

# Python Cookbook voor MTE

W.M. Jutte en J. Antonissen

22 september 2021



# Inhoudsopgave

<b>1 CircuitPython</b>	<b>1</b>
<b>1 Getting Started</b>	<b>3</b>
1.1 Introduction . . . . .	3
1.2 Welcome to CircuitPython tutorial . . . . .	3
1.3 Plotten met Mu editor . . . . .	3
1.4 Een extra woord over bibliotheken . . . . .	5
1.5 Veilig omgaan met het bordje . . . . .	5
1.6 Bordje updaten . . . . .	7
<b>2 Frequently Asked Questions</b>	<b>9</b>
2.1 Bordje problemen . . . . .	9
2.2 Code Problemen . . . . .	10
<b>3 Ledjes</b>	<b>11</b>
3.1 Extern LEDje laten branden . . . . .	12
3.2 Meerdere LEDjes . . . . .	14
3.3 Meerdere LEDjes met minder pinnen door middel van een shift-register . . . . .	16
3.4 Neopixel . . . . .	19
3.5 LCD Screen . . . . .	20
<b>4 Sensoren</b>	<b>23</b>
4.1 Potentiometer . . . . .	24
4.2 Een licht aan en uit zetten op basis van de licht intensiteit . . . . .	26
4.3 Een licht feller laten branden als het donkerder wordt. . . . .	28
4.4 Ultrasone afstandssensor . . . . .	29
4.5 Joystick . . . . .	31
4.6 RC Lowpass Filter . . . . .	33
<b>5 Aandrijving</b>	<b>35</b>
5.1 Transistors . . . . .	35
5.2 DC Motor . . . . .	38
5.3 Twee DC motoren . . . . .	40
5.4 Servo . . . . .	43

5.5	Twee DC motoren, een servo en een toeter aansturen met een Joystick . . . . .	45
<b>6</b>	<b>Netwerken</b>	<b>51</b>
6.1	Maken van een WIFI access point . . . . .	51
6.2	Data sturen van client naar server over WIFI . . . . .	52
6.2.1	Server . . . . .	52
6.2.2	Client . . . . .	55
6.3	Besturen van een servo met laptop . . . . .	55
6.3.1	Server . . . . .	56
6.3.2	Client . . . . .	57
6.4	Gebruiken van keyboard toetsaanslagen . . . . .	58
6.5	Gebruik van keyboard om een Servo aan te sturen over WIFI . .	59
6.5.1	Client: . . . . .	59
6.5.2	Server . . . . .	63
<b>7</b>	<b>Remote Controlled boot</b>	<b>65</b>
7.1	Circuit . . . . .	65
7.2	Code . . . . .	66
7.2.1	Computer client side . . . . .	66
7.2.2	Boat server side . . . . .	69

**Deel I**

**CircuitPython**



# Hoofdstuk 1

## Getting Started

### 1.1 Introduction

De CircuitPython omgeving is ontworpen voor beginners in software en elektrotechnica. Voordat je met het CircuitPython bordje aan de slag kan moet het één en ander worden geïnstalleerd en geüpdatet. Adafruit heeft een beginners tutorial voor CircuitPython die je gaat doorlopen in 1.2.

### 1.2 Welcome to CircuitPython tutorial

Alle andere benodigdheden om te starten met CircuitPython wordt behandelt in Adafruit's kennismaking tutorial. Doorloop Adafruits' **"Welcome to CircuitPython"** tutorial tot en met het kopje **"CircuitPython Libraries"** op:

<https://learn.adafruit.com/welcome-to-circuitpython/>

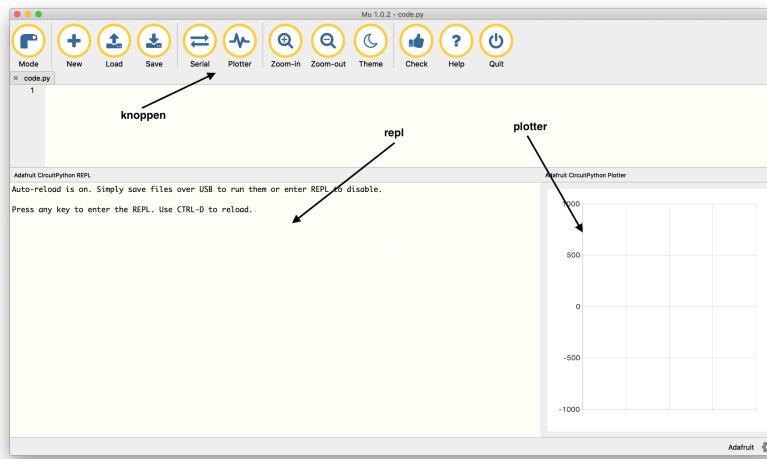
Op deze website staat de basis van circuit python, lees dit goed door. Download ook de Mu editor zoals op de guide staat aangegeven.

<https://codewith.mu/>

### 1.3 Plotten met Mu editor

In de Mu editor zit een ingebouwde plotter. Dit is erg handig om bijvoorbeeld sensordata direct te plotten. Om gebruik te maken van de plotter moet data in een tuple geprint worden. Open mu editor om dit te testen. Klik op Serial en Plotter bovenaan mu editor. Als het goed is ziet je scherm eruit als figuur 1.1. Als er een programma loopt, klik dan in het vak van de repl en doe ctrl+c om de code te stoppen en druk op een willekeurige knop om in de

repl te gaan. Print nu een paar tuples. Een tuple is een datastructuur dat normale haakjes () gebruikt. Type in de repl bijvoorbeeld de volgende commands, print((1,2)), print((3,2)), a=(0,0), print(a). Als het goed is zie je dan hetzelfde als in figuur 1.2. Probeer ook één getal te plotten door bijvoorbeeld print((4)) te typen. Als het goed is zie je niets gebeuren. Dit komt door een eigenaardigheid in python dat een enkel getal in een tuple geschreven moet worden als (4,) en niet (4) . Als je print((4,)) typt wordt de waarde wel geplot.



Figuur 1.1: Mu editor met de REPL en plotter open



Figuur 1.2: Plotten van tuples

<https://learn.adafruit.com/make-it-graph-plot>

## 1.4 Een extra woord over bibliotheken

In deze handleiding wordt gebruik gemaakt van de volgende bibliotheken:

- adafruit\_bus\_device map
- adafruit\_esp32spi map
- adafruit\_hcsr04.mpy
- adafruit\_motor map
- neopixel.mpy

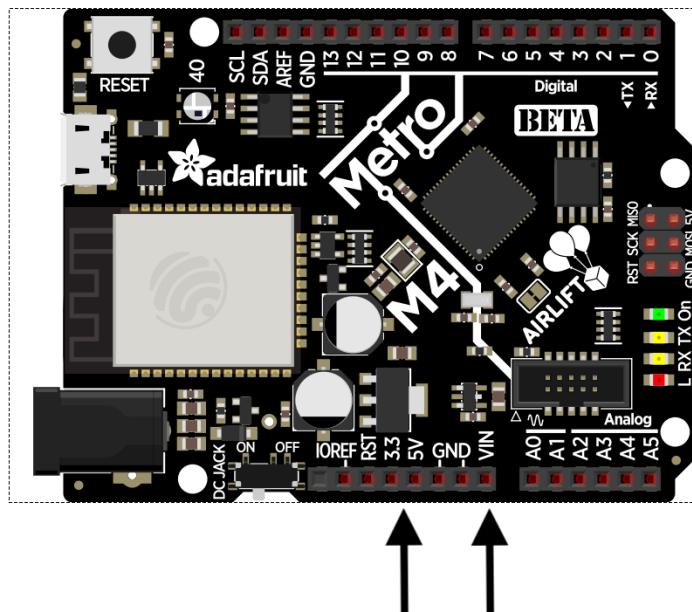
Haal de up-to-date bibliotheken uit adafruits library bundle en zet ze in de lib folder van je CIRCUITPY drive. De bundel is te downloaden op

<https://circuitpython.org/libraries>.

Let op, je kan niet alle bibliotheken op het bordje zetten, hiervoor is niet genoeg ruimte. Zet dus alleen de bibliotheken die je gebruikt op het bordje.

## 1.5 Veilig omgaan met het bordje

De bordjes kunnen kapot gaan als de pinnen op het bordje teveel stroom moeten leveren of als er teveel stroom het bordje instroomt. Het Adafruit Metro M4 Express bordje heeft een 3.3 V, 5 V en een  $V_{in}$  output pin, te zien in figuur 1.3. Deze pinnen kunnen bij elkaar maximaal 500 mA leveren en kunnen gebruikt worden om je projectje te voorzien van stroom. Alle andere pinnen op het bordje zijn I/O pinnen en zijn niet bedoeld om zulke hoge stroom te leveren. In de datasheet van de ATSAM51D wordt een maximum van 2 mA per I/O pin aangegeven. Als je hier niet aan houdt is het mogelijk dat componenten in het bordje doorbranden. Zie <https://www.alanzucconi.com/2016/09/17/how-to-destroy-an-arduino-board/> voor meer informatie over hoe je een bordje kapot kan maken.



Figuur 1.3: De spanningsbronnen op de Metro M4 Express

## 1.6 Bordje updaten

Let op: dit alleen uitvoeren als de docent hier om vraagt!

Het bordje kan geüpdateet worden door het bordje in de bootloader modus te zetten en het update bestand erop te slepen.

Download eerst de update van de bootloader op de volgende link (onderaan op pagina). Let op download alleen de stabiele versie, dus geen alpha of beta builds.

[https://circuitpython.org/board/metro\\_m4\\_airlift\\_lite/](https://circuitpython.org/board/metro_m4_airlift_lite/)

Klik twee keer op reset om Het bordje in bootloader modus te zetten. Sleep vervolgens het update bestand (de nieuwe bootloader) op het bordje. Het bordje zal opnieuw opstarten. Na het updaten van de bootloader zal CircuitPython van het bordje zijn verdwenen, deze moet er dus opnieuw worden opgezet. Download op dezelfde pagina ook de laatste versie van CircuitPython, zet het bordje weer in bootloader modus en sleep het circuitPython bestand naar de bootloader. Het bordje zal weer opnieuw opstarten.

Gefeliciteerd, je hebt nu de bootloader en CircuitPython geüpdateet!



## **Hoofdstuk 2**

# **Frequently Asked Questions**

### **2.1 Bordje problemen**

Het aansluiten van het bordje kan soms lastig zijn. Hieronder een aantal tips:

#### **USB kabel**

Voor het aansluiten van het bordje heb je een USB-A naar micro-USB kabel nodig. Sommige van deze kabels zullen niet werken, deze leveren bijvoorbeeld wel stroom naar geen date. Als het bordje niet werkt,bijvoorbeeld als hij wel aan gaat maar windows geen verbinding kan maken, probeer dan dus eerst altijd een andere kabel. Pas na 3 kabeltjes kan je met enige zekerheid zeggen dat je dit probleem hebt verholpen.

#### **Anti virus**

Sommige anti-virus software blokkeren USB apparaten. Probeer dus ook eens je bordje aan te sluiten terwijl je anti-virus programma uit staat.

#### **Bordje niet gevonden**

Het kan zijn dat de verkeerde bootloader op het bordje staat. Volg de stappen in het hoofdstuk bordje updaten om de bootloader te updaten en Cirquit-Python op het bordje te zetten.

[https://circuitpython.org/board/metro\\_m4\\_airlift\\_lite/](https://circuitpython.org/board/metro_m4_airlift_lite/)

## 2.2 Code Problemen

Bij het coderen kunnen veel fouten krijgen. Een computer (en dus ook het bordje) werkt alleen als de code erop foutloos is.

Let bij het schrijven van het programma op het volgende:

### Serial en Code check

In MU zitten de 'Serial (of REPL)' en 'Code Check' knoppen. Hiermee kan je goed controleren of er fouten in je code zitten. De Serial knop geeft in een schermpje onderaan aan welke fouten er in je code zitten, hierbij staat ook op welke regel zich de fout bevindt. De Code Check knop controleert de code tussen de code door.



Figuur 2.1: Serial en Code Check

### Code problemen

Kom je er niet uit ? check dan het volgende lijstje:

- Heet het bestand op het bordje code.py ?
- Staat jou code.py bestand op het bordje ?
- Staat jou code in het bestand code.py ?
- Zitten er typ fouten in je code ?
- Staan er onnodige spaties of hoofdletters in je code ?
- Stuur je de poorten aan die je wilt gebruiken ?

# **Hoofdstuk 3**

## **Ledjes**

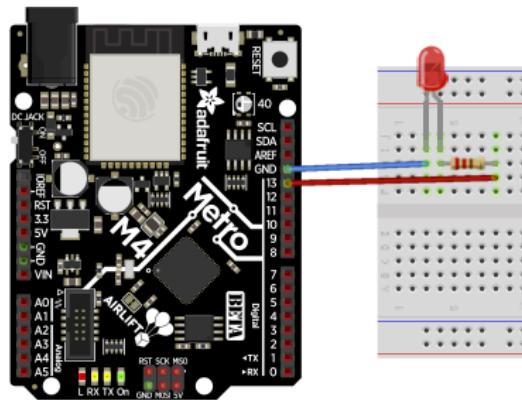
In de introductie heb je kennis gemaakt met CircuitPython en Mu editor en heb je een ledje op het bordje zelf aangestuurd. Een micro-controller wordt voornamelijk gebruikt om externe hardware aan te sturen. In dit hoofdstuk worden ledjes op verschillende manieren aangestuurd. Een ledje laten branden is niet heel spannend, maar het is één van de simpelste manieren om code te testen. Het maken van een led circuit is erg eenvoudig en je krijgt gelijk visuele feedback. Ook zit tussen het aansturen van een ledje of een motor weinig verschil.

