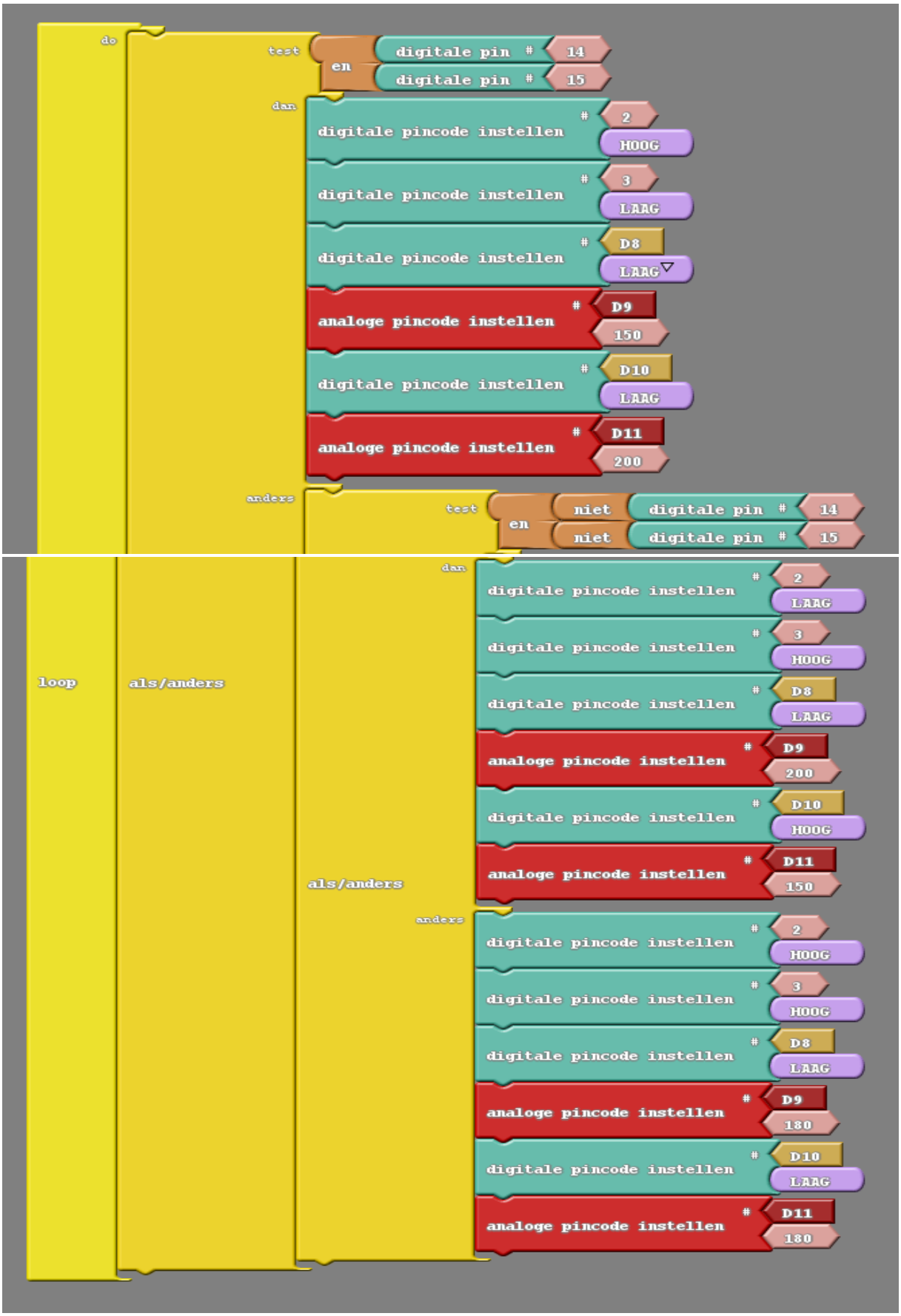
6. Uitdagingen voor de lijnvolgers

- Schrijf een programma waarbij je de robot de rand van een zwarte lijn laat volgen. Je meet de zwart wit waarden met 2 lijnvolgers. Gebruik ze digitaal.
- Maak dezelfde oefening als in uitdaging 1 maar zorg nu dat je de zwarte streep zelf volgt.
- Laat de lijnvolger rijden en voorzie een afstandssensor (theorie en code, zie eerder hoofdstuk in cursus). Wanneer de robot een hindernis op 5 cm benadert, moet hij stoppen. Bij het weghalen van de hindernis rijd hij verder.
- Je kan ook vorige oefening zo uitbreiden dat de robot omdraait bij de hindernis en ondertussen een geluidje maakt en/of LEDs laat knipperen.
- Zorg voor 2 gelijklopende parkoeren en hou een echte race! Moge de beste winnen!
- Probeer de lijnvolgers te **kalibreren** wanneer je een **analoge versie** hebt (zie info verder in cursus).



Voorbeeld van een lijnvolger parkoer

Een echt race parkoer!



Voorbeeld van de lijn volgercode in Ardublock met digitale sensors

lijnvolger_robot

```
//test blauwe lijnvolgers via digitale uitgang op zwarte lijn
//eerst afstellen , zwart = 1 (groene LED uit)

int LijnL = 14;//digitale lijnvolgers (analoog geen nut, sensors geven enkel hoog of laag (4.5V of 0V)
int LijnR = 15;


#define DIRL 8
#define PWML 9
#define DIRR 10
#define PWMR 11
#define LEDL 2 // er zitten ook leds op de sensors: wit = hoog = aan, zwart = laag = uit
#define LEDR 3

void setup()
{
pinMode( PWML , OUTPUT);//zet motoren outputs klaar
pinMode( DIRL, OUTPUT);
pinMode( PWMR , OUTPUT);
pinMode( DIRR, OUTPUT);

pinMode(LEDL, OUTPUT);//zet led's klaar
pinMode(LEDR, OUTPUT);


digitalWrite(LEDL,LOW);//2 led's uitzetten
digitalWrite(LEDR,LOW);

pinMode (LijnL, INPUT);
pinMode (LijnR, INPUT);
}
```

Voorbeeld van code van een lijnvolger in c-code (de setup) met digitale sensors

lijnvolger_robot§

```
void loop()
{
if ((digitalRead(LijnL) == 1) and (digitalRead(LijnR) == 1)) // L= zwart en R= zwart (naar links)
{
    digitalWrite(LEDL,HIGH);
    digitalWrite(LEDR,LOW);

    digitalWrite( DIRL , LOW );//vooruit
    analogWrite(PWML , 150);//links beetje PWM
    digitalWrite( DIRR , LOW );//vooruit
    analogWrite(PWMR , 200);//rechts meer PWM
}
else if ((digitalRead(LijnL) == 0) and (digitalRead(LijnR) == 0)) // L=wit en R = wit (naar rechts)
{                                          //spinnen is nog beter voor korte bochten
    digitalWrite(LEDR,HIGH);
    digitalWrite(LEDL,LOW);

    digitalWrite( DIRL , LOW );//links vooruit
    analogWrite(PWML , 200);//veel PWM
    digitalWrite( DIRR , HIGH );//bij spinnen achteruit
    analogWrite(PWMR , 150);     // beetje PWM

}
else //rechtdoor  wit zwart
  {
    digitalWrite(LEDL,HIGH);
    digitalWrite(LEDR,HIGH);

    digitalWrite( DIRL , LOW );//vooruit
    analogWrite(PWML , 180);
    digitalWrite( DIRR , LOW );//vooruit
    analogWrite(PWMR , 180);
  }
}
```

Vervolg voorbeeld van de lijnvolger met digitale sensors (volg de rand).

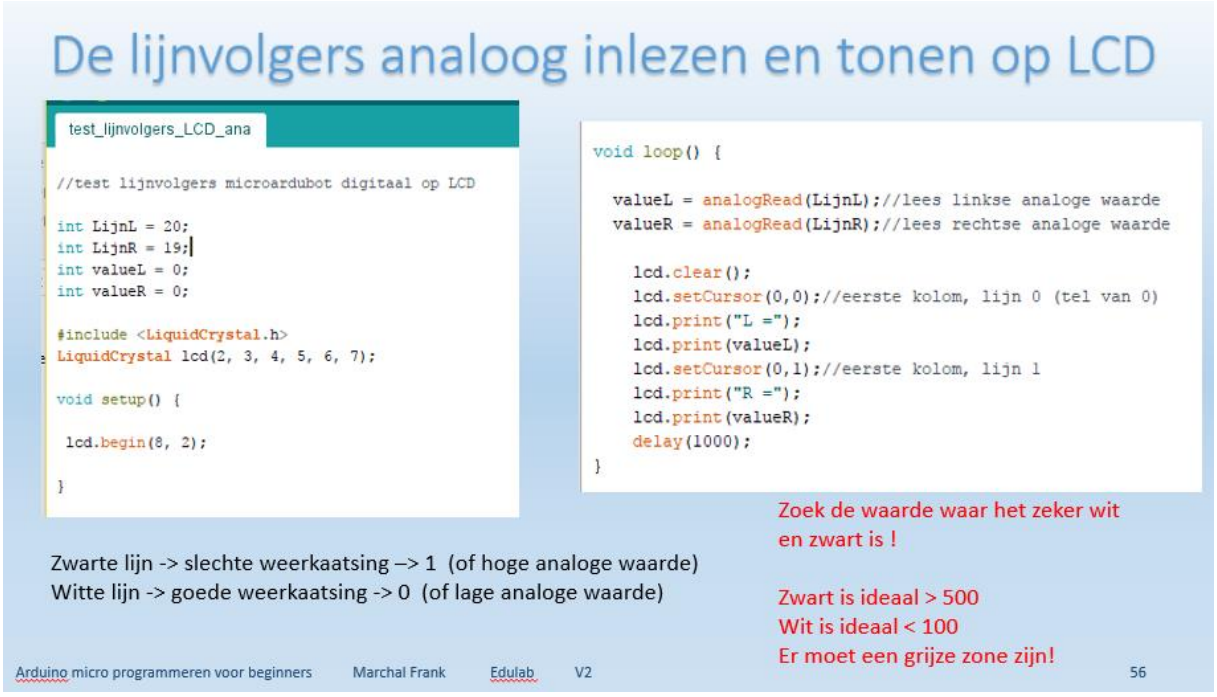
Mogelijk moeten de getallen van de PWM aangepast worden bij jou. Dit omdat elke robot niet even opgeladen batterijen aan boord heeft. Pas dus de snelheden aan voor jouw robot!

Kalibratie met analoge lijnvolgers:

Stap 1:

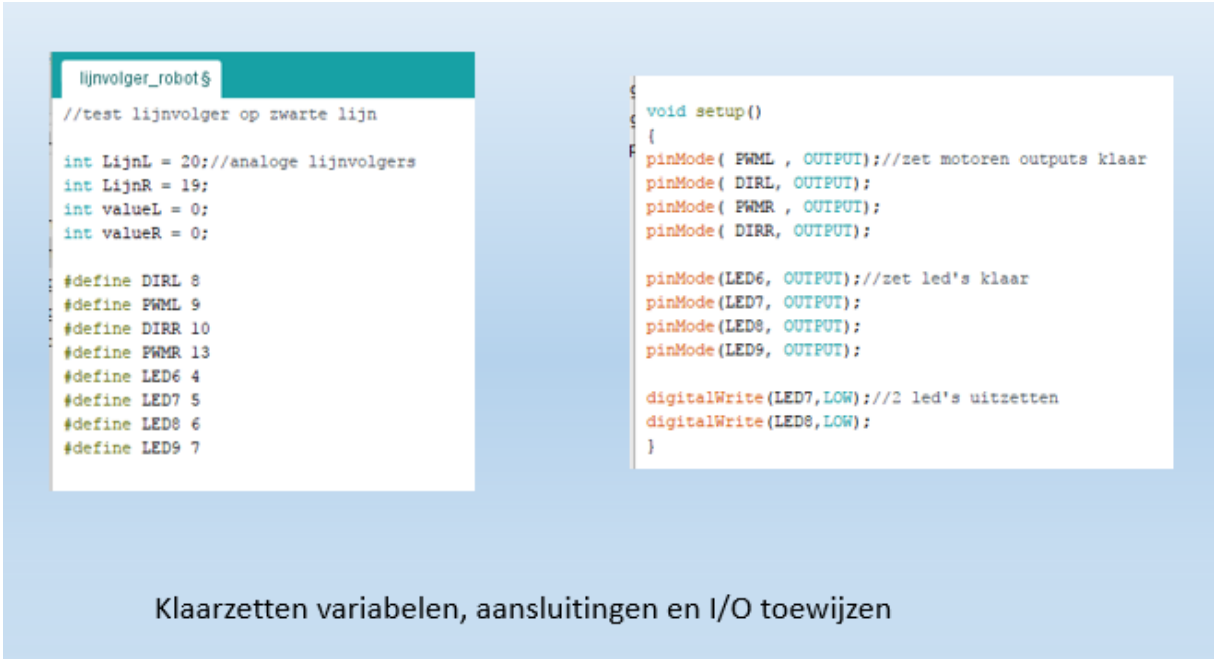
In onderstaande figuur kan je een programma terugvinden dat je kan gebruiken om analoge lijnvolgers in te lezen en de waarde te tonen op een 2x8 LCD scherm.

Test dit uit op jouw lijnvolger robot. Stop het programma in de robot en meet met de lijnvolgers op de witte en zwarte oppervlaktes. Onthoud dan jouw waardes voor zwart en wit.



Stap 2:

Geef nu de lijnvolgercode in voor analoge sensors.



```
void loop()
{

valueL = analogRead(LijnL);
valueR = analogRead(LijnR);

if ((valueL > 38) and (valueR > 38)) // L= zwart en R= zwart (naar links)
{
    digitalWrite(LED9,HIGH);
    digitalWrite(LED6,LOW);

    digitalWrite( DIRL , LOW );//vooruit
    analogWrite(PWML , 150);//links beetje PWM
    digitalWrite( DIRR , LOW );//vooruit
    analogWrite(PWMR , 250);//rechts meer PWM
}
else if ((valueL < 35) and (valueR < 35)) // L=wit en R = wit (naar rechts)
{                                         //spinnen is nog beter voor korte bochten
    digitalWrite(LED6,HIGH);
    digitalWrite(LED9,LOW);

    digitalWrite( DIRL , LOW );//links vooruit
    analogWrite(PWML , 240);//veel PWM
    digitalWrite( DIRR , HIGH );//bij spinnen achteruit
    analogWrite(PWMR , 150);     // beetje PWM

}
```

38 = zwart (best getal > 500) en 35 = wit (best getal < 100)

```
else //rechtdoor  wit  zwart
  {
    digitalWrite(LED6,HIGH);
    digitalWrite(LED9,HIGH);

    digitalWrite( DIRL , LOW );//vooruit
    analogWrite(PWML , 180);
    digitalWrite( DIRR , LOW );//vooruit
    analogWrite(PWMR , 180);

  }

}
```

De lijnvolger bestaat zelf uit 3 stukken code na het meten van de sensors:

1. Moet hij naar links?
2. Moet hij naar rechts?
3. Anders rechtdoor

Arduino micro programmeren voor beginners Marchal Frank Edulab V2 59

Vul jouw gemeten waardes in. Hopelijk liggen de zwarte en witte waardes ver genoeg van elkaar zodat er een duidelijk verschil meetbaar is. Soms moet je ook meerdere metingen op verschillend plaatsen op het parkoer doen en dan de gemiddelde waarde berekenen.

Want het licht kan op verschillende plekken anders invallen en zo de sensor anders beïnvloeden.

Succes.

De lijnvolger goed afstellen?

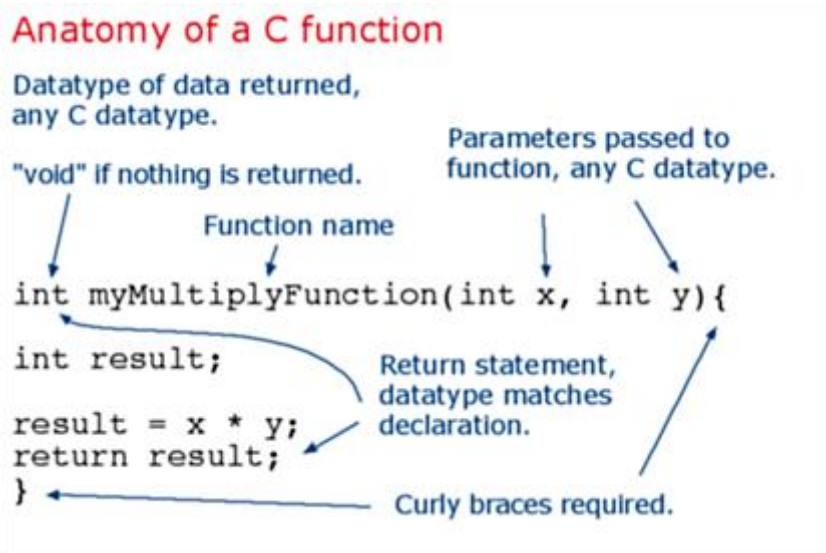
- Check of de witte LED brandt aan de lijnvolgers (jumper aanwezig?)
- Check of de lijnvolgers niet op de grond slepen (eventueel hoger hangen op max 5mm)
- Gebruik de testcode om de lijnvolgers analoog op het LCD te tonen
  -> Zorg voor een goed verschil tussen zwart en wit
- Gebruik goed geladen batterijen (9V of powerbank)
- Gebruik een test mat die “niet glanzend” is (veel moeilijker af te stellen)
- Eventueel de lijnvolgers extra afschermen van buitenlicht en UV invloeden

15. Subroutines / functies

We willen een programma **overzichtelijk en simpel** houden. En handige manier om dit te bekomen is een stel regels code in groepjes samen te nemen en deze op 1 plaats te zetten (**subroutine** of **functie** maken). Nu gaan we er voor zorgen dat **we van gelijk welke plek** in de hoofd code (loop) deze subroutine kunnen aanspreken.