### 3.1 Extern LEDje laten branden

**Probleem:** Je wilt een externe LED laten knipperen.

**Benodigdheden:**

- 1x LED
- 1x  $680\Omega$  weerstand (Of weerstand die in de buurt komt)



Figuur 3.1: Led Circuit

**Oplossing:**

```

1  ## LED.py
2
3 #importeer modules
4 import board
5 import digitalio
6 import time
7
8 #initialiseer Digital Output op pin D13
9 led = digitalio.DigitalInOut(board.D13)
10 led.direction = digitalio.Direction.OUTPUT
11
12 #Herhaal oneindig:
13 while True:
14     led.value = True #zet pin waarde hoog
15     time.sleep(0.2) #wacht 0.5 seconde
16     led.value = False #zet pin waarde laag
17     time.sleep(0.8) #wacht 0.5 seconde

```

**Discussie:** Om een ledje te laten knipperen zijn drie modules nodig. De "board" module, zodat de pinnen aangeroepen kunnen worden met het nummer dat op het bordje staat. De "time" module om gebruik te kunnen maken van tijd en timing functies en de digitalio module zodat de I/O pinnen ingesteld kunnen worden.

De LED zit aangesloten op pin 13 en kan daarom aan en uit gezet worden door de spanning op pin 13 hoog en laag te zetten. In de code kan je controle krijgen over pin 13 met het gebruik van de DigitalInOut() functie van de digitalio module. De functie neemt als parameter het nummer van de pin dat je wilt gebruiken en creëert dan een 'object', zodat je pin 13 met code kan aansturen. In de code krijgt het object de naam led toegewezen. Een I/O pin kan zowel voor het uitlezen en het aansturen van spanning gebruikt worden. In de code moet daarom verteld worden hoe we de pin willen gebruiken. Om de I/O pin voor aansturen te gebruiken wordt de "directioneigenschap" van het led object op output gezet.

Na het initialiseren van de pin kan de spanning hoog en laag gezet worden door de "valueeigenschap" van het led object respectievelijk de waarde True of False te geven. De time.sleep() functie wordt gebruikt om de code te vertragen tussen het aan en uit zetten.

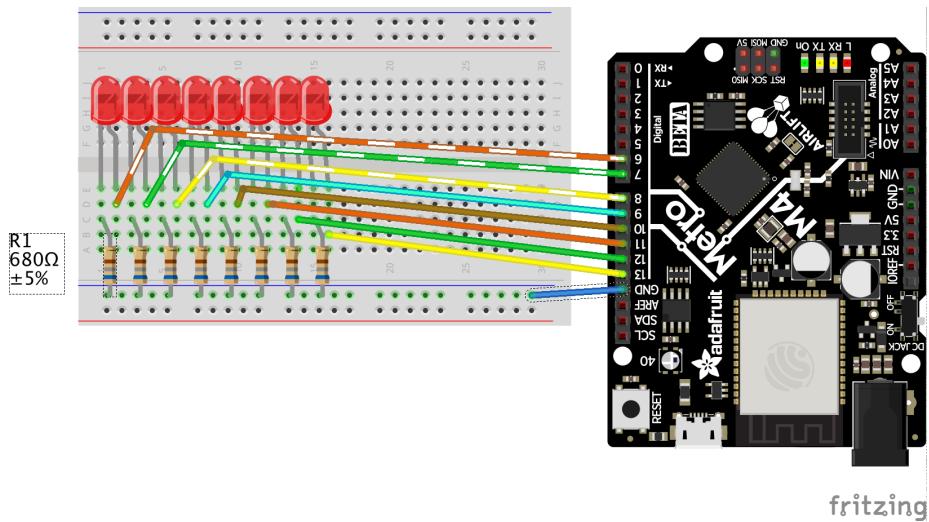
## 3.2 Meerdere LEDjes

**Probleem:** Je wilt meerdere ledjes tegelijkertijd en onafhankelijk van elkaar aansturen.

**Benodigdheden:**

- 8x LED
- 8x  $680\Omega$  weerstand

**Oplossing :**



Figuur 3.2: Acht Ledjes

```

1 ## Multiple_LED.py
2
3 # importeer modules
4 import board
5 import digitalio
6 import time
7
8 # initialiseer Digitale Outputs
9 led0 = digitalio.DigitalInOut(board.D13)
10 led0.direction = digitalio.Direction.OUTPUT
11 led1 = digitalio.DigitalInOut(board.D12)
12 led1.direction = digitalio.Direction.OUTPUT
13 led2 = digitalio.DigitalInOut(board.D11)
14 led2.direction = digitalio.Direction.OUTPUT
15 led3 = digitalio.DigitalInOut(board.D10)
16 led3.direction = digitalio.Direction.OUTPUT
17 led4 = digitalio.DigitalInOut(board.D9)
18 led4.direction = digitalio.Direction.OUTPUT
19 led5 = digitalio.DigitalInOut(board.D8)
20 led5.direction = digitalio.Direction.OUTPUT
21 led6 = digitalio.DigitalInOut(board.D7)
22 led6.direction = digitalio.Direction.OUTPUT
23 led7 = digitalio.DigitalInOut(board.D6)
24 led7.direction = digitalio.Direction.OUTPUT
25
26 ledlist=[led0 ,led1 ,led2 ,led3 ,led4 ,led5 , led6 , led7 ]
27
28
29 # Herhaal oneindig :
30 while True:
31     for led in ledlist:
32         led.value=True
33         time.sleep(0.2)
34     for led in reversed(ledlist):
35         led.value=False
36         time.sleep(0.2)

```

**Discussie:** Voor elk ledje moet 1 pin geïnitialiseerd worden. Voor 8 ledjes moeten dus 8 pinnen op digitale output gezet worden. Dit gaat hetzelfde als met 1 ledje als in recept 3.1. Daarna worden alle ledjes in een lijst gezet, zodat met een for loop door de ledjes gelooped kan worden. In de oneindige while loop gaat het programma gelijk de eerste for loop in. Hier worden de ledjes één voor één aangezet in de volgorde van ledlist met 0.2s wachtijd tussen elk ledje. Vervolgens gaat het de tweede for loop in, waar de ledjes in omgekeerde volgorde van de ledlist worden uitgezet.

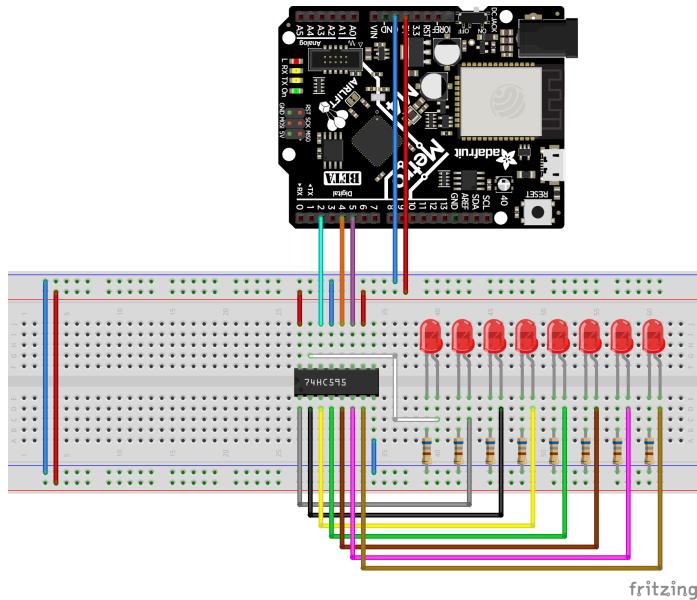
### 3.3 Meerdere LEDjes met minder pinnen door middel van een shiftregister

**Probleem:** Je wilt meerdere ledjes aansturen, maar wilt niet per ledje 1 pin opofferen.

**Benodigdheden:**

- 8x LED
- 8x  $680\Omega$  weerstand (in dit geval mag een lagere weerstand van  $220\Omega$  gebruikt worden omdat de stroom niet door de I/O pinnen wordt geleverd)
- 1x SN74HC595N shift register

**Oplossing :**



Figuur 3.3: Acht ledjes met een shiftregister

### 3.3. MEERDERE LEDJES MET MINDER PINNEN DOOR MIDDEL VAN EEN SHIFTREGISTER17

```
1 ## Shiftregister.py
2
3 import board
4 import digitalio
5 import time
6
7 SER_pin = board.D2
8 RCLK_pin = board.D4
9 SRCLK_pin = board.D5
10
11 SER = digitalio.DigitalInOut(SER_pin)
12 SER.direction = digitalio.Direction.OUTPUT
13 RCLK = digitalio.DigitalInOut(RCLK_pin)
14 RCLK.direction = digitalio.Direction.OUTPUT
15 SRCLK = digitalio.DigitalInOut(SRCLK_pin)
16 SRCLK.direction = digitalio.Direction.OUTPUT
17
18 val_register=[False for i in range(8)]
19
20
21
22 def writereg(val_register):
23     RCLK.value = False
24
25     for i in reversed(range(8)):
26         SRCLK.value = False
27         SER.value = val_register[i]
28         SRCLK.value = True
29     RCLK.value = True
30
31
32 writereg(val_register)
33
34 while True:
35     for i in range(8):
36         val_register[i] = True
37         print(val_register)
38         writereg(val_register)
39         time.sleep(0.1)
40
41     for i in reversed(range(8)):
42         val_register[i] = False
43         writereg(val_register)
44         print(val_register)
45         time.sleep(0.1)
```

**Discussie:** Een shiftregister wordt gebruikt om een groot aantal ledjes (of iets anders) aan te sturen, zonder dat je voor elk ledje een nieuwe pin nodig hebt, zoals in recept 3.2. Je kunt met 3 I/O pinnen een oneindig aantal ledjes aansturen als je meerdere shiftregisters aan elkaar koppelt. Het gebruik van een shiftregister is wel iets complexer. Dus hoe werkt het?

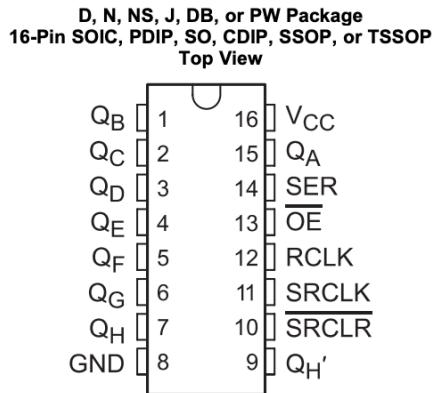
Het schema van het shiftregister is te zien in figuur 3.4.

1. Qa, Qb.... ,Qh zijn de pinnen waar de ledjes op aangesloten worden.
2. Vcc en GND zijn de pinnen die de stroom aan zowel de ledjes als de IC leveren.
3. De  $\overline{OE}$  pin schakelt Qa-Qh tegelijk uit als er een hoge spanning op staat. In ons geval is het niet nodig om Qa-Qh tegelijk uit te zetten, dus kan de pin direct op ground aangesloten worden.
4. De  $\overline{SRCLR}$  pin cleared het shift register als er een lage spanning op gezet wordt. Dit is onnodig, dus deze pin wordt aangesloten op 3.3V, net als Vcc.
5.  $Q_H'$  kun je gebruiken om meerdere IC's aan elkaar te koppelen, dat hier niet wordt gedaan. Deze kan dus open blijven.

De overgebleven SER, RCLK en SRCLK worden gebruikt om Qa-Qh in te stellen, waar de ledjes aan vast zitten. Een shiftregister werkt met het principe van doorschuiven. Het shiftregister heeft 8 stukjes geheugen, één register per Q pin. Als de SRCLK pin van een lage naar een hoge spanning gebracht wordt, wordt een 1 of een 0 in het register van Qa gezet, afhankelijk van of de spanning op de SER pin respectievelijk hoog of laag is. Tegelijkertijd schuift de waarde die in het register van Qa stond naar Qb, van Qb naar Qc, etc.. Door de gewenste waardes in de goede volgorde op de SER pin te zetten en de SRCLK van laag naar hoog te brengen kunnen de gewenste waarden in de registers van de Q-pinnen gezet worden. Voordat de waarden actief worden moet de RCLK van laag naar hoog gebracht worden.

Het gebruik van het shiftregister werkt als volgt. Eerst zet je RCLK en SRCLK laag. Wat je als eerst op de SER pin zet bepaald uiteindelijk of de laatste pin  $Q_H$  aan of uit gaat staan. Als je het ledje op  $Q_H$  aan wilt, zet je een hoge spanning op de SER pin en breng je SRCLK van laag naar hoog. In het register van Qa komt nu een 1. Vervolgens breng je SRCLK weer laag en bepaal je of  $Q_G$  aan of uit moet. Als Qh uit moet, zet je een lage spanning op de SER pin en breng je SRCLK weer naar hoog. De 1 van Qa schuift door naar Qb en in Qa komt nu een 0. Dit doe je in totaal acht keer tot alle waarden in de registers staan. Tot slot breng je RCLK van laag naar hoog om het definitief te maken en gaan de gewenste ledjes aan de Q-pinnen branden.