Zo krijgen we ook de kans om code **steeds opnieuw** te gebruiken. Hierdoor wordt de code ook **korter.**

Tot nog toe heb je al heel wat functies gebruikt waar een subroutine achter verborgen zit, vb digitalWrite(pin,HIGH) of digitalRead(pin). We geven langs deze weg **parameters mee** en sturen deze waardes naar de subroutines digitalWrite. Een subroutine kan ook **een waarde teruggeven**, zoals bij digitalRead.




```
void setup() {
  Serial.begin(9600);

  DashedLine();
  Serial.println("| Program Menu |");
  DashedLine();
}

void loop() {
}

void DashedLine()
{
  Serial.println("----------------");
}
```

Function is called here

Function is called again

Function is created here

Om de werking van een **subroutine zonder parameters** aan te tonen gaan we het volgende voorbeeld maken:

functie_geen_parameter

```
#define BUTTON 2
#define LED 3
int state = 1;

//plaats bovenaan in de code eerst de functies
void ledknipper() {

  for (int i = 0; i < 10; i++) {
    digitalWrite(LED, HIGH);
    delay(100);
    digitalWrite(LED, LOW);
    delay(100);
  }

}

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BUTTON, INPUT_PULLUP);

}

void loop() {
  state = digitalRead(BUTTON);
  if ( state == LOW) {

    ledknipper();//functie zonder parameters
  }

}
```

In deze code is de functie ledknipper() bedacht. Wanneer deze aangeroepen wordt in de loop springt de microcontroller naar de functie bovenaan in de code. Deze wordt uitgevoerd en daarna springt de microcontroller terug naar waar hij gebleven was in de loop.

Volgend voorbeeld geeft aan hoe we parameters kunnen meegeven met een functie:

functie_parameters_meegeven

```
#define BUTTON 2
#define LED 3
int state = 1;


//plaats bovenaan in de code eerst de functies
void ledknipper(int aantal) {

  for (int i = 0; i < aantal; i++) {
    digitalWrite(LED, HIGH);
    delay(100);
    digitalWrite(LED, LOW);
    delay(100);
  }

}

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BUTTON, INPUT_PULLUP);

}

void loop() {
  state = digitalRead(BUTTON);
  if ( state == LOW) {

    ledknipper(10);//functie met parameter
  }

}
```

In de ledknipper functie wordt nu de waarde “10” meegegeven. In de functie wordt deze waarde gezien als “aantal”. De variabele aantal wordt in de functie zelf gedeclareerd als integer. Deze waarde wordt dan gebruikt in de functie in de for lus.

Nu kan je op elke plek in de loop deze functie aanroepen en de waarde veranderen.

functie_parameters_meegeven_v2 §

```
#define BUTTON 2
#define LEDR 3
#define LEDG 4
int state = 1;

//plaats bovenaan in de code eerst de functies
void ledknipper(int aantal) {

  for (int i = 0; i < aantal; i++) {
    digitalWrite(LEDR, HIGH);
    delay(100);
    digitalWrite(LEDR, LOW);
    delay(100);
  }
}

void setup() {
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(BUTTON, INPUT_PULLUP);
}

   void loop() {
     state = digitalRead(BUTTON);
     if ( state == LOW) {

       digitalWrite(LEDG,LOW);
       ledknipper(10);//functie met parameter
     }
     else{

      ledknipper(5);
      digitalWrite(LEDG,HIGH);
     }
   }
```

Op 2 plekken in de code maken we gebruik van dezelfde functie. We vullen een andere waarde in om aan te geven dat de waarde niet vast zit aan de functie. Het moet enkel een integer getal zijn (max 2^16 en positief).

functie_parameters_meegeven_teruggeven §

```
#define BUTTON 2
#define LEDR 3
#define LEDG 4
int state = 1;
int resultaat = 0;
int done;

//plaats bovenaan in de code eerst de functies
int ledknipper(int aantal) {

  for (int i = 0; i < aantal; i++) {
    digitalWrite(LEDR, HIGH);
    delay(100);
    digitalWrite(LEDR, LOW);
    delay(100);
  }
  done = done + 1;
  return done;//teruggegeven parameter
}

void setup() {
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(BUTTON, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  state = digitalRead(BUTTON);
  if ( state == LOW) {

    delay(100);//antidender
    digitalWrite(LEDG, LOW);
    resultaat = ledknipper(10);//functie met parameter
    Serial.println(resultaat);

  }
  else {

    resultaat = ledknipper(5);//ook nu wordt done uitgevoe
    //wordt enkel niet afgeprint
    digitalWrite(LEDG, HIGH);

  }
}
```

In deze code wordt een parameter meegegeven en ook een waarde “returned”. Test ook deze code uit.

Uitdagingen:

1. Maak een PWM signaal dat je stuurt naar een LED. Je kan de AAN tijd via een zelfgemaakte PWM() functie aanpassen in de code. Meet ook het resultaat met een digitale scope.
2. Maak in een wiskundige functie wisk() een berekening om een macht uit te rekenen x^y. Geef via de functie de waarde x en y mee en return het resultaat. Print dit af op een serial monitor telkens je op een laag actieve knop drukt.

Hoe kan je meerdere "tabs" maken in Arduino IDE om jouw code overzichtelijker te houden?

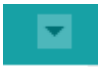
De volgende oefening toont hoe je kan springen van de ene tab naar de andere:

Maak eerst een main tab (de naam van jouw programma mag ook behouden blijven).

meer_tabs | led_sturing

```
#define BUTTON 2
#define LED 13

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BUTTON,INPUT);

  }

void loop() {

  if(digitalRead(BUTTON)== 1){

    delay(100);
    Jump_To_Led_On();//function in nieuwe tab
  }
  else{

    Jump_To_Led_Off();//function in nieuwe tab
  }
}
```

Merk op dat de 2 functies voor de LED sturing niet te vinden zijn op deze TAB.

Maak nu een nieuwe TAB door op de  te drukken in de Arduino IDE.

Geef deze TAB een nieuwe naam, overeenkomende met welke code je er wil in stoppen. We noemen het in dit voorbeeld "led_sturing". Automatisch wordt er een nieuwe *.INO met deze naam aangemaakt. Plaats hier in de code / functiesvan de LED sturing.

meer_tabs | led_sturing

```
void Jump_To_Led_On(){

  digitalWrite(LED,HIGH);
}

void Jump_To_Led_Off(){

  digitalWrite(LED,LOW);
}
```

Tijdens het compileren zal de compiler automatisch deze tabs aan elkaar linken.

Extra info over functies:

1. *Wat zijn Functies en Waarom hebben we ze nodig?*

Als eerste hebben we theoretisch geen functies nodig voor een programma, echter onze code zou dan super lang worden om maar te zwijgen over de beroerde leesbaarheid van onze code. Daarnaast besparen functies ons een hoop werk – zeker als we de functie kunnen hergebruiken.
We hebben al een aantal functies gezien en we hebben er zelfs al twee verplichte functies gedefinieerd in onze Arduino programma's: "setup()" en "loop()".
Maar wat zijn dan die functies (functions)? En waarom willen we ze gebruiken?
Een **functie** (ook wel **subroutine** genoemd) kunnen we zien als een groep instructies, welke samen een specifieke taak uitvoeren.
We maken deze functies om ons programma beter georganiseerd te houden, beter leesbaar te maken, en vaak omdat we deze "taak" elders willen hergebruiken. En dat "elders" kan zelfs in een ander programma zijn.
In bepaalde talen noemt men een functie ook wel een "subroutine", en dat beschrijft precies wat het is: een soort klein programma in ons programma. En net als bij een programma, kan een functie ook functies gedefinieerd hebben in de functie – een functie is echter ook onderhevig aan de eerder genoemde "scope". Dus een functie die in een functie gedefinieerd wordt is dus alleen maar bekend in de functie waar het gedefinieerd is – dus net zoals we zagen bij variabelen!
Ik kan me voorstellen dat dit net even te ver ging om te bevatten, het is iets waar we later vanzelf mee te maken gaan krijgen, hou het dus in gedachten.
We kunnen twee soorten functies maken: **een functie die een waarde terug geeft**, en **een functie die geen waarde terug geeft**.
Die laatste is wat men, in sommige programmeertalen, een "procedure" noemt. De taal C echter, noemt beiden gewoon een functie, of in het Engels: een "function".

Een functie, is een groep instructies met een bepaalde taak in gedachten.

Wanneer definiëren we een functie?
*Wanneer **code meerder keren herhaald** wordt in een programma,*
*Als onze code **beter leesbaar** wordt door het gebruik van functies,*
*Als we onze code **beter kunnen beheren** met functies*
*Als we onze code **willen hergebruiken**, b.v. in andere programma's.*

Laten we eens een voorbeeld doorlopen zodat we het concept beter begrijpen.
Stel we hebben een hond, en we moeten die vier keer per dag uitlaten: Om **8:00**, **12:00**, **17:00**, en om **21:00**.
De taak “hond uitlaten” kan omschreven worden met de volgende instructies:
– Jas aandoen
– Hondenriem bij de hond aandoen
– Naar buiten gaan
– 15 minuten rond lopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.
Dus voor een hele dag zou onze “code” er zo uitzien:
Als het 8:00 is doe dan:
– Jas aandoen
– Hondenriem bij de hond aandoen
– Naar buiten gaan
– 15 minuten rond lopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.
Als het 12:00 is doe dan:
– Jas aandoen
– Hondenriem bij de hond aandoen
– Naar buiten gaan
– 15 minuten rond lopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.
Als het 17:00 is doe dan:
– Jas aandoen
– Hondenriem bij de hond aandoen
– Naar buiten gaan
– 15 minuten rond lopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.
Als het 21:00 is doe dan:
– Jas aandoen
– Hondenriem bij de hond aandoen
– Naar buiten gaan
– 15 minuten rond lopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.

Zo wordt ons programma best lang he? En we zien aardig wat herhaling van dezelfde instructies …
We zouden echter de functie “HondUitlaten()” kunnen definiëren, bijvoorbeeld:

HondUitlaten():
– Jas aandoen
– Hondenriem bij de hond aandoen
– Naar buiten gaan
– 15 minuten rond lopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.

Nu wordt ons programma een stuk overzichtelijker:

Als het 8:00 is dan
HondUitlaten()
Als het 12:00 is dan
HondUitlaten()
Als het 17:00 is dan
HondUitlaten()
Als het 21:00 is dan
HondUitlaten()

Zoals je ziet is onze code nu een **stuk netter en beter te lezen**. We hebben nu de programmeertaal uitgebreid, voor ons programma, met de functie “HondUitlaten”.
In de meeste scenario’s, vooral als een functie meerdere keren gebruikt wordt in een programma, zal dit zelfs **leiden tot een kleiner programma** (na compileren) voor de Arduino of Computer. Het is efficiënter.
Maar functies hebben nog een groot voordeel. Stel we moeten de instructies voor het hond uitlaten aanpassen, bijvoorbeeld door “deur van het slot halen” ene “deur weer op slot doen” toe te voegen. Bij een functie hoeven we dat dus maar op 1 plaats te doen: in de functie definitie. In tegenstelling tot het eerste voorbeeld waar we dat op 4 plaatsen moeten veranderen. Onze code wordt dus **beter beheerbaar**.
Stel we hebben een meer generieke functie gemaakt, dan zouden we deze in een zogenaamde bibliotheek (librarys) kunnen zetten zodat we deze functie in andere programma’s ook weer kunnen gebruiken – en we dus tijd besparen wanneer we weer een ander programma gaan maken. Dit noemt met “re-use” of “hergebruik” van code. We gaan nog even niet in op het gebruik van libraries, dat is voor een stadium waarin we gevorderder zijn.

Vergeet niet:
*Een functie **is net als een klein programma** … binnen een programma*
*Een functie **kan zelf ook functies bevatten** …*
*Een functie **heeft een “scope”** – zoals we zagen bij variabelen.*

2. *Zelf Functie(s) maken*

In de programmeertaal C, zoals we die voor Arduino programmeren gebruiken, is gelukkig redelijk eenvoudig. De basis structuur ziet er zo uit:

```
datatype FunctieNaam ( FunctieParameters ) {
  // functie code
}
```

Ik hoop dat je nog iets van **DataTypes** hebt onthouden zoals we dat in Hoofdstuk 4 bespraken. Mocht dit niet zo zijn, dan kun je altijd nog even terug kijkenen ik zal een paar details hier weer vermelden.
Het datatype geeft aan wat voor antwoord we terug mogen verwachten van de functie. Echter … niet alle datatypes zijn bruikbaar hier voor, en dan moeten we vooral denken aan arrays!
De meest gebruikte datatypes voor functies zijn **boolean**, **int**, **char**, **float** en **void** (andere datatypen zoals double, long en de "unsigned" types werken ook).
Uiteraard moet onze functie ook een naam krijgen, de **FunctieNaam**, voor welke we dezelfde regels moeten volgens als namen voor variabelen en constanten:

***Functie naam**:*
***Moeten beginnen met een letter** (a, b, …, z, A, B, … , Z) **of een underscore**(_)*
*Mag **letters** bevatten*
*Mag **underscore**(s) bevatten*
*Mag **nummers** bevatten*
MAG GEEN speciale tekens, symbolen of spaties bevatten
*Is **hoofdletter gevoelig**!*

Een functie kan nul of meer **Functie Parameters** accepteren. Het aantal parameters is echter **vast** en **voorgedefinieerd** voor onze functie!

In een aantal C programmeertaal dialecten, moeten we een functie declareren (aankondigen) voor we ze kunnen gebruiken. In C voor Arduino Programmeren is dit echter niet perse noodzakelijk en daarom gaan we daar hier niet op in.

We hebben overigens ook al een aantal functies gebruikt en aangeroepen, ook al waren we ons daar misschien niet van bewust, waarbij we parameters naar de functie sturen door ze tussen ronde haakje te zetten. Een voorbeeld: Serial.print("Hallo");
Hier roepen we de functie "print()" aan van object "Serial" en geven het de parameter "Hallo" (een string). Meer over objecten later.
De code block, tussen accolades en na de functie naam en parameters, bevat de instructies voor onze functie – net zoals we dat ook bij "if" en de verschillende loops zagen ("for", "while" en "do … while …").

Een simpele (en niet zo zinvolle) functie definitie zou kunnen zijn:

```
void ZegHallo() {
  Serial.println("Hallo");
}
```

Deze functie, “ZegHallo()”, gebruikt **geen parameters**, omdat er niks tussen de ronde haakjes () staat.
Het stuurt ook geen antwoord terug omdat we het datatype “void” gebruiken – wat zoveel betekend als “niks, nada, noppes”.
De “body” van de functie (de code blok), bevat de instructies van de functie, en in dit geval voert het alleen maar de tekst “Hallo” uit.
Een voorbeeld hoe we dit kunnen gebruiken:

```
void setup() {
  // set the speed for the serial monitor:
  Serial.begin(9600);
  for(int A=1; A<=5; A++) {
    ZegHallo();
  }
}
void loop() {
  // leave empty for now
}
void ZegHallo() {
  Serial.println("Hallo");
}
```

Dit zou er een beetje vertrouwd uit moeten zien.
Aan het einde definieren we de functie “ZegHallo()”.
In de “for”-loop roepen we de functie 5 keer aan, wat zoiets als dit oplevert

```
Hallo
Hallo
Hallo
Hallo
Hallo
```

Makkelijk toch?