Dit proces is gecodeerd in de writereg() functie. Je geeft de functie een lijst met welke van de Qa-Qh aan en uit moeten. De functie loopt dan van achter naar voren door de lijst, omdat je het laatste ledje als eerste in moet voeren. Als in alle registers de juiste waarde staat wordt een hoge spanning op RCLK gezet om het definitief te maken.



Figuur 3.4: Pin-out diagram van SN74HC595N

## 3.4 Neopixel

**Probleem:** Je wilt de neopixel (RGB LED) op het bordje gebruiken.

**Oplossing:** CircuitPython heeft een speciale bibliotheek voor het gebruik van de neopixel. Code om de RGB over te laten springen van rood naar groen naar blauw is hieronder te zien.

```

1  ## Neopixel.py
2
3 import time
4 import board
5 import neopixel
6
7 #maak Neopixel object
8 led = neopixel.NeoPixel(board.NEOPIXEL, 1)
9
10 led.brightness = 0.3
11
12 while True:
13     led[0] = (255, 0, 0)
14     time.sleep(0.5)
15     led[0] = (0, 255, 0)
16     time.sleep(0.5)
17     led[0] = (0, 0, 255)
18     time.sleep(0.5)
```

**Discussie:** Zorg er eerst voor dat de neopixel bibliotheek in de lib folder op je bordje staat. Om vervolgens gebruik te maken van de neopixel bibliotheek moet de bibliotheek eerst worden geïmporteerd. Vervolgens kan gebruik gemaakt worden van de NeoPixel() functie om een NeoPixel object te maken dat controle

geeft over de NeoPixel via simpele commando's in de code. De NeoPixel() functie heeft als eerste parameter de I/O pin waar de neopixel op aangesloten is en als tweede parameter het aantal neopixels. In ons geval zit één neopixel aangesloten op een interne pin met de naam board.NEOPIXEL. Na het creëren van een neopixel object met de naam 'led', kan de neopixel heel makkelijk worden ingesteld door led aan te roepen in de code. De felheid kan worden ingesteld door led.brightness aan te passen en de kleur van de neopixel kan worden ingesteld door de verhouding van rood/groen/blauw licht in te stellen. Dit kan door elk van de kleuren een waarde tussen de 0 en 255 te geven.

**Extra informatie over het gebruik van de neopixel :**

<https://learn.adafruit.com/circuitpython-essentials/circuitpython-internal-rgb-led>  
<https://circuitpython.readthedocs.io/projects/neopixel/en/latest/>

### 3.5 LCD Screen

**Probleem:** Je wilt tekst naar een LCD screen printen.

**Benodigdheden:**

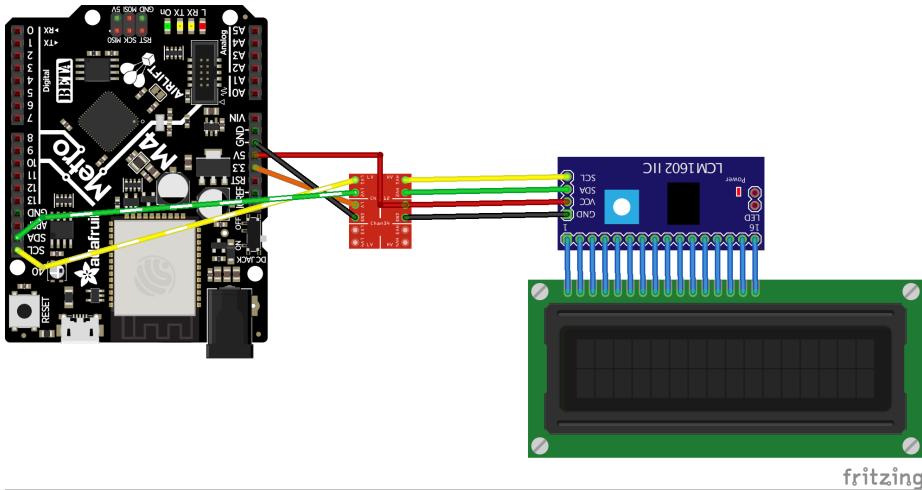
- 1x Bi-Directional Level-Shifter 5-3.3V (Niet nodig maar zorgt voor feller licht)
- 1x LCD scherm
- 1x PCF8574 i2c LCD controller

**Oplossing :**

```

1 ## LCD.py
2
3 from lcd.lcd import LCD
4 from lcd.i2c_pcf8574_interface import I2CPCF8574Interface
5 from lcd.lcd import CursorMode
6 import time
7
8
9
10
11 # Talk to the LCD at I2C address 0x27.
12 lcd = LCD(I2CPCF8574Interface(board.I2C(), 0x27), num_rows=2,
13             num_cols=16)
14 lcd.set_cursor_pos(0, 0)
15 lcd.print("Hello!\nHow are you?")
16 time.sleep(2)
17 lcd.clear()

```



Figuur 3.5: LCD Screen Circuit

**Discussie:** Tussen het LCD scherm en het CircuitPython bordje zit een level shifter, omdat het lcd scherm 5V logic gebruikt en het bordje 3.3V. De level shifter heeft een HV en LV pin, waarop de referentie spanning gezet moet worden. Zet op LV dus 3.3V en op HV 5V. De level shifter converteert dan automatisch een spanning van 3.3V op de pinnen LV1-LV4 naar een spanning van 5V op respectievelijk HV1-HV4. De level shifter werkt ook in de andere richting.

Download voor het gebruik van de code eerst de LCD bibliotheek van [https://github.com/dhalbert/CircuitPython\\_LCD](https://github.com/dhalbert/CircuitPython_LCD) door op *clone or download* en dan *download ZIP* te klikken. Unzip de .zip file en zet het mapje met de naam 'lcd' in de lib folder op je CircuitPython bordje. In de code worden eerst de benodigde modules geïmporteerd. Daarna wordt een LCD object gemaakt met de naam "lcd". De positie van de cursor kan verplaatst worden met `object.set_cursor_pos()` en er kan tekst geprint worden met `object.print()`. Met `object.clear()` wordt het scherm leeg gemaakt en wordt de cursor positie terug naar het begin verplaatst.

Doet de bibliotheek het niet ?

Pas dan de bibliotheek als volgt aan:

```

1 def _write4bits(self, value):
2     """Pulse the 'enable' flag to process value."""
3     with self.i2c_device:
4         self._i2c_write(value & ~PIN_ENABLE)
5         # This 1us delay is probably unnecessary, given the
6         # time needed
7         # to execute the statements.
8         microcontroller.delay_us(1)
9         self._i2c_write(value | PIN_ENABLE)
10        microcontroller.delay_us(1)
11        self._i2c_write(value & ~PIN_ENABLE)
12        # Wait for command to complete.
13        microcontroller.delay_us(100)

```

Voor het volgende stukje code:

```

1 def _write4bits(self, value):
2     """Pulse the 'enable' flag to process value."""
3     with self.i2c_device:
4         self._i2c_write(value & ~PIN_ENABLE)
5         # This 1us delay is probably unnecessary, given the
6         # time needed
7         # to execute the statements.
8         #microcontroller.delay_us(1)
9         self._i2c_write(value | PIN_ENABLE)
10        #microcontroller.delay_us(1)
11        self._i2c_write(value & ~PIN_ENABLE)
12        # Wait for command to complete.
13        #microcontroller.delay_us(100)

```

# **Hoofdstuk 4**

## **Sensoren**

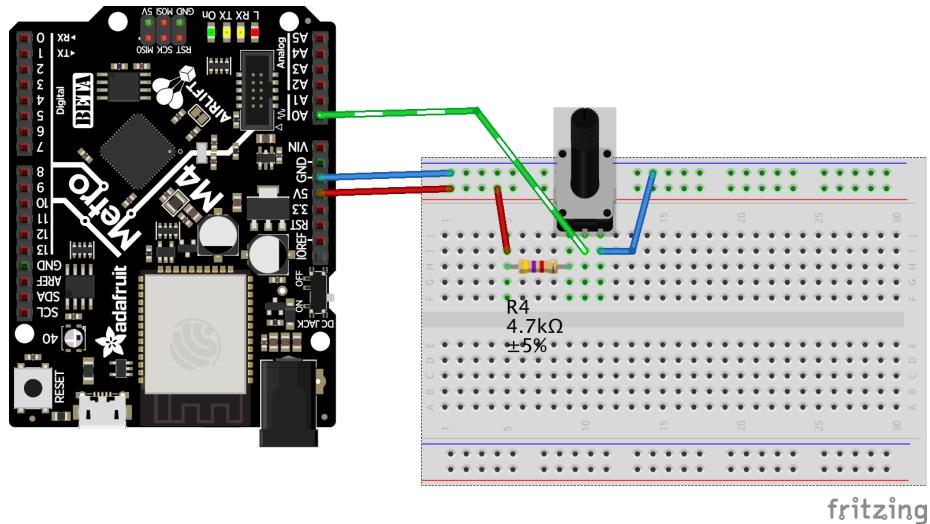
Informatie over de buitenwereld krijg je door middel van sensoren. Sensoren zijn er in allerlei soorten en maten, maar werken vaak op hetzelfde principe. Een fysische gebeurtenis wordt geconverteerd naar een elektrisch signaal dat kan worden verwerkt door de microprocessor. Dit hoofdstuk bevat recepten voor het uitlezen van een aantal sensoren.

## 4.1 Potentiometer

**Probleem:** Je wilt een potentiometer uitlezen.

**Benodigdheden:**

- 1x Potentiometer
- 1x  $4.7\text{k}\Omega$  weerstand



Figuur 4.1: Potentiometer Sensor Circuit

**Oplossing:**

```

1  ## Lightsensor_TEMT6000.py
2
3  import time
4  import board
5  from analogio import AnalogIn
6
7  #initiation
8  analog_in = AnalogIn(board.A0)
9
10 #Convert 16-bit value to voltage
11 def get_voltage(pin):
12     return (pin.value * 3.3) / 65535
13
14
15 while True:
16     print((get_voltage(analog_in),))
17     time.sleep(0.1)

```

**Discussie:** Een potentiometer is een variabele spanningsdeler. Een potentiometer verbindt zowel de eerste pin en tweede pin met een weerstand als de tweede en derde pin met een weerstand. Door aan de potentiometer te draaien verander je de verhouding tussen de 2 weerstanden. Een potentiometer van  $10\text{ k}\Omega$  heeft wel altijd een totale weerstand van  $10\text{ k}\Omega$  tussen de eerste en derde pin. Door aan de pin te draaien kan je de verhouding wel veranderen in bijvoorbeeld  $1\text{ k}\Omega$  tussen pin 1 en 2 en  $9\text{ k}\Omega$  tussen pin 2 en 3 of  $0\text{ k}\Omega$  tussen 1 en 2 en  $10\text{ k}\Omega$  tussen 2 en 3. Als  $3.3\text{ V}$  op pin 1 staat en pin 3 verbonden is met ground verandert de spanning op pin 2 evenredig met de stand van de potentiometer. Omdat de tweede pin van de potentiometer in dit circuit direct zit aangesloten op een I/O pin, wordt een extra resistor toegevoegd om een short-circuit te voorkomen. Dit kan gebeuren in het geval dat de weerstand tussen pin 1 en pin 2  $0\text{ }\Omega$  is.

Voor het uitlezen van een analoog signaal heeft CircuitPython de analogio bibliotheek. In de code hoef je alleen een object te maken door het analoge pin nummer als parameter aan de `AnalogIn()` functie te geven. De waarde van de spanning op de pin kan dan uitgelezen worden door `objectnaam.value` aan te roepen in de code. De analoge waarde wordt uitgedrukt in een 16-bit digitaal getal. Een 16-bit digitaal getal kan 65536 waarden representeren, de waarden 0-65535. De waarde 0 wordt uitgelezen als een spanning van  $0\text{ V}$  op de pin wordt gezet en de waarde 65535 als een spanning van  $3.3\text{ V}$  op de pin wordt gezet. Om het digitale getal te converteren naar de analoge spanning wordt in de gedefinieerde `get_voltage()` functie de uitgelezen waarde gedeeld door het aantal mogelijke nummers en vermenigvuldigd met 3.3. In de while loop wordt vervolgens elke tiende seconde de waarde geprint. Open ook de **plotter** in Mu editor om de waarden in de grafiek te zien veranderen als je aan de potmeter draait.

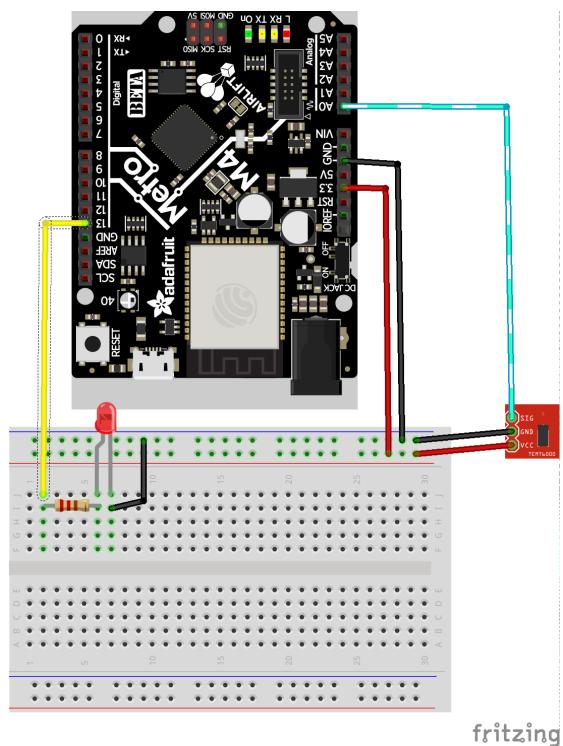
## 4.2 Een licht aan en uit zetten op basis van de licht intensiteit

**Probleem:** Je wilt een automatisch licht dat aangaat als het donker wordt.

**Benodigdheden:**

- 1x LED
- 1x  $680\Omega$  weerstand
- 1x TEMT6000 lichtsensor

**Oplossing :**



Figuur 4.2: TEMT6000 lightsensor circuit

## 4.2. EEN LICHT AAN EN UIT ZETTEN OP BASIS VAN DE LICHT INTENSITEIT 27

```
1 ## Ligthsensor-TEMT600-OnOff.py
2
3 import time
4 import board
5 import digitalio
6 from analogio import AnalogIn
7
8 #initiate pins
9 led = digitalio.DigitalInOut(board.D13)
10 led.direction = digitalio.Direction.OUTPUT
11 analog_in = AnalogIn(board.A0)
12
13 #convert 16-bit value to voltage
14 def get_voltage(pin):
15     return (pin.value * 3.3) / 65535
16
17
18 while True:
19     if analog_in.value < (63536/2):
20         led.value = True
21     else:
22         led.value = False
23     print((get_voltage(analog_in),))
24     time.sleep(0.1)
```

**Discussie:** De lichtsensor is een analoge sensor waar een spanning uitkomt die evenredig is met de lichtintensiteit. De werking van een lichtsensor is vergelijkbaar met dat van een potentiometer, beide werken met het principe van een spanningsdeler. In het geval van een lichtsensor is het alleen niet een mechanische draaiknop dat de verhouding tussen de weerstanden verandert, maar de lichtintensiteit. De code om de spanning van de lichtsensor uit te lezen is precies hetzelfde als bij de potentiometer. De analogio module wordt geïmporteerd en uitgelezen met de `get_voltage()` functie. In dit geval wordt niet alleen de waarde van de sensor uitgelezen, maar wordt ook een LED aangezet als de lichtintensiteit een sensor spanning lager dan 1.65 V geeft.

### 4.3 Een licht feller laten branden als het donkerder wordt.

**Probleem:** Je wilt de felheid van een ledje aanpassen op de intensiteit van het licht.

**Benodigdheden:**

- 1x LED
- 1x  $680\Omega$  weerstand
- 1x TEMT6000 lichtsensor

**Oplossing:** Circuit in Figuur 4.2

```

1 ## Lightsensor_TEMT6000_PWM.py
2
3 import time
4 import pulseio
5 import board
6 from analogio import AnalogIn
7
8 #initiation
9 pwm = pulseio.PWMOut(board.D13)
10 analog_in = AnalogIn(board.A0)
11
12 #Convert 16-bit value to voltage
13 def get_voltage(pin):
14     return (pin.value * 3.3) / 65535
15
16
17 while True:
18     pwm.duty_cycle=65535-(analog_in.value) #donkerder is feller
19     print((get_voltage(analog_in),))
20     time.sleep(0.1)

```

**Discussie:** Dit is een variatie op het aan en uitzetten van de lichtsensor. Hier wordt Pulse-Width Modulation gebruikt om het ledje feller en minder fel te maken op basis van de lichtintensiteit. De pulseio module wordt geïmporteerd om PWM te gebruiken. In plaats van een DigitalInOut wordt nu een PWMout object gemaakt met de naam 'pwm' die geassocieerd is met pin D13. De intensiteit van de LED kan nu geregeld worden door de waarde pwm.duty\_cycle te veranderen. Ook PWM heeft een waarde tussen 0 en 65535, net als het uitlezen van een analoog signaal. De duty\_cycle is lineair gerelateerd aan het voltage uit de pin. Een waarde van 0 zet een spanning van 0V op de I/O pin en 65535 een waarde van 3.3V. In de code wordt een spanning van 3.3V op de pwm pin gezet als een lichtintensiteit van 0V wordt gemeten, 1.65V als 1.65V wordt gemeten en 0V als een spanning van 3.3V wordt gemeten.

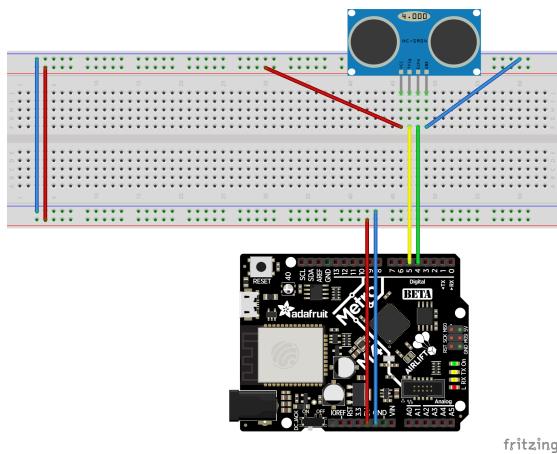
## 4.4 Ultrasonice afstandssensor

**Probleem:** Je wilt de afstand tot een object meten met een ultrasone sensor.

**Benodigdheden:**

- 1x HCSR04 Ultrasone afstandssensor

**Oplossing :**



Figuur 4.3: Ultrasonic Distance Circuit

```

1 ## UltrasonicDistance.py
2
3 import time
4 import board
5 import adafruit_hcsr04
6
7 sonar = adafruit_hcsr04.HCSR04(trigger_pin=board.D5, echo_pin=board
8 .D4)
9
10 while True:
11     try:
12         print((sonar.distance,))
13     except RuntimeError:
14         print("Retrying!")
15         time.sleep(0.1)

```

**Discussie:** Voor het gebruik van een ultrasone afstandssensor hoef je zelf weinig te programmeren, want Adafruit heeft hier een speciale bibliotheek voor in hun CircuitPython library bundel. Deze bundel kan je vinden op <https://circuitpython.org/libraries>. Zorg dat het adafruit\_hcsr04.mpy bestand in de lib folder op je bordje staat.

In de code hoef je nu alleen de adafruit\_hcsr04 te importeren en een nieuw object te maken met de HCSR04 functie, waarin je de trigger en echo pin nummers moet geven als parameters. Dit object krijgt in de code de naam sonar. De afstand kan nu eenvoudig gemeten worden door sonar.distance aan te roepen. Dit wordt in de code gedaan in een try/except block. Dit voorkomt dat de code crasht als er een runtime error optreedt bij het aanroepen van sonar.distance.

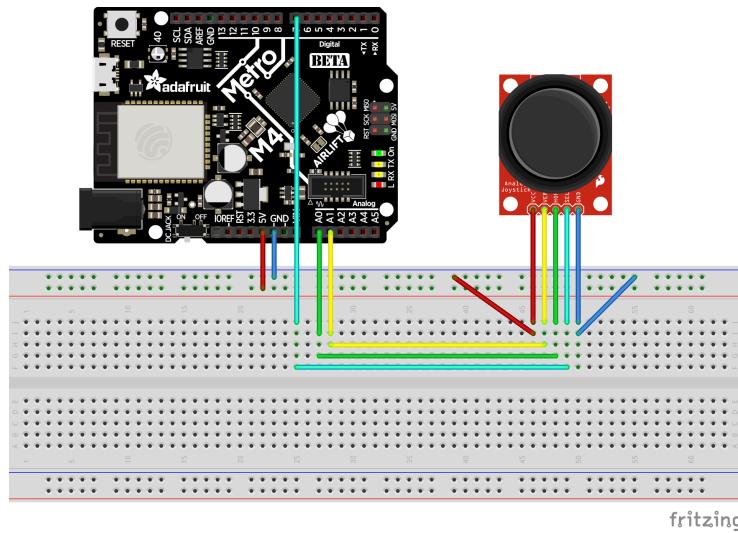
## 4.5 Joystick

**Probleem:** Je wilt een joystick uitlezen en gebruiken in een project.

**Benodigdheden:**

- 1x Joystick

**Oplossing :**



Figuur 4.4: Joystick Circuit

```

1 ## Joystick.py
2
3 import time
4 import board
5 from digitalio import DigitalInOut, Direction, Pull
6 from analogio import AnalogIn
7
8 #initialize joystick analog pins
9 LeftRight_Reading = AnalogIn(board.A1)
10 UpDown_Reading = AnalogIn(board.A0)
11
12 #initialize joystick pressdown switch
13 switch = DigitalInOut(board.D7)
14 switch.direction = Direction.INPUT
15 switch.pull = Pull.UP
16
17 #define function to readout voltage
18 def get_voltage(pin):
19     return (pin.value * 3.3) / 65535
20
21
22 while True:
23     if switch.value: #if button is not pressed
24         print((get_voltage(LeftRight_Reading),get_voltage(
25             UpDown_Reading)))
26     else:           #if button is pressed
27         print("button pressed")
time.sleep(0.1)

```

**Discussie:** Een joystick bestaat uit 2 analoge potentiometers en 1 digitale knop. De benodigde modules voor het uitlezen van de joystick zijn de board, time, digitalio en analogio module.

Links/rechts wordt uitgelezen op A1, omhoog/omlaag op A0 en het indrukken op D7. De richting van D7 wordt op input gezet en een pull-up weerstand wordt gebruikt om de pin hoog te houden zolang de knop niet wordt ingedrukt. Als geen pull-up weerstand zou worden gebruikt, dan is niet bekend wat de waarde van de input is als de knop niet wordt ingedrukt. Het kan dan gebeuren dat de uitgelezen spanning dichter bij ground zit dan bij 3.3V en wordt dan uitgelezen alsof de knop wordt ingedrukt, terwijl dit niet gebeurd. Pull-up en pull-down weerstanden worden gebruikt om deze 'zwevende' waarden een vaste waarde te geven.

In de oneindige while loop wordt elke tiende seconde eerst gekeken of de knop van de joystick is ingedrukt. Als dit niet zo is worden de 2 spanningen geprint. Als de knop wel wordt ingedrukt wordt "button pressed" geprint. Open ook de **plotter** in mu editor om de waarden te zien veranderen als je de joystick beweegt.

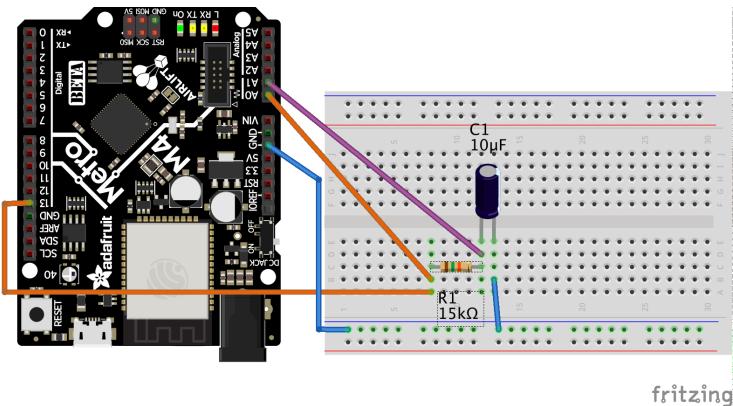
## 4.6 RC Lowpass Filter

**Probleem:** Je wilt het effect van een low-pass filter zien.

**Benodigdheden:**

- 1x  $15\text{ k}\Omega$  weerstand
- 1x  $10\text{ }\mu\text{F}$  condensator

**Oplossing :**



Figuur 4.5: RC-filter Circuit

```

1  ## PWM_Filter.py
2
3  # CircuitPython AnalogIn Demo
4  import time
5  import board
6  import pulseio
7  from analogio import AnalogIn
8
9  unfiltered_in = AnalogIn(board.A0)
10 filtered_in=AnalogIn(board.A1)
11
12 wave = pulseio.PWMOut(board.D13, frequency=1, duty_cycle=2**15)
13
14
15 def get_voltage(pin):
16     return (pin.value * 3.3) / 65536
17
18
19 while True:
20     print((get_voltage(unfiltered_in),get_voltage(filtered_in)))
21     time.sleep(1/100)

```

**Discussie:** Een Resistor Capacitor low-pass filter wordt gebruikt om hoge frequenties in signalen weg te filteren en lage frequenties door te laten. Low-pass filters worden vaak gebruikt om ruis uit sensor signalen te halen, maar ook door bijvoorbeeld dj's om hoge frequenties uit muziek te filteren. Om inzicht te krijgen in het effect van een RC-filter wordt in dit recept een PWM blokgolf gebruikt, die zowel gefilterd als ongefilterd wordt uitgelezen door een analoge input.