3. Waarden doorgeven aan een Functie

Dat laatste voorbeeld was natuurlijk bijzonder simpel. Laten we eens kijken hoe we de functie kunnen gebruiken in ons lampen voorbeeld door 5 lampen aan te zettenmet behulp van een "for"-loop.
Hiervoor maken we de functie "ZetLampAan", welke we een lamp nummer gaan geven voor de lamp die aangezet moet worden.
We weten eigenlijk wel dat de lamp nummers van het datatype "int" gaat worden (zie ook de definitie van "A" in de "for"-loop).
We weten ook dat de functie niets terug hoeft te geven.
Al dat gecombineerd:

```
void ZetLampAan(int LampNummer) {
  Serial.print("Zet de volgende lamp AAN: ");
  Serial.println(LampNummer);
}
```

De parameter "LampNummer" is dus gedefineerd als een "int". Je ziet dat we de variabele "LampNummer" in onze functie definiëren. Het is belangrijk om te weten dat deze variable de waarde aan gaat nemen van de parameter die we aan de functie doorgeven als we de functie aanroepen. Deze waarde wordt **gekopieerd** – dus als we de waarde van LampNummer veranderen, dan wordt de originele parameter (wat ook een variabele kan zijn) daardoor niet veranderd!
Om nog eens terug te komen op "scope": de variabele "LampNummer" is natuurlijk niet bekend buiten de functie "ZetLampAan".
Als we in regel 6, de functie aanroepen, geven we het de waarde van de variabele "A" mee. Deze waarde wordt de **gekopieerd** in de variabele "LampNummer".
Alles even bij elkaar gezet:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4    for(int A=1; A<=5; A++) {
5      ZetLampAan(A);
6    }
7  }
8  void loop() {
9    // leave empty for now
10 }
11 void ZetLampAan(int LampNummer) {
12   Serial.print("Zet de volgende lamp AAN: ");
13   Serial.println(LampNummer);
14 }
```

Zie je hoe de variabele "LampNummer" wordt gebruikt in de functie?
Dit is natuurlijk maar een eenvoudig voorbeeld. Wat gebeurt er nu als we meerdere waarden aan de functie door willen geven. Bijvoorbeeld door een boolean toe te voegen welke aangeeft of de lamp nu **aan** (**true**) of **uit** (**false**) gezet dient te worden?

Als we meerder parameters willen gebruiken, dan moeten we ze scheiden met komma's (,).

> *Parameters in een functie worden gescheiden gehouden door een komma, zowel bij functie definitie als bij functie aanroep.*

Zoals met alle waarden die we door willen geven bij de parameters, moeten we ook die extra parameters voorzien van een naam (LichtAan) en een data type (boolean) zodat de functie weet wat voor waarde het kan verwachten.
Uiteraard moeten we even een "if" statement in de functie hangen, zodat afhankelijk van de boolean waarde, de lamp AAN of UIT geet gaat worden. Dit laat zien dat we functies zodanig kunnen schrijven dat ze voor meerdere situaties inzet baar kunnen zijn.
Het resultaat:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4    for(int A=1; A<=5; A++) {
5      ZetLampAan(A, true);
6    }
7    for(int A=1; A<=5; A++) {
8      ZetLampAan(A, false);
9    }
10 }
11 void loop() {
12   // leave empty for now
13 }
14 void ZetLampAan(int LampNummer, boolean LampAan) {
15   if(LampAan) {
16     Serial.print("Zet de volgende lamp AAN: ");
17   }
18   else
19   {
20     Serial.print("Zet de volgende lamp UIT: ");
21   }
22   Serial.println(LampNummer);
23 }void setup() {
24   // put your setup code here, to run once:
25 }
26 void loop() {
27   // put your main code here, to run repeatedly:
28 }
```

Zoals je ziet de "for"-loop, gaat door 5 iteraties, waarbij 5 lampen AAN gezet gaan worden.
De doorgegeven waarden zijn dus ook weer apart gehouden door een komma te plaatsen!
In de volgende "for"-loop, weer 5 iteraties, waar we de lampen UIT zetten.

De output ziet er ongeveer zo uit:

```
Zet de volgende lamp AAN: 1
Zet de volgende lamp AAN: 2
Zet de volgende lamp AAN: 3
Zet de volgende lamp AAN: 4
Zet de volgende lamp AAN: 5
Zet de volgende lamp UIT: 1
Zet de volgende lamp UIT: 2
Zet de volgende lamp UIT: 3
Zet de volgende lamp UIT: 4
Zet de volgende lamp UIT: 5
```

Bedenk wel dat deze voorbeeld wel een beetje simpel zijn, maar zodra je jouw eigen programma's gaat schrijven, die wat groter zijn, dan zul je snel leren hoe handig functies kunnen zijn.

4. *Antwoord van een Functie krijgen*

We weten nu dus dat we een functie waarden kunnen meegeven door middel van parameters. Maar hoe krijgen we een antwoord terug van een functie – bijvoorbeeld voor een complexe berekening?
Herinner je de "void" in onze voorgaande voorbeelden? Dat is waar we definiëren wat voor datatype we als antwoord mogen verwachten.
In de functie zelf moeten we echter duidelijk aangeven wat de "**return**" waarde is. De term "return" is Engels voor "terug", dus de waarde die we "terug" sturen.

*Als we een functie definiëren welke een waarde terug gaat geven (dus niet "void"!) dan moeten we de instructie "**return**" gebruiken om een antwoord terug te sturen. Dat antwoord moet van **hetzelfde datatype** zijn als in de functie definitie!*

We zouden bijvoorbeeld ons geld voorbeeld, dat we eerder hebben gebruikt, als voorbeeld kunnen gebruiken. Ook dit is natuurlijk geen ingewikkelde situatie, maar het dient slechts ter illustratie hoe een functie werkt.
Stel we maken een functie "TelAmMijnGeld", welke we twee parameters geven. Ik heb deze parameters expres een andere naam gegeven om aan te geven dat de waarden van de parameters gekopieerd worden. Maar … deze namen mogen gerust dezelfde zijn, want de waarde die als parameter wordt opgegeven is een ander variabele dan de varaiabele met dezelfde naam in de functie – denk terug aan de "scope"van variabelen.
In de functie dus 2 parameters, en als antwoord het "Totaal". De "return" waarde wordt vervolgens toegewezen aan de variabele "AlMijnGeld".
Probeer de volgende code maar eens.

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4    // define our variables
5    int ZakGeld;
6    int SpaarGeld;
7    int AlMijnGeld;
8    // assign the values
9    ZakGeld = 4;
10   SpaarGeld = 12;
11   AlMijnGeld = TelAmMijnGeld(ZakGeld, SpaarGeld);
12   // print the values to the serial monitor
13   Serial.print("ZakGeld = ");
14   Serial.println(ZakGeld);
15   Serial.print("SpaarGeld = ");
16   Serial.println(SpaarGeld);
17   Serial.print("AlMijnGeld = ");
18   Serial.println(AlMijnGeld);
19 }
20 void loop() {
21   // leave empty for now
22 }
23 int TelAmMijnGeld(int OpZak, int OpDeBank) {
24   int Totaal;
25   Totaal = OpZak + OpDeBank;
26   return Totaal;
27 }
```

Een functie die terug komt met een waarde zouden we kunnen zien als een variabele welke een waarde bevat. Uiteraard kunnen we niets toewijzen aan deze zogenaamde variabele, maar we kunnen de waarde wel “uitlezen” en gebruiken waar we vaak ook een variabele of constante kunnen gebruiken. Kijk in et volgende aangepaste voorbeel maar eens naar regel 21 … We hebben als de “parameter” voor de functie “Serial.println() ” gewoon onze nieuwe functie gebruikt … en dat werkt prima! We hoeven dus de “return” (terug) waarde van een functie echt niet eerst in een variabele op te slaan – tenzij we deze waarde meerdere keren nodig hebben, want in zo’n geval willen we de berekening maar 1x doen, en iedere keer dat we de functie aanroepen, wordt de functie uitgevoerd.

Een functie welke een antwoord (return) waarde geeft kan gebruikt worden alsof het een variabele of constante is …

Elke keer als een functie wordt aangeroepen, zullen de instructies in de functie opnieuw uitgevoerd worden. Als een antwoord dus meerdere keren gebruikt wordt, overweeg dan om deze waarde in een variabele op te slaan in plaats van herhaaldelijk aanroepen van de functie.