De time, board, pulseio en analogio modules worden eerst geïmporteerd. Er worden 2 analoge inputs geïnitialiseerd op A0 en A1 en er wordt een PWM signaal met een frequentie van 1 Hz en een duty cycle van 0.5 op pin D13 geïnitialiseerd. Één signaal gaat direct van de PWM output naar A0. Dit signaal is ongefilterd. Het andere signaal gaat eerst door de RC filter voordat het bij A1 aankomt. De outputs worden vervolgens elke honderdste seconde geprint. Open de plotter in mu editor om de verschillende signalen te zien.

# Hoofdstuk 5

## Aandrijving

De combinatie van het uitlezen van sensoren met het aandrijven van hardware vormt de basis van alle meet- en regel systemen. Dit hoofdstuk bevat recepten voor het aansturen van hardware.

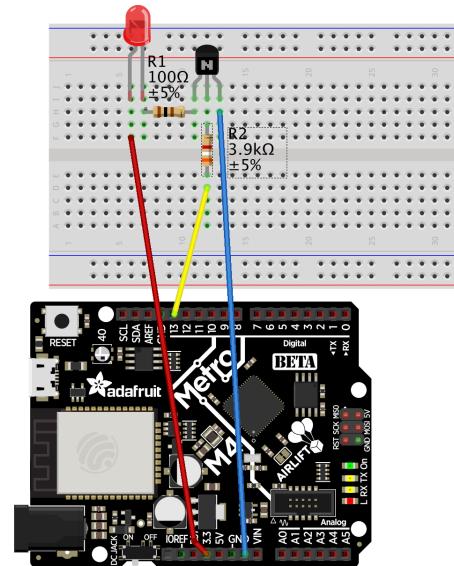
### 5.1 Transistors

**Probleem:** Je wilt een systeem aansturen dat meer power nodig heeft dan je I/O pinnen kunnen leveren, zie hoofdstuk 1.5. Door middel van een transistor kan het circuit, net als met een schakelaar, open en gesloten worden. Een I/O pin stuurt deze 'schakelaar' aan, terwijl een externe bron de stroom levert.

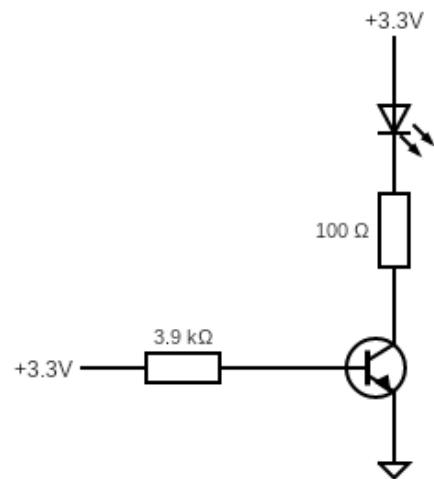
**Benodigdheden:**

- 1x LED
- 1x  $100\ \Omega$  weerstand
- 1x  $3.9\ k\Omega$  weerstand
- 1x NPN transistor (BJT)

**Oplossing :**



Figuur 5.1: Transistor Circuit



Figuur 5.2: Transistor Diagram

**Transistor Berekening:**

$$\beta \approx 25 \rightarrow 100 \quad (5.1)$$

$$V_{source} = 3.3 \text{ V} \quad (5.2)$$

$$V_{CE} = 0.1 \text{ V} \quad (5.3)$$

$$V_{BE} = 0.8 \text{ V} \quad (5.4)$$

$$V_{LED} = 2 \text{ V} \quad (5.5)$$

$$I_{C_{Verzadigd}} = 12 \text{ mA} \quad (5.6)$$

$$I_B > \frac{I_{C_{SAT}}}{25} = 0.48 \text{ mA} \quad (5.7)$$

$$R_C \approx \frac{V_{source} - V_{CE} - V_{LED}}{I_{C_{SAT}}} \approx 100 \Omega \quad (5.8)$$

$$R_B < \frac{V_{source} - V_{BE}}{I_B} < 5.2 \text{ k}\Omega \quad (5.9)$$

**Discussie:** Transistors zijn op verschillende manieren te gebruiken. Als versterker of als schakelaar. In dit voorbeeld wordt de transistor als schakelaar gebruikt. Eerst moet bepaald worden hoeveel stroom er door de LED moet. Maximale stroom door een LED is vaak rond de 20 mA. In dit voorbeeld wordt een veilige stroom van 12 mA genomen. Op basis van de Collector-Emitter spanningsval(te vinden in de transistor datasheet), de spanningsval over de LED en de bronspanning kan de weerstand in serie met de LED berekend worden. Met deze weerstand is de 12 mA de maximale stroom die van de collector naar de emitter kan stromen. Dit is dan ook de stroom die loopt als de transistor verzadigd is. Om de transistor als schakelaar te gebruiken en niet als versterker, moet er voor gezorgd worden dat de Base-Emitter stroom hoog genoeg is om de transistor te verzadigen. Dit is afhankelijk van de verzadigde stroom(12 mA) en de *gain* of versterking  $\beta$ .  $\beta$  is te vinden in de data sheet en ligt vaak tussen de 25 en 100. Om er zeker van te zijn dat verzadiging bereikt wordt, wordt met een lage versterking van 25 de benodigde base-emitter stroom bepaald. De benodigde base-emitter stroom kan vervolgens samen met de bronspanning en base-emitter spanningsval gebruikt worden om de weerstand aan de base te berekenen.

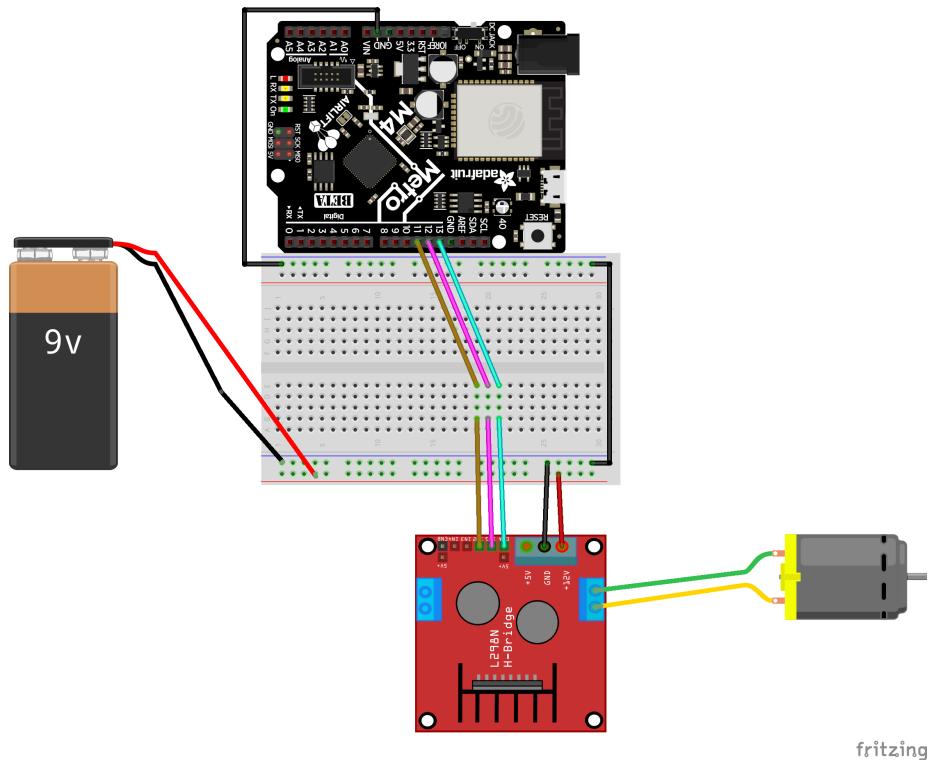
## 5.2 DC Motor

**Probleem:** Je wilt een DC aansturen met je micro-controller, dat zowel vooruit als achteruit kan draaien.

**Benodigdheden:**

- 1x 9V batterij
- 1x L298N H-Bridge Motor Driver
- 1x DC Motor

**Oplossing :**



Figuur 5.3: MotorDriver Circuit

```

1  ## DC_Motor.py
2
3  import board
4  import pulseio
5  import time
6  import digitalio
7
8  #Pin definitions
9  ENA_pin = board.D13
10 IN1_pin = board.D12
11 IN2_pin = board.D11
12
13 #Pin Initiation
14 ENA = pulseio.PWMOut(ENA_pin)
15 IN1 = digitalio.DigitalInOut(IN1_pin)
16 IN1.direction = digitalio.Direction.OUTPUT
17 IN2 = digitalio.DigitalInOut(IN2_pin)
18 IN2.direction = digitalio.Direction.OUTPUT
19
20 # Function definitions
21 def forward(DutyC):
22     ENA.duty_cycle = 0
23     IN1.value = True
24     IN2.value = False
25     ENA.duty_cycle = int(DutyC*(2**16-1))
26
27 def backward(DutyC):
28     ENA.duty_cycle = 0
29     IN1.value = False
30     IN2.value = True
31     ENA.duty_cycle = int(DutyC*(2**16-1))
32
33 while True:
34     forward(0.5)
35     time.sleep(0.5)
36     backward(0.2)
37     time.sleep(0.5)

```

**Discussie:** Het bordje kan zelf niet het vermogen leveren voor de motor en met alleen een positieve spanning kan de motor normaal gesproken maar in 1 richting draaien. Met het gebruik van een H-brug kan zowel een externe spanningsbron gebruikt worden als in 2 richtingen gedraaid worden. Met de IN1 en IN2 pinnen op de H-brug kan de richting ingesteld worden door op 1 pin een hoge spanning te zetten en op de ander een lage spanning. Met de ENA pin wordt de motor aan en uit gezet. Door hier een blok golf op te zetten met PWM kan ook de snelheid geregeld worden door de duty cycle aan te passen.

In de code worden zoals gewoonlijk eerst de benodigde modules geimporteerd. Daarna wordt gedefinieerd welke I/O pinnen zijn aangesloten op de pinnen van de H-brug. De ENA pin wordt geinitialiseerd als PWM, en de IN1 en IN2 als digitale outputs. Er worden ook functies gedefinieerd om de motor vooruit en achteruit te laten draaien, met als input de duty cycle. Allebei de functies werken hetzelfde. Eerst wordt de motor uitgezet, dan wordt de richting

bepaald door de IN1 en IN2 pinnen en dan wordt de motor aan gezet met de juiste duty cycle.

De oneindige while loop laat de motor een halve seconde vooruit draaien en daarna een halve seconde achteruit.

### 5.3 Twee DC motoren

**Probleem:** Je wilt twee DC motoren tegelijkertijd aansturen.

**Benodigdheden:**

- 1x 9V batterij
- 1x L298N H-Bridge Motor Driver
- 2x DC Motor

**Oplossing :**

```

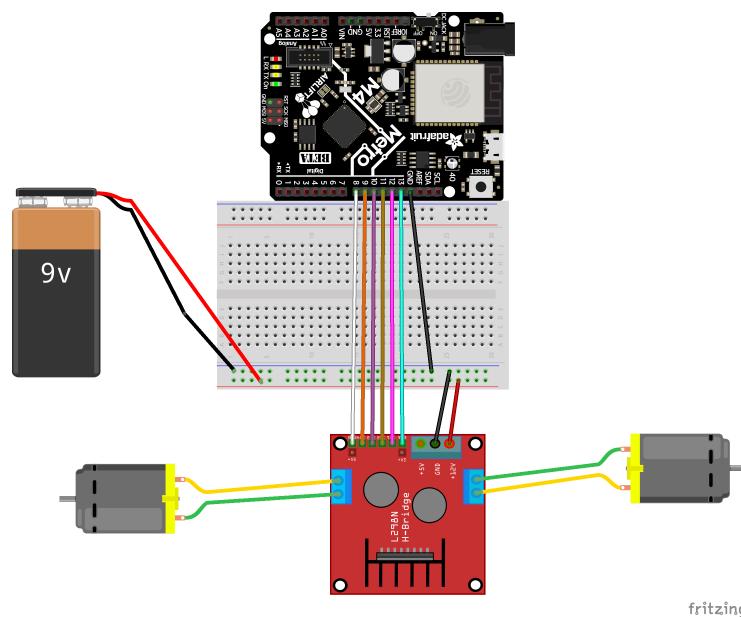
1 ## Two_DC_Motor.py
2
3 import board
4 import pulseio
5 import time
6 import digitalio
7
8 #Define pins
9 ENA_pin = board.D13
10 IN1_pin = board.D12
11 IN2_pin = board.D11
12 ENB_pin = board.D8
13 IN3_pin = board.D10
14 IN4_pin = board.D9
15
16 #initialize DC motor pins
17 ENA = pulseio.PWMOut(ENA_pin)
18 IN1 = digitalio.DigitalInOut(IN1_pin)
19 IN1.direction = digitalio.Direction.OUTPUT
20 IN2 = digitalio.DigitalInOut(IN2_pin)
21 IN2.direction = digitalio.Direction.OUTPUT
22
23 ENB = pulseio.PWMOut(ENB_pin)
24 IN3 = digitalio.DigitalInOut(IN3_pin)
25 IN3.direction = digitalio.Direction.OUTPUT
26 IN4 = digitalio.DigitalInOut(IN4_pin)
27 IN4.direction = digitalio.Direction.OUTPUT
28
29
30 def DC1_forward(DutyC):
31     ENA.duty_cycle = 0
32     IN1.value = True
33     IN2.value = False
34     ENA.duty_cycle = int(DutyC*(2**16-1))
35

```

```

36 | def DC1_backward(DutyC):
37 |     ENA.duty_cycle = 0
38 |     IN1.value = False
39 |     IN2.value = True
40 |     ENA.duty_cycle = int(DutyC*(2**16-1))
41 |
42 | def DC2_forward(DutyC):
43 |     ENB.duty_cycle = 0
44 |     IN3.value = True
45 |     IN4.value = False
46 |     ENB.duty_cycle = int(DutyC*(2**16-1))
47 |
48 | def DC2.backward(DutyC):
49 |     ENB.duty_cycle = 0
50 |     IN3.value = False
51 |     IN4.value = True
52 |     ENB.duty_cycle = int(DutyC*(2**16-1))
53 |
54 | while True:
55 |     DC1_forward(0.5)
56 |     DC2.backward(0.3)
57 |     time.sleep(0.5)
58 |
59 |     DC1_backward(0.2)
60 |     DC2.forward(0.5)
61 |     time.sleep(0.5)

```



Figuur 5.4: Double Motor Driver Circuit

**Discussie:** De code is haast hetzelfde als met 1 DC motor in recept 5.2, maar alles dubbel gedefinieerd.

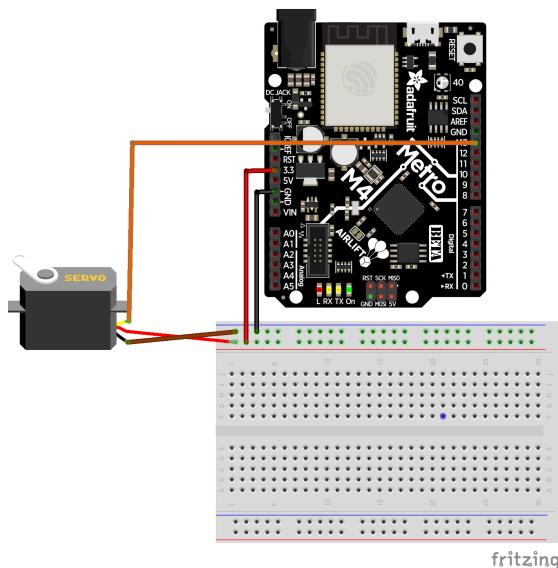
## 5.4 Servo

**Probleem:** Je wilt een Servo aansturen.

**Benodigdheden:**

- 1x Servo

**Oplossing :**



Figuur 5.5: Servo Circuit

```

1  ## Servo.py
2
3  import time
4  import board
5  import pulseio
6  from adafruit_motor import servo
7
8  # create a PWMOut object on Pin D13.
9  pwm = pulseio.PWMOut(board.D13, duty_cycle=2 ** 15, frequency=50)
10
11 # Create a servo object, my-servo.
12 my_servo = servo.Servo(pwm)
13
14 while True:
15     for angle in range(0, 180, 5): # 0 - 180 degrees, 5 degrees at
16         # a time.
17         my_servo.angle = angle
18         time.sleep(0.01)
19         time.sleep(0.5)
20     for angle in range(180, 0, -5): # 180 - 0 degrees, 5 degrees at
21         # a time.
22         my_servo.angle = angle
23         time.sleep(0.01)
24         time.sleep(0.5)

```

**Discussie:** Voor het aansturen van een servo hoef je zelf weinig te programmeren, want adafruit heeft hier een speciale bibliotheek voor in hun CircuitPython library bundel. Deze bundel kan je vinden op <https://circuitpython.org/libraries>. Zorg dat de adafruit\_motor map in de lib folder op je bordje staat.

Importeer de servo module van de adafruit\_motor bibliotheek en ook de time, board en pulsio modules. Een servo object kan eenvoudig gemaakt worden door een pwm object aan de Servo() functie van de servo module te geven. Daarna kan de hoek van de servo ingesteld worden door objectnaam.angle van het servo object aan te passen. In de code gaat de servo van 0 tot 180 graden en terug.

## 5.5. TWEE DC MOTOREN, EEN SERVO EN EEN TOETER AANSTUREN MET EEN JOYSTICK45

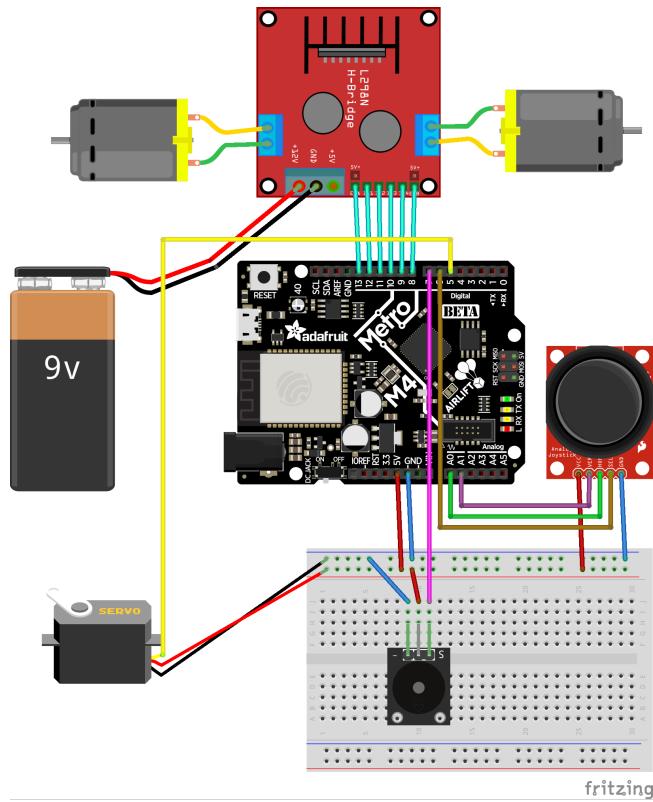
### 5.5 Twee DC motoren, een servo en een toeter aansturen met een Joystick

**Probleem:** Je wilt twee DC motoren tegelijk besturen afhankelijk van hoe ver de joystick naar voren staat. Je wilt tegelijkertijd een servo besturen afhankelijk van hoe ver de joystick naar links en naar rechts staat. Tot slot wil je de toeter aanzetten als je op de joystick drukt.

**Benodigdheden:**

- 1x 9V batterij
- 1x L298N H-Bridge Motor Driver
- 2x DC Motor
- 1x Servo
- 1x  $100\Omega$  weerstand
- 1x  $8\Omega$  Speaker

**Oplossing :**



Figuur 5.6: Twee DC motoren, een servo en een toeter aangestuurd door een joystick circuit

```

1 # Two_DC_Motors_Servo_Joystick.py
2
3 import board
4 import pulseio
5 import time
6 import digitalio
7 from analogio import AnalogIn
8 from adafruit_motor import servo
9 import simpleio
10
11 # Pin definitions
12 # DC motor pins
13 ENA_pin = board.D13
14 IN1_pin = board.D12
15 IN2_pin = board.D11
16 ENB_pin = board.D8
17 IN3_pin = board.D10
18 IN4_pin = board.D9
19
20 # Servo pin
21 servo_pin = board.D5

```

## 5.5. TWEE DC MOTOREN, EEN SERVO EN EEN TOETER AANSTUREN MET EEN JOYSTICK47

```

22 # Joystick pins
23 joy_LeftRight_pin = board.A1
24 joyUpDown_pin = board.A0
25 joy_button_pin = board.D6
26
27 # Horn pin
28 horn_pin = board.D7
29
30
31
32 # Define DC motor class
33 class DC_motor:
34     def __init__(self, EN_pin, IN1_pin, IN2_pin):
35         self.EN = pulseio.PWMOut(EN_pin)
36         self.IN1 = digitalio.DigitalInOut(IN1_pin)
37         self.IN1.direction = digitalio.Direction.OUTPUT
38         self.IN2 = digitalio.DigitalInOut(IN2_pin)
39         self.IN2.direction = digitalio.Direction.OUTPUT
40
41     def forward(self, DutyC):
42         self.EN.duty_cycle = 0
43         self.IN1.value = True
44         self.IN2.value = False
45         self.EN.duty_cycle = int(DutyC*(2**16-1))
46
47     def backward(self, DutyC):
48         self.EN.duty_cycle = 0
49         self.IN1.value = False
50         self.IN2.value = True
51         self.EN.duty_cycle = int(DutyC*(2**16-1))
52
53     def setMotor(self, DutyC):
54         if DutyC > 0:
55             self.forward(DutyC)
56         else:
57             self.backward(abs(DutyC))
58
59
60     def toDutyCycle(bitValue):
61         # map 0-3,3V van joystick naar -3,3 tot 3,3V voor motors
62         return (bitValue*2-(2**16-1))/(2**16-1)
63
64     def toAngle(bitValue):
65         # bitvalue naar servo hoek
66         return bitValue/(2**16-1)*180
67
68     def setHorn(buttonpressed):
69         # als wel ingedrukt duty_cycle 50%
70         if buttonpressed:
71             simpleio.tone(horn_pin, 400, 0.1)
72             time.sleep(0.1)
73             simpleio.tone(horn_pin, 400, 0.1)
74             time.sleep(0.1)
75
76     # Initialization
77     # Create motor objects:
78     motor1 = DC_motor(ENA_pin, IN1_pin, IN2_pin)

```

```
79 | motor2 = DC_motor(ENB_pin, IN3_pin, IN4_pin)
80 |
81 | # Create Servo object
82 | pwm = pulseio.PWMOut(servo_pin, duty_cycle=2 ** 15, frequency=50)
83 | my_servo = servo.Servo(pwm)
84 |
85 | # Initialize joystick pins
86 | joy_button = digitalio.DigitalInOut(joy_button_pin)
87 | joy_button.direction = digitalio.Direction.INPUT
88 | joy_button.pull = digitalio.Pull.UP
89 |
90 | joy_LeftRight = AnalogIn(joy_LeftRight_pin)
91 | joyUpDown = AnalogIn(joyUpDown_pin)
92 |
93 |
94 | while True:
95 |     # Read out and convert joystick values
96 |     LeftRightAngle = toAngle(joy_LeftRight.value)
97 |     UpDownDutyC = toDutyCycle(joyUpDown.value)
98 |     ButtonState = not joy_button.value
99 |
100 |     # set DC motors, servo and claxxon
101 |     my_servo.angle = LeftRightAngle
102 |     motor1.setMotor(UpDownDutyC)
103 |     motor2.setMotor(UpDownDutyC)
104 |     setHorn(ButtonState)
```

**Discussie:** Naarmate code meer gaat doen, wordt code vaak ook langer. Het wordt daarom steeds belangrijker om een goede structuur in je code aan te brengen. In deze lap code worden eerst de imports gedaan, daarna de definities, dan de initialisatie en tot slot het programma.

Vooraf een naam voor de I/O pinnen definiëren en later deze naam in de code gebruiken is erg handig. Als je namelijk de ENA pin niet op D13 maar op D2 wilt aansluiten, hoef je alleen de definitie van de ENA\_pin te veranderen en verder niet door de code te zoeken waar je allemaal D13 hebt gebruikt. Ook is het veel makkelijker om te zien of je pinnen per ongeluk dubbel gebruikt.

In het kader van het lezen en onderhouden van code kan je zien dat het besturen van een motor in een class is verwerkt. In recept 5.3 moest voor het gebruik van 2 motoren, 2 keer de code voor de initialisatie en voor elke motor een forward() en backward() functie geschreven worden. Als je nog een functie wil schrijven voor het gebruik van de motor, zoals setMotor() in deze code, moet ook dat 2x apart gemaakt worden. Er is een ongeschreven regel in programmeren die zegt 'don't repeat yourself'. Als je jezelf moet herhalen is er vaak een betere manier.

Het enige verschil tussen de eerste en tweede DC motor is welke pinnen gebruikt worden. De initialisatie en wat er moet gebeuren om de motor voor en achteruit te laten draaien is hetzelfde, alleen moet het dus op andere I/O pinnen gebeuren. Er kan dan een class gemaakt worden voor motoren die de algemene eigenschappen van een motor bij elkaar houdt. Als er een motor is, dan wordt die altijd bestuurd met 3 pinnen. Deze pinnen moeten geïnitialiseerd worden en worden altijd gebruikt om de snelheid en richting van de motor in te stellen.

Als dan objectnaam=DC\_Motor() aangeroepen wordt in de code, met als parameters de 3 pinnen, wordt een instantie gemaakt van deze class. Bij het aanmaken van een instantie wordt **altijd** eerst de `_init_` functie doorlopen. Deze initialiseert de pinnen. Elk DC\_Motor object krijgt automatisch zijn eigen forward(), backward() en setMotor() methodes. Een methode is een functie die hoort bij een class en kan je aanroepen met objectnaam.methode(). De aangemaakte motor1 en motor2 objecten kan je apart aansturen in de code. Je laat motor1 vol vooruit draaien door motor1.forward(1) aan te roepen en motor2 vol achteruit draaien door motor2.backward(1) aan te roepen. Als je een derde motor hebt kan je met 1 regel een extra motor gebruiken.