```
void setup() {
  // set the speed for the serial monitor:
  Serial.begin(9600);
  // define our variables
  int ZakGeld;
  int SpaarGeld;
  // assign the values
  ZakGeld = 4;
  SpaarGeld = 12;
  // print the values to the serial monitor
  Serial.print("ZakGeld = ");
  Serial.println(ZakGeld);
  Serial.print("SpaarGeld = ");
  Serial.println(SpaarGeld);
  Serial.print("AlMijnGeld = ");
  Serial.println( CalculateMyMoney(ZakGeld, SpaarGeld) );
}
void loop() {
 // leave empty for now
}
int TelAmMijnGeld(int OpZak, int OpDeBank) {
 int Totaal;
 Totaal = OpZak + OpDeBank;
 return Totaal;
}
```

Ik adviseer weer om wat te gaan spelen met functies, bedenk een eigen functie, voeg wat extra parameters toe, etc.

16. Arrays

Als we drie (of nog meer) LED's aan willen sluiten en achter elkaar als een 'looplichtje' willen laten knipperen moeten we voor elke LED hetzelfde stukje code schrijven. Bijvoorbeeld…

array_oef1 §

```
int LedPin1 = 10;
int LedPin2 = 11;
int LedPin3 = 12;

void setup() {
  pinMode(LedPin1, OUTPUT);
  pinMode(LedPin2, OUTPUT);
  pinMode(LedPin3, OUTPUT);

}

void loop() {
  digitalWrite(LedPin1, HIGH);
  delay(50);
  digitalWrite(LedPin1, LOW);
  delay(1000);
  digitalWrite(LedPin2, HIGH);
  delay(50);
  digitalWrite(LedPin2, LOW);
  delay(1000);
  digitalWrite(LedPin3, HIGH);
  delay(50);
  digitalWrite(LedPin3, LOW);
  delay(1000);
}
```

Dat is onhandig. Wat we eigenlijk willen programmeren is iets als "doe eerst iets met de LED op pin 10, daarna hetzelfde met de LED pin 11 en daarna hetzelfde met de LED pin 12." We willen dus drie keer achter elkaar hetzelfde doen maar het enige verschil is het pinnummer.

We hebben twee dingen nodig.

I. Een 'rijtje' met de ledpin nummers. Een rijtje heet in programmeertaal **een 'array'.** Met de volgende opdracht vertellen we de Arduino dat er in de variabele ledpins drie ledpinnummers staan namelijk nr. 10, 11 en 12

```
int ledpins[3]={10,11,12};
```

Als we nu in ons programma ergens digitalWrite(ledpin[2], HIGH); schrijven dan zou de led op pin 11 aan moeten zetten. Dat klopt bijna maar niet helemaal.
In Arduino wordt de **eerste in een array met ledpin[0]** aangegeven, de tweede met ledpin[1] enzovoort.
Als we de drie led's op pin 10,11 en 12 aan willen zetten moeten we dus schrijven digitalWrite(ledpin[0], HIGH); digitalWrite(ledpin[1], HIGH); digitalWrite(ledpin[2], HIGH);

Maar nu zijn we toch nog steeds bezig om steeds hetzelfde stukje code voor elke Led apart te schrijven? Klopt!

II. Daarom hebben we **een 'tellertje'** nodig dat van 0 tot en met 2 doortelt. Tellertjes komen heel veel voor in programma's. Het gaat zo…

```
for (i=0;  i<3;  i++)  {
 pinMode(ledpins[i],OUTPUT);
}
```

In gewone taal staat hier:

a. Geef de variabele i eerst de waarde 0.

b. Doe alles wat er tussen de accolades staat.

c. Verhoog daarna de teller met 1. Die opdracht kan in Arduino heel kort. i++

d. Controleer of de teller nog geen 3 is.

Nee? Blijf dan de stappen b t/m d herhalen.

Ja? Dan zijn we klaar!

Hiermee kan ons programma om drie LED's na elkaar te laten knipperen een stuk korter.

array_oef2

```
/* Meerdere led's en het gebruik van een array */
int ledpins[3] = {10, 11, 12};//van 0 tot 2 tellen

void setup() {
  for (int i = 0;  i < 3;  i++) {
    pinMode(ledpins[i], OUTPUT);
  }
}

void loop() {
  for (int i = 0;  i < 3;  i++) {

    digitalWrite(ledpins[i], HIGH);
    delay(50);
    digitalWrite(ledpins[i], LOW);
    delay(100);
  }
}
```

Nu worden de 3 leds veel overzichtelijker bij de pinMode ingesteld.

We roepen nu via de index (plaats positie in de array) steeds de locatie op waar de pin nummer zich bevindt. Steeds onthouden dat we van "0" beginnen te rekenen.

Meer info over arrays vind je hier:

1. *Wat zijn Arrays?*

We weten nu dat een “array” (Array Data Type, Array Data Structuur) gezien kan worden als een **verzameling van elementen van hetzelfde data type,** welke we ieder met een **index nummer** kunnen benaderen. Een array, in z’n eenvoudigste vorm, kan gezien worden als een lijst.
Uiteraard is het gebruik van een array niet beperkt tot tekst (zie string). Een array (lijst) kan gebruikt worden met **elk data type** die we kennen in de C programmeertaal zoals we deze gebruiken voor het Arduino Programmeren. We kunnen bijvoorbeeld een lijst met nummers maken, of boolean (b.v. lampen aan of uit), en zelfs arrays van objecten is mogelijk, zoals van bijvoorbeeld het “String”-object.

> *Een **Array** kan gezien worden als een **lijst van elementen van hetzelfde data type**,waarbij we **elementen individueel kunnen benaderen met een index nummer**.*

We hebben het voorbeeld van een string: `char Naam[5] = "Hans";`
Wat resulteerde in een array (lijst) van 5 elementen, waarbij elk element een “char” is:

positie (index)	Geheugen locatie inhoud
0	H
1	a
2	n
3	s
4	NULL

De variabele “Naam” wijst naar de **geheugen locatie** van het eerste element, dus de geheugen locatie die de letter “H” van de string bevat. De **index** van deze eerste locatie is “0” (nul). Immers: arrays beginnen met element tellen vanaf nul.

Laten we eens een andere array maken, bijvoorbeeld met booleans …
Stel we hebben 5 lampen, en deze kunnen AAN of UIT zijn, wat dus prima is voor het gebruik van een boolean waarbij we AAN als “true” zien, en UIT als “false” zien. We kunnen hiervoor een variabele maken, b.v. “LampenAan” en maken deze een array van booleans (Engels: Array of Boolean):

```
bool LampenAan[5];
```

Zoals met gewone variabelen kunnen we de variabele meteen waarde(n) toewijzen, echter voor arrays gaat dat net een beetje anders dan voor gewone variabelen. In tegenstelling tot gewone variabelen, waar we slechts een enkele waarde toewijzen, willen we bij een array meteen meerdere waarden toewzijen. De programmeertaal C heeft daarvoor een aparte notatie die gebruik maakt van accolades. Doet je vast een beetje denken aan de code blokken die we

eerder gezien hebben. Misschien is het nog niet zo gek om het als code blok te zien: we wijzen namelijk een heel blok in 1x toe.
Stel we definiëren onze variabele “LampenAan”, voor 5 lampen, en zetten alle waarden initieel op UIT (=**false**):

```
bool LampenAan[5] = { false, false, false, false, false };
```

We geven dus een lijst met waarden door, ingesloten door accolades, en de waarden gescheiden door een komma.
Net zoals we dat met een string konden, kunnen we voor onze nieuwe variabele ook elk element met een index nummer benadere (lezen en schrijven) zoals we laten zien in dit voorbeeld:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4    bool LampenAan[5] = { false, false, false, false, false };
5    // schakel lampen
6    LampenAan[2] = true;   // Zet de 3de lamp AAN
7    LampenAan[4] = false;  // Zet de 5de lamp UIT
8    // kijk of de 2de lamp aan staat
9    if(LampenAan[1]==true) {
10     Serial.println("Tweede lamp staat AAN!");
11   }
12   // Kijk of de eerste lamp aan staat
13   if(LampenAan[0]) {
14     Serial.println("Eerste lamp staat AAN!");
15   }
16   // Als de 4de lamp niet aanstaat, zet dan ook de 3de lamp aan
17   if(!LampenAan[3]) {
18     Serial.println("4de lamp was niet aan, 3de lamp werd dus aangezet!");
19     LampenAan[2] = true;
20   }
21 }
22 void loop() {
23   // leave empty for now
24 }
```

Dit voorbeeld is natuurlijk helemaal niet spannend, en het doet ook eigenlijk niks leuks, maar het geeft een voorbeeld hoe we met een array kunnen werken.
Merk wel op, en we hebben dit al eerder gezien, dat we met een array van booleans werken. Dat wil dus zeggen dat we daar handig gebruik van kunnen maken in het “if” statement, welke daardoor korter wordt.
De langere methode, voor het “if” statement (zie regels 12 en 17), is als we LampenAan[1]==true hadden gezet in de conditie van het “if” statement. Maar omdat de waarde al een boolean is, hebben we al een boolean waarde voor onze conditie. Maar omdat LampenAan[1] vergelijken met een boolean weer een boolean oplevert (en eigenlijk niets bijdraagt) kunnen we daar net zo goed alleen maar de boolean waarde wegzetten als conditie.