Na de definitie van de class, worden een paar functies gedefinieerd. De "toDutyCycle"functie converteert de uitgelezen 16-bit waarde van de verticale stand van de joystick naar een positieve of negatieve DutyCycle. De "toAngle"functie converteert de horizontale stand naar een hoek en de "setHorn"functie zet de toeter aan als True wordt gegeven als parameter en uit als een False wordt gegeven.

Na alle definities worden de motoren, servo, joystick en toeter objecten geïnitialiseerd. Daarna komt het programma in de oneindige while loop. In

deze loop worden de joystick waarden eerst uitgelezen en geconverteerd naar de betekenis van de waarde. JoyStick vol naar achter betekent een duty cycle van -1, vol naar voor +1. Joystick naar links een servo angle van 0 graden, joystick in het midden 90 graden en rechts 180 graden. Knop ingedrukt een waarde True, niet ingedrukt een waarde False. Vervolgens worden deze geconverteerde waarden gebruikt om de motoren, servo en toeter in te stellen.

# Hoofdstuk 6

## Netwerken

### 6.1 Maken van een WIFI access point

**Probleem:** Je wilt een privé netwerk creëren door het bordje te gebruiken als WIFI access point.

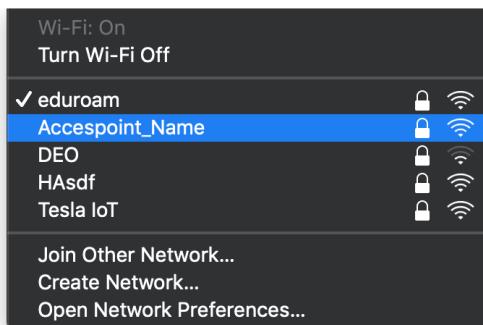
**Oplossing:**

```
1  ## WiFiAccessPoint.py
2
3  #imports
4  import board
5  import busio
6  from digitalio import DigitalInOut
7  from adafruit_esp32spi import adafruit_esp32spi
8
9
10 #setup ESP32 control
11 esp32_cs = DigitalInOut(board.ESP_CS)
12 esp32_ready = DigitalInOut(board.ESP_BUSY)
13 esp32_reset = DigitalInOut(board.ESP_RESET)
14 spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
15 esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready,
16                                         esp32_reset)
17 #Create WiFi access point
18 esp.create_AP('Accespoint_Name', 'Password')
19
20 #Do nothing
21 while True: pass
```

**Discussie:** De Metro M4 Express heeft een tweede processor op het bordje, de ESP32. De ESP32 heeft WIFI en de coprocessor wordt gebruikt om het netwerk en communicatie over het netwerk te regelen. De ESP32 kan niet direct geprogrammeerd worden en het instellen en communiceren met de coprocessor gebeurt via SPI communicatie. De details hiervan zijn onbelangrijk, omdat Adafruit een bibliotheek heeft die deze communicatie regelt. Je hoeft dus alleen

te weten hoe je gebruik maakt van deze bibliotheek. Zorg ervoor dat de adafuit\_esp32spi bibliotheek up to date is en in de lib folder staat op je bordje, zie hoofdstuk 1.4.

Als je gebruik wilt maken van de ESP32 zijn altijd dezelfde 5 regels code nodig om controle te krijgen over de ESP32. Deze stappen zijn te zien onder de comment "setup ESP32 control". De laatste stap creëert een esp object met een heleboel mogelijkheden voor het gebruik van het netwerk. Één daarvan is het opzetten van een access point. De create\_AP() methode heeft als eerste parameter de naam van het netwerk en als tweede parameter het password. "Passin de while loop zorgt dat de loop niks doet, maar het programma wel blijft draaien. Als het programma klaar is wordt het netwerk namelijk gelijk weer afgebroken. Als alles goed gegaan is, staat de naam van je gecreëerde netwerk tussen je WIFI netwerken, zoals in Figuur 6.1. Als je inlogt op dit netwerk heb je geen toegang meer tot het internet, omdat dit een privé netwerk is.



Figuur 6.1: Access point in WIFI netwerken

#### Meer informatie :

<https://circuitpython.readthedocs.io/projects/esp32spi/en/latest/api.html>

## 6.2 Data sturen van client naar server over WIFI

**Probleem:** Je wilt over het WIFI netwerk data sturen naar het bordje en dit vervolgens printen naar de terminal.

### 6.2.1 Server

**Probleem:** Je moet data kunnen ontvangen op je bordje via WIFI.

## Oplossing :

```

1  ## Minimal_Server.py
2  ##ESP32 docs at https://circuitpython.readthedocs.io/projects/
   esp32spi/en/latest/api.html
3
4  import board
5  import busio
6  from digitalio import DigitalInOut
7  from microcontroller import const
8  from adafruit_esp32spi import adafruit_esp32spi
9  import adafruit_esp32spi.adafruit_esp32spi_socket as socket
10
11
12 NO_SOCKET_AVAIL = const(255) #255 defined by the socket library as no
   socket available
13
14 #setup esp32 control
15 esp32_CS = DigitalInOut(board.ESP_CS)
16 esp32_ready = DigitalInOut(board.ESP_BUSY)
17 esp32_reset = DigitalInOut(board.ESP_RESET)
18 spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
19 esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_CS, esp32_ready,
   esp32_reset)
20
21 #Create WIFI access point
22 esp.create_AP(b'ESP32', b'JohanJohan')
23
24 #Setup server
25 port = 80
26 socket.set_interface(esp)
27 server_socket = socket.socket()
28 esp.start_server(port, server_socket.socknum)
29 ip = esp.pretty_ip(esp.ip_address)
30 print("Server available at {0}:{1}".format(ip, port))
31
32 #Initiate client socket
33 client_sock = socket.socket(socknum=NO_SOCKET_AVAIL)
34
35 while True:
36     #Wait for client connection
37     while client_sock.socknum == NO_SOCKET_AVAIL:
38         client_sock_num = esp.socket_available(server_socket.
39             socknum) #wait for request, if get request pass client
               socket number
40         client_sock = socket.socket(socknum=client_sock_num)
41             #create socket object with the client socket
               number
42
43     #Once connected, receive data
44     if client_sock.connected() and client_sock.available()!= 0:
45         encodedData = client_sock.recv()
46         data = encodedData.decode()
47         print(data)
48
49     #Once disconnected reinitiate empty client socket and wait for
      connection.
50     elif not client_sock.connected():

```

```

49 |     print('Connection lost, looking for new connection ...')
50 |     client_sock = socket.socket(socknum=NO_SOCK_AVAIL)

```

**Discussie:** De code lijkt op het eerste gezicht heel complex en om het helemaal goed te begrijpen heb je kennis nodig over netwerk communicatie. Dat gaat dit project te buiten. Gelukkig is de code goed te gebruiken zonder al die kennis en is een globaal idee goed genoeg.

Om data uit te wisselen tussen het bordje en een ander apparaat zijn 2 dingen nodig. Ten eerste moeten de apparaten op hetzelfde netwerk zitten, dit wordt gedaan door met het bordje een privé netwerk te creëren en als access point te functioneren, zoals in 6.1. Ten tweede moet over dit netwerk een verbinding gemaakt worden tussen de twee apparaten, waarover ze informatie kunnen versturen. Let goed op dat er een verschil is tussen een netwerk en een verbinding. Heel het "world wide web" (internet) zit op hetzelfde netwerk, maar je hebt met je computer niet met alle websites een verbinding.

Achter de schermen draait een website op een *server*. Een server is een computer dat een programma draait dat wacht tot een ander apparaat, een *client*, een verbinding maakt. De client stuurt dan een datapakketje naar de server, waarin naar de gewenste pagina wordt gevraagd. De server ontvangt dit datapakketje, leest de aanvraag en stuurt de gewenste pagina terug naar de client.

Dit principe wordt ook gebruikt voor het communiceren tussen je bordje en je computer. Eerst wordt een in de code een WIFI netwerk gecreëerd. Daarna wordt een server gestart op de esp32 met de adafruit\_esp32spi bibliotheek. Deze server wacht tot een client verbinding maakt. Dan gaat de code naar de oneindige while loop die uit 2 stukken bestaat. Het 1e stuk bevat nog een while loop, waar de code blijft lopen tot een client verbinding maakt. Als er dan een verbinding is gaat het door naar het 2e stuk met de if/elif statement. Er wordt gekeken of er een verbinding is en of er data is gestuurd door de client. Als dat het geval is wordt de data gedecodeerd en geprint naar de terminal. De plaats waar de print statement staat is de plek waar je ook andere dingen kan doen met de data, zoals motors aansturen en data terugsturen (zie recept 6.3). Zolang de verbinding bestaat worden het eerste stuk met de while loop en het stuk met de elif statement overgeslagen en wordt alleen de if statement uitgevoerd als er data is verstuurd door de client. Anders gebeurt er niks en blijft het lopen tot er wel iets is gestuurd of de verbinding wordt verbroken. Als de client de verbinding verbreekt wordt de elif statement uitgevoerd en komt de code terug in de while loop, waar het programma wacht op een nieuwe verbinding.

### 6.2.2 Client

**Probleem:** Je wilt data sturen naar een server vanaf je pc.

**Oplossing:**

```

1  ## Minimal_Client.py
2
3  import socket
4
5  #Host name and port number of server
6  HOST = '192.168.4.1'
7  PORT = 80
8
9  #convert to host and port to address
10 addr=socket.getaddrinfo(HOST, PORT)[0][4]
11 print(socket.getaddrinfo(HOST, PORT))
12
13 #connect to address
14 s=socket.socket()
15 s.connect(addr)
16 print(s)
17
18 while True:
19     message=input("Type a message (-q to quit): ")
20     if message=="-q": #possibility to quit the program
21         quit()
22     encodedMessage=message.encode()
23     s.sendall(encodedMessage)

```

**Discussie:** De **client** kant moet niet geupload worden op je bordje, maar moet je runnen met python op je computer. Houd er rekening mee dat de code alleen werkt als je op hetzelfde netwerk zit als je bordje. Log dus eerst in op het WIFI netwerk van je bordje.

Verbinding maken met- en data sturen naar een server kan eenvoudig met de socket module van Python's standaard bibliotheek. Je geeft de server's host adres en port nummer en met 3 regels code kan je verbinden met de server.

In de oneidige while loop wordt de gebruiker gevraagd om een bericht te typen. Als dit bericht '-q' is wordt het programma afgesloten en anders wordt het geconverteerd naar een bytarray dat kan worden verstuurd met de sendall() methode van de socket library. Als het goed is zie je de text die je stuurt verschijnen in de REPL van Mu editor.

## 6.3 Besturen van een servo met laptop

**Probleem:** Je wilt over WIFI een servo aansturen.

### 6.3.1 Server

```

1  ## RC_Servo.py
2
3  ##imports
4  import busio
5  from digitalio import DigitalInOut
6  import time
7  import board
8  import pulseio
9
10 from micropython import const
11
12 from adafruit_esp32spi import adafruit_esp32spi
13 import adafruit_esp32spi.adafruit_esp32spi_socket as socket
14 from adafruit_motor import servo
15
16
17 #constant definitions
18 NO_SOCK_AVAIL = const(255)
19 PORT = 80
20 SSID = b'RC_Servo'
21 PASSWORD = b'JohanJohan'
22 SERVO_PIN = board.D6
23
24
25 ##setup
26
27 #initiate servo
28 pwm = pulseio.PWMOut(SERVO_PIN, duty_cycle=2 ** 15, frequency=50)
29 my_servo = servo.Servo(pwm)
30 my_servo.angle = 90
31
32
33 #Obtain control over esp32
34 esp32_CS = DigitalInOut(board.ESP_CS)
35 esp32_ready = DigitalInOut(board.ESP_BUSY)
36 esp32_reset = DigitalInOut(board.ESP_RESET)
37
38 spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
39 esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_CS, esp32_ready,
40                                         esp32_reset)
41
42 #create access point
43 esp.create_AP(SSID, PASSWORD)
44
45 #Host server
46 socket.set_interface(esp)
47 server_socket = socket.socket()
48 esp.start_server(PORT, server_socket.socknum)
49 ip = esp.pretty_ip(esp.ip_address)
50 print("Server available at {0}:{1}".format(ip, PORT))
51
52 #initiate client socket
53 client_sock = socket.socket(socknum=NO_SOCK_AVAIL)
54

```

```

55 | while True:
56 |
57 |     #wait for client connection
58 |     while client_sock.socknum == NO_SOCK_AVAIL:
59 |         client_sock_num = esp.socket_available(server_socket.
60 |             socknum) #wait for request, if get request pass client
61 |             socket number
62 |             client_sock = socket.socket(socknum=client_sock_num)
63 |                 #create socket object with the client socket
64 |                 number
65 |             if client_sock.socknum != NO_SOCK_AVAIL: print('connected')
66 |
67 |             #once connected, receive data
68 |             if client_sock.connected() and client_sock.available() != 0:
69 |                 bytedata = client_sock.recv()
70 |                 data = bytedata.decode()
71 |                 try: my_servo.angle = int(data) #—————
72 |                 except: pass
73 |
74 |             #once disconnected replace client socket for empty socket and
    wait for new client.
75 |             elif not client_sock.connected():
76 |                 print('connection lost, looking for new connection... ')
77 |                 client_sock = socket.socket(socknum=NO_SOCK_AVAIL)

```

### 6.3.2 Client

```

1  ## RC_Servo_Client.py
2
3  import socket
4
5  HOST = '192.168.4.1' # The server's hostname or IP address
6      '192.168.4.1'
7  PORT = 80 # The port used by the server
8
9  addr = socket.getaddrinfo(HOST, PORT)[0][4]
9  print(socket.getaddrinfo(HOST, PORT))
10
11 s = socket.socket()
12 s.connect(addr)
13 print(s)
14
15 while True:
16     message = input("Type na angle between 0 and 180 (-q to quit):
17         ") #prompt
18     if message == "-q": #possibility to quit the program
19         break
20     encodedMessage = message.encode()
20     s.sendall(encodedMessage)

```

**Discussie:** Dit is een combinatie van het sturen van data over WIFI in recept 6.2 en het aansturen van een servo in recept 5.4. Het enige verschil aan de

*Client* kant ten opzichte van recept 6.2 is dat je een hoek moet invoeren. Aan de server kant is het verschil dat de servo geinitialiseerd wordt en dat in plaats van data te printen, nu de hoek van de servo ingesteld wordt.