Je ziet dat we in regel 22 ook de logische operator “not” hebben gebruikt – en de “not” operator draait een boolean gewoon om, dus true wordt false en false wordt true. In het voorbeeld willen we zien op “LampenAan[3]” NIET aan staat.
Omdat we de variabele naam en het gebruik van de waarden (true=AAN en false=UIT) een beetje logisch hebben uitgekozen, kun je dit ook gewoon zo lezen.
Dus “als **NIET** LampenAan[3] doe dan …“.
Als de lamp dus aanstaat, dan hebben we een “true” (waar), en de “not” operator maakt er “false” (onwaar) van. Dus de “if” conditie faalt.
Maar als de lamp uit staat dan geeft LampenAan[3] een “false”, en door de “not” operator wordt dit “true”, dus de “if” conditie slaagt en de code wordt uitgevoerd.

2. *Waarom beginnen Arrays met tellen bij nul?*

Ehm, je zei de 4de lamp, maar in de code type je een 3, is dat niet een foutje?
Kun je nog herinneren dat we bij een string (array van karakters) zeiden dat we **beginnen te tellen met nul**?
Nou, dat geldt dus voor alle array … bij een **array beginnen we altijd met nul voor de index nummers**.

> *Arrays zijn **ZERO INDEXED (nul geïndexeerd)**, wat wil zeggen dat we **altijd beginnen te tellen met nul** en dus het **eerste index nummer is ook nul**!*

Laten we even kijken hoe dat in z’n werkt gaat.
Dus de variabele wijst naar de geheugen locatie van het eerste element van de array. Die geheugen locatie bevat dus de waarde van het eerste element. Dit noemen we een zogenaamde **“pointer”** en deze wijst dus naar **waar de “lijst” (array) begint**.
Een pointer (Engels voor: **Aanwijzer**) wijst dus naar een geheugen locatie. Als we de waarde van het eerste element willen lezen, dan kijken we naar de geheugen locatie waar de pointer naar toe wijst plus nul.
Voor het volgende element, kijken we naar het geheugen adres + 1. De daarop volgende staat in geheugen adres +2, etc.
Dus het index nummer wordt bij het geheugen adres opgeteld om het geheugen adres te krijgen van het gewenste element in de array.
Dat was de versimpelde uitleg, want in werkelijkheid is het een beetje complexer. Misschien kun je nog herinneren dat niet ieder data type even veel geheugen in beslag neemt. Zo gebruiken een byte en een char maar 1 byte, terwijl een int(eger) 2 bytes gebruikt.
Dat wil dus zeggen dat simpel weg “1” gebruiken voor iedere volgende geheugen locatie niet altijd even lekker zal werken. Het is handig om te weten dat een geheugen locatie altijd een enkele byte bevat. Dus als we een datatype gebruiken die meer dan 1 byte nodig heeft, dan zullen we dit probleem anders moeten oplossen.

Laten we als voorbeeld een array maken van int (integer, of te wel gehele nummers):

```
int Nummers[5];
```

We hebben dus een array van 5 int elementen, en **elke** *int* **heeft <u>2 bytes</u> aan geheugen nodig**.
Het eerste element is een eitje, want die wijst al meteen naar de juiste locatie.
Maar als we bedenken dat de geheugen locatie met de volgende reken truc wordt bepaalt, dan wordt het e.e.a. misschien duidelijker:

geheugen locatie + (**index nummer** * 2 bytes)

Het **index nummer** voor het eerste element is **nul**, dus de berekening wordt:

geheugen locatie + (0 * 2 bytes)

Vermenigvuldigen met nul levert altijd nul, dus de uitkomst wordt:

geheugen locatie + 0

Dus we krijgen voor het eerste element netjes de correcte geheugen locatie.
Als we naar het tweede element (index = 1) van onze int array gaan kijken, dan wordt de berekening:

geheugen locatie + (index nummer * 2 bytes)
geheugen locatie + (1 * 2 bytes)
geheugen locatie + 2

Dus de geheugen locatie voor het twee element, met (index = 1) levert met deze truc dus ook weer de juiste locatie op.
Je hoeft dit echt niet te onthouden hoor, maar het geeft je een idee hoe dit mechanisme werkt en waarom we dus met nul beginnen als we array elementen gaan tellen. Het laat je dus ook zien waarom een array (of variabele) correct gedefinieerd moet worden, waarbij het datatype toch wel belangrijk is zodat de Arduino weet in wat voor stappen het door het geheugen moet gaan om de juiste waarde te vinden.
Vergeet dus niet: Tellen begint bij nul!

3. Multi-Dimensionale Arrays

We hebben het al gezegd, de array in z'n eenvoudigste vorm, is een simpele lijst. Het is **"plat".**
We kunnen in een array ook array's zetten – een beetje complex misschien, maar daardoor kunnen we zogenaamde **multi-dimensionale** arrays maken.
Wat zeg je nou?

A. Dimensies

OK, ik kan me voorstellen dat je nog niet met dimensies hebt gewerkt, althans niet bewust, dus ik zal een simpele uitleg proberen te geven.
Laten we voor de eenvoud een dimensie zien als een richting waarin je jezelf kunt bewegen.
Als eerste dimensie: je kunt links en rechts bewegen.
Dus we kunnen horizontaal bewegen en in de wiskunde noemen we dat de X-as. We zouden dat in een tabel bijvoorbeeld een kolom kunnen zien: een simpel lijstje. Lastig is wel dat in de wiskunde we horizontaal bewegen (regel) terwijl we in de programmeertaal C en in ons boodschappen lijstje verticaal bewegen (kolom).

> *De **eerste dimensie is horizontaal (breedte)**,*
> *zeg maar een **KOLOM** in een tabel (een lijstje),*
> *of de **X-as** in de wiskunde.*

Bedenk dat we ook omhoog en omlaag kunnen bewegen – en dit is weer een andere dimensie! Dit is dus verticaal bewegen en in de wiskunde noemen we dat de Y-as, of in een tabel zouden dit kolommen kunnen zijn.
Niet vergeten: de wiskundige verticale beweging is dus net andersom in de taal "C".

> *De **tweede dimensie is verticaal (hoogte)**,*
> *zeg maar een to be a **REGEL** in een tabel (meerder lijstjes),*
> *of de **Y-as** in de wiskunde.*

We kunnen nog meer dimensies bedenken, maar in de array voorbeelden zal ik daar niet op in gaan.
Voor wie nieuwsgierig is: we hebben allemaal weleens van 3D gehoord, of het nou voor een 3D film was, of een 3D TV, het maakt niet uit. Allen bieden ze "diepte".
We kunnen dus vooruit en achteruit bewegen en in de wiskunde noemen we dat de Z-as.

> *De **derde dimensie is diepte**,*
> *In de wiskunde de **Z-as**.*

We weten dus nu dat er 3 dimensies zijn, maar we kunnen zo eindeloos doorgaan. Een nadeel echter is dat we meer dan 3 dimensies vaak niet meer kunnen bevatten in ons hoofd. Maar voor wie het wil weten: de 4de dimensie is tijd – dus bewegen in tijd.
Voor onze arrays blijven we lekker bij 1 or 2 dimensies omdat het tekenen van een 3D tabel al lastig wordt, laat staan een 4D tabel.

B. Enkelvoudige Dimensionale Array

Een enkelvoudige (single) dimensie array kunnen we het eenvoudigste zien als en simpele lijst, of als een gewone **KOLOM**, zoiets als dit:

1D Array voorbeeld	
Index	**Value**
0	A
1	B
2	**C**
3	D
4	E

We bewegen dus horizontaal, links-rechts, of te wel over X-as.
In dit voorbeeld heeft de array 5 karakters (niet te verwarren met een string, want die zou met een NULL karakter eindigen!!) met de letters “ABCDE”. We weten ook al dat we elementen eenvoudig met een index nummer kunnen benaderen, en om de “C”-waarde te benaderen doen we dus zoiets als: **variable[2]**.

```
variabele[index]
```

Het toewijzen van initiele waarden hebben we al gezien: bool LampenAan[5] = { false, false, false, false, false };

C. Twee Dimensionale Array

Als we nu gaan kijken naar 2 dimensies, dan kunnen we ons dat voorstellen als **REGELS** en **KOLOMMEN** van een tabel, dus de X- en Y-as in de wiskunde.
Gelukkig is dat ook niet moeilijk te bevatten. Zie het als een tabel, b.v. een school rooster of een tabel in een programma zoals Excel.

2D Array voorbeeld					
Index 2 Index 1	**0**	**1**	**2**	**3**	**4**
0	A	F	K	P	U
1	B	G	**L**	Q	V
2	C	H	M	R	W
3	D	I	N	S	X
4	E	J	O	T	Y

Onze 2D array bevat dus 25 waarden en wel: “ABCDEFGHIJKLMNOPQRSTUVWXY”.

Omdat dit een 2D array is, heeft het ook 2 indexen die we gebruiken moeten om een element te benaderen:

variabele[index1, index2]

Als we dus de letter “L” willen benaderen, dan doen we dat dus als volgt: **variabele[1][2]**.
Merk op: je kunt dit ook schrijven als: variabele[1,2]
Het initieel toewijzen van een 1D array hebben we al gezien: bool LampenAan[5] = { false, false, false, false, false };
Het toewijzen van waarden voor een 2D array is wat lastiger, het is immers niet meer een simpel lijstje!
Weet je nog dat we zeiden: arrays in een array? Nou dat geldt dus ook voor de waarden die we gaan toewijzen.
We zagen voorheen dat we een “set” met waarden door konden geven door de “set” te omringen met accolades en de waarden te scheiden met komma’s: { 1,2,3 }
Omdat we arrays in een array plaatsen moeten we dus voor elke “kolom” zo’n setje maken en meegeven als waarde. Maar niet vergeten dat we waarden (en dus ook de setjes) scheiden met komma’s en omringen ze met accolades.
We krijgen daarom zoiets als: datatype naam = { { setje1 }, { setje2 }, {setje3} };
Hierbij is zo’n setje dus zoiets als: { 1,2,3 }
In een stukje code, om voorgaand voorbeeld tabel te maken, zou dat er dus zo uit zien.
Niet vergeten: het zijn karakters, dus een enkel aanhalingsteken gebruiken!

```
boolean variable[5,5] = { { 'A','B','C','D','E' },
                          { 'F','G','H','I','J' },
                          { 'K','L','M','N','O' },
                          { 'P','Q','R','S','T' },
                          { 'U','V','W','X','Y' } };
```

Zie je dat de setjes gescheiden zijn door komma’s en het geheel nog eens extra omringd is met accolades? Ieder setje wordt als een “waarde” gezien.

*We zouden nu verder kunnen gaan met een **3-dimensional earray**, maar omdat het diepte toe voegt (Z-as), zou dit betekenen dat onze tabel een **kubus** zou worden.*

D. Een toepassing voorbeeld van een 2D array

Nu vraag je jezelf misschien: wat moet ik nou met een array die meerdere dimensies heeft?
Stel je voor, we gebruiken een Arduino om de lichten aan te sturen in meerdere kamers.
Elke kamer heeft 4 lampen en we hebben 3 kamers.Een tabel, een 2D array dus, laten we het weer LampenAan noemen, zou dat kunnen omvatten en er zo uit kunnen zien.

2-D Array Voorbeeld			
Kamer: Lamp:	**0**	**1**	**2**
0	true	false	true
1	true	false	false

2-D Array Voorbeeld			
Kamer: Lamp:	**0**	**1**	**2**
2	true	false	true
3	true	false	false

Zie dit dus als een lijstje van lampen, voor elke kamer.
Dus als we de status van de lampen in de eerste kamer willen zien (index = 0!!!), dan kijken we naar de volgende waarden: LampenAan[0,0], LampenAan[0,1], LampenAan[0,2] en LampenAan[0,3]. Op deze manier kunnen we onze waarden gemakkelijk opslaan en eenvoudig benaderen, zeker als we een “for”-loop hierbij gaan gebruiken.
Stel we willen alle lampen in alle kamers door lopen om te kijken of ze aan staan of niet. Je zou natuurlijk 12 aparte variabelen kunnen maken (LampAan1, LampAan2, etc), of elke element van de array individueel intypen. Maar met een “for”-loo[in een “for”-loop gaat dat vlotter:

```
#define kamers 3
#define lampenPerKamer 4
void setup() {
  Serial.begin(9600);
  boolean LampenAan[kamers][lampenPerKamer] = { { true, true, true, true },
                                                { false, false, false, false },
                                                { true, false, true, false } };
  for(int kamer=0; kamer<=kamers-1; kamer++) {     // Tel de KAMERS
    for(int lamp=0; lamp<=lampenPerKamer-1; lamp++) {   // Tel de LAMPEN
                                                        //voor elke KAMER
      Serial.print("Kamer ");
      Serial.print(kamer);
      Serial.print(" Lamp ");
      Serial.print(lamp);
      if(LampenAan[kamer][lamp]) {
        Serial.println(" is AAN");
      }
      else
      {
        Serial.println(" is UIT");
      }
    } // einde lamp loop
    Serial.println(); // lege regel tussen kamers
  } // einde kamer loop
}
void loop() {
  // leave empty for now
}
```

Dus even kort: we hebben 3 kamers, met elk 4 lampen, dus: 3 lijsten, met elk 4 items op de lijst!

Zie je hoe ik de “#define” stiekem heb gebruikt om een constante te definiëren?
Ik heb dat expres gedaan omdat we deze twee waarden op meerdere plaatsen gebruiken in ons programma:

Bij de array definitie
In de “for”-loops

Zie je ook dat ik in de “for”-loop de betreffende constante gebruik maar verminderd me 1?
Dat heb ik gedaan omdat wij mensen 3 kamers tellen: Kamer 1, 2 en 3.
Maar … je raad het al: bij array’s starten we met tellen bij nul.
Dus: Kamer 0, 1, en 2.
Vandaar dus de “-1” om op dezelfde telling uit te komen zoals we die voor arrays moeten gebruiken als we elementen gaan aanspreken.
Zie je ook dat ik zinvolle namen heb gekozen voor de variabelen?
Het programma wordt veel duidelijker door zinvolle namen te geven aan variabelen en het gebruik van dit soort constanten te gebruiken in plaats van zinloze namen of gewoon nummers?
Wist je overigens als we een constructie, zoals de “for”-loop, in een zelfde constructie (dus ook een “for”-loop) plaatsen, dat men dit “nesting” noemt? Leuk om te weten misschien maar niet super belangrijk om te onthouden hoor.
Zie je ook dat als we 2 van dit soort “for”-loops in elkaar zetten, dat de code soms wat lastiger wordt om te lezen? Dat is nou precies de reden waarom ik bij de “accolade sluiten” (}) even een opmerking plaats zodat ik weet waar de accolade bij hoort.
De uitleg voor deze code is misschien wat voor de hand liggend, maar voor de zekerheid even:
We starten een “for”-loop om de kamers te gaan tellen (kamer) en voor iedere “kamer” gebruiken we een “for”-loop om de lampen te tellen en doorlopen.
Net na de “for”-loop voor de lampen, maar nog steeds in de “for”-loop van de kamers zie je dat ik ook een lege “Serial.println()” heb geplaatst – dit resulteert in een lege regel tussen de kamer lijsten.
Vergeet dus niet dat de “for”-loop voor de lampen voor elke kamer herhaald wordt!
De output nog even voor de volledigheid:

```
Kamer 0 Lamp 0 is AAN
Kamer 0 Lamp 1 is UIT
Kamer 0 Lamp 2 is AAN
Kamer 0 Lamp 3 is UIT

Kamer 1 Lamp 0 is UIT
Kamer 1 Lamp 1 is UIT
Kamer 1 Lamp 2 is UIT
Kamer 1 Lamp 3 is AAN

Kamer 2 Lamp 0 is AAN
Kamer 2 Lamp 1 is UIT
Kamer 2 Lamp 2 is AAN
Kamer 2 Lamp 3 is UIT
```

Uitdagingen arrays:

1. Maak de bovenstaande voorbeelden en probeer zelf te ontdekken hoe arrays werken, zowel in 1 als 2 dimensies.

17. Veel succes met deze Arduino cursus.

Uiteraard kan je op de Arduino robot nog andere componenten aansluiten.

Denk maar aan de analoge ingangen (LDR, temperatuursensor, potmeter), een LCD scherm, 7-segment display, remote control sturing, servo en stappenmotoren enz…

Zie hiervoor de gevorderden deel 1 + 2 cursus.

Houd zeker de www.edulab.be website in't oog voor een vervolg cursus of kijk eens op www.arduino.cc voor voorbeeld oefeningen.

Je kan ons ook terugvinden op facebook.

EDULAB
FRANK MARCHAL
HOMMELHEIDE 45
B-3500 HASSELT
+32 (0) 498 82 15 90
INFO@EDULAB.BE
WWW.EDULAB.BE