## 6.4 Gebruiken van keyboard toetsaanslagen

**Probleem:** Je wilt in een programma het keyboard van je laptop gebruiken.

**Oplossing:**

```

1  ## PrintKeyStroke.py
2
3  import keyboard
4
5  def key_handler(key_event):
6      print(key_event.name, key_event.event_type)
7
8  keyboard.hook(key_handler)
9
10 while True: pass

```

**Discussie:** Voor het gebruik van je keyboard heb je de python bibliotheek "keyboard" nodig. Open de terminal op je laptop en type:

pip install keyboard

In de code wordt eerst de bibliotheek keyboard geïmporteerd. Daarna wordt de functie gedefinieerd die bepaald wat er gebeurd als je een toets indrukt. Met keyboard.hook(functienaam) koppel je het keyboard aan de functie en 'luistert' het programma op de achtergrond naar toetsaanslagen. Bij een keyboard event, zoals het indrukken of losslaten van een toets, wordt de lopende code onderbroken en wordt een eventobject doorgegeven aan de functie. Het eventobject heeft de eigenschappen "name en event\_type". De name eigenschap is de toets en event\_type zegt of de toets is ingedrukt of losgelaten. In de oneindige while loop hoeft verder niets geprogrammeerd te worden, omdat door keyboard.hook() op de achtergrond geluisterd wordt. Het zou kunnen dat je de code als administrator moet runnen. Als de code werkt, worden de toetsnaam en toetsrichting naar de terminal geprint.

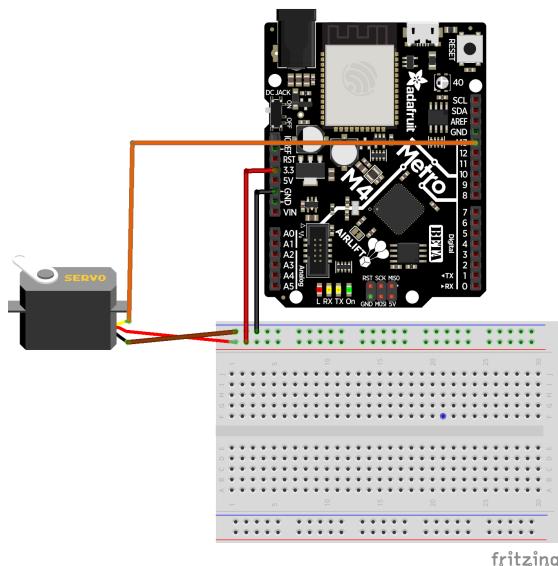
<https://pypi.org/project/keyboard/>

## 6.5 Gebruik van keyboard om een Servo aan te sturen over WIFI

**Probleem:** Je wilt over WIFI met je keyboard een servo aansturen.

**Benodigdheden:**

- 1x Servo



Figuur 6.2: Servo Circuit

### 6.5.1 Client:

**Oplossing:**

```

1  ## Keyboard_Servo_Client_v2.py
2
3  import socket
4  import time
5  import keyboard
6
7  #Start netwerk connection
8  HOST = '192.168.4.1'
9  PORT = 80
10
11 #Dictionary for coding tuples
12 PACKDICT={}
13 PACKDICT[-1]='a'
14 PACKDICT[0]='b'
15 PACKDICT[1]='c'
16

```

```

17
18
19 #functions for handling keystrokes
20 def key_handler(key_event):
21     #redirect to appropriate press or release function
22     if key_event.event_type=='down':key_press_handler(key_event)
23     elif key_event.event_type=='up':key_release_handler(key_event)
24
25 def key_press_handler(key_event):
26     if key_event.name=='left': button_state['left']=1
27     elif key_event.name=='right': button_state['right']=1
28
29 def key_release_handler(key_event):
30     if key_event.name=='left': button_state['left']=0
31     elif key_event.name=='right': button_state['right']=0
32
33
34 def sendData(data):
35     #convert to string, encode and send
36     dataString=PACK_DICT[data]
37     dataEncoded=dataString.encode()
38     s.sendall(dataEncoded)
39
40
41 def checkStateChange(keyDirection):
42     #function for checking a change in pressed buttons
43     if current_direction==keyDirection:
44         return 0
45     return 1
46
47
48 #Initialize dictionary to keep track of pressed buttons
49 button_state={
50     'left':0,
51     'right':0
52 }
53
54 #To keep track of the current direction
55 current_direction=0
56
57 #Setup connection
58 addr=socket.getaddrinfo(HOST, PORT)[0][4]
59 print(socket.getaddrinfo(HOST, PORT))
60 s=socket.socket()
61 s.connect(addr)
62 print(s)
63
64 #hook keyboard listerer to key_handler function
65 keyboard.hook(key_handler)
66
67
68 while True:
69     #check dictionary for pressed buttons and obtain direction
70     keyDirection=button_state['right']-button_state['left']
71     #check if keyDirection is different from current direction
72     stateChange=checkStateChange(keyDirection)
73

```

## 6.5. GEBRUIK VAN KEYBOARD OM EEN SERVO AAN TE STUREN OVER WIFI61

```
74 |     #if keys changed send:  
75 |     if stateChange:  
76 |         sendData(keyDirection)      #send it  
77 |         current_direction=keyDirection #change direction  
78 |         time.sleep(50/1000)
```

**Discussie:** Om een servo met een keyboard te besturen over WIFI moeten 2 dingen gebeuren. Ten eerste moet bijgehouden worden welke knoppen zijn ingedrukt en ten tweede moet deze informatie over het netwerk gestuurd worden.

Laten we eerst kijken naar het bijhouden van welke knoppen zijn ingedrukt. Een dictionary met de naam button\_state wordt geïnitialiseerd met de keys 'left' en 'right' om dit bij te houden. Om te zien hoe de dictionary gebruikt wordt, moeten je kijken naar de 3 functies om key events te regelen. De key\_handler() functie is de functie die met keyboard.hook() gebonden wordt aan het keyboard. Bij elk keyboard event wordt het programma onderbroken en wordt key\_handler() gerund. De key\_handler() functie kijkt dan of een knop is ingedrukt of losgelaten en stuurt het key\_event dan door naar de key\_press\_handler() functie als een knop is ingedrukt en naar de key\_release\_handler() als een knop is losgelaten. Deze functies kijken vervolgens om welke knop het gaat en zet in de button\_state dictionary een 1 als de knop is ingedrukt en een 0 als een knop is losgelaten.

Om deze informatie te sturen over het netwerk moet eerst een netwerk connectie gemaakt worden. Het maken van een netwerk en het sturen van data is al besproken in 6.2 en 6.3. De data moet nog wel in een goede vorm gezet worden, zodat het over het netwerk gestuurd kan worden. Dit wordt gedaan in de infinite while loop. Eerst wordt de knoprichting bepaald van button\_state. Als alleen rechts is ingedrukt krijgt keyDirection de waarde 1, als alleen links is ingedrukt -1, als rechts en links zijn ingedrukt 0 en als niks is ingedrukt ook 0.

Voordat keyDirection over het netwerk gestuurd wordt, wordt gekeken of de richting anders is dan de laatst gestuurde waarde. Alleen als de waarde is veranderd wordt de nieuwe waarde over het netwerk gestuurd. Dit wordt gedaan om het netwerk te ontlasten. Het controleren van de verandering is de taak van de checkStateChange() functie, waar keyDirection vergeleken wordt met current\_direction en een 0 of 1 terug geeft als de richting respectievelijk niet of wel is veranderd. Als de richting is veranderd gaat het programma de if statement in. Eerst wordt de waarde met de sendData() functie gestuurd. In deze functie wordt keyDirection eerst verandert in een alfabetische letter met het gebruik van de PACK\_DICT. De waarden op deze manier "coderen" maakt het een stuk eenvoudiger om de data na het sturen weer uit te pakken. Data wordt namelijk byte voor byte verzonden. Één string character is gelijk aan 1 byte. Als je de waarde -1 converteert naar de string '-1' moeten 2 bytes verstuurd worden, de - en de 1. Deze variabele lengte maakt waarden uitpakken aan de kant van

de ontvanger een stuk complexer, mede doordat meerdere waarden in de buffer kunnen staan. De waarden 0,1,0,-1,0,-1 uit de string "010-10-1" vereist meer dan alleen splitsen, omdat er soms 1 en soms 2 characters gebruikt worden voor een waarde. "bcbaba"splitsen in b,c,b,a,b,a en vervolgens omzetten met een dictionary in 0,1,0,-1,0,-1 maakt dit heel eenvoudig. Na het coderen van de waarden in de sendData() functie, wordt de character met .encode() verandert in binaire data en vervolgens verstuurd met sendall(). Tot slot wordt current\_direction geüpdatet en wacht het programma 50 ms voordat het opnieuw de while loop doorloopt.

## 6.5. GEBRUIK VAN KEYBOARD OM EEN SERVO AAN TE STUREN OVER WIFI63

### 6.5.2 Server

Oplossing:

```
1  ## Keyboard-Servo-Server-v2.py
2
3  ##imports
4  import busio
5  from digitalio import DigitalInOut
6  import time
7  import board
8  import pulseio
9
10 from micropython import const
11
12 from adafruit_esp32spi import adafruit_esp32spi
13 import adafruit_esp32spi.adafruit_esp32spi_socket as socket
14 from adafruit_motor import servo
15
16
17 #constant definitions
18 NO_SOCK_AVAIL = const(255)
19 PORT=80
20 SSID=b'ESP32',
21 PASSWORD=b'JohanJohan',
22 SERVO_PIN=board.D13
23
24
25 #Dictionary for Decoding data
26 UNPACK_DICT={}
27 UNPACK_DICT['a']=0
28 UNPACK_DICT['b']=90
29 UNPACK_DICT['c']=180
30
31
32 ##setup
33 #initiate servo
34 pwm = pulseio.PWMOut(SERVO_PIN, duty_cycle=2 ** 15, frequency=50)
35 my_servo = servo.Servo(pwm)
36 my_servo.angle=90
37
38
39 #Obtain control over esp32
40 esp32_CS = DigitalInOut(board.ESP_CS)
41 esp32_ready = DigitalInOut(board.ESP_BUSY)
42 esp32_reset = DigitalInOut(board.ESP_RESET)
43
44 spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
45 esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_CS, esp32_ready,
46                                         esp32_reset)
47
48 #create access point
49 esp.create_AP(SSID, PASSWORD)
50
51 #Host server
52 socket.set_interface(esp)
53 server_socket=socket.socket()
54 esp.start_server(PORT, server_socket.socknum)
```

```

55 ip = esp.pretty_ip(esp.ip_address)
56 print("Server available at {0}:{1}".format(ip, PORT))
57
58 #initiate client socket
59 client_sock = socket.socket(socknum=NO_SOCK_AVAIL)
60
61 while True:
62
63     #wait for client connection
64     while client_sock.socknum==NO_SOCK_AVAIL:
65         client_sock_num=esp.socket_available(server_socket.socknum)
66             #wait for request, if get request pass client socket
67             #number
68             client_sock=socket.socket(socknum=client_sock_num)
69                 #create socket object with the client socket
70                 #number
71             if client_sock.socknum!=NO_SOCK_AVAIL: print('connected')
72
73     #once connected, recieve data
74     if client_sock.connected() and client_sock.available()!=0:
75         bytedata=client_sock.recv()                      #recieve data
76         datastring=bytedata.decode()                     #convert to
77             string
78         rudderAngle=UNPACK_DICT[datastring[-1]]        #convert to
79             string
80         print(rudderAngle)
81         my_servo.angle=rudderAngle                      #set servo
82
83
84     #once disconnected replace client socket for empty socket and
85     #wait for new client.
86     elif not client_sock.connected():
87         print('connection lost, looking for new connection... ')
88         client_sock = socket.socket(socknum=NO_SOCK_AVAIL)

```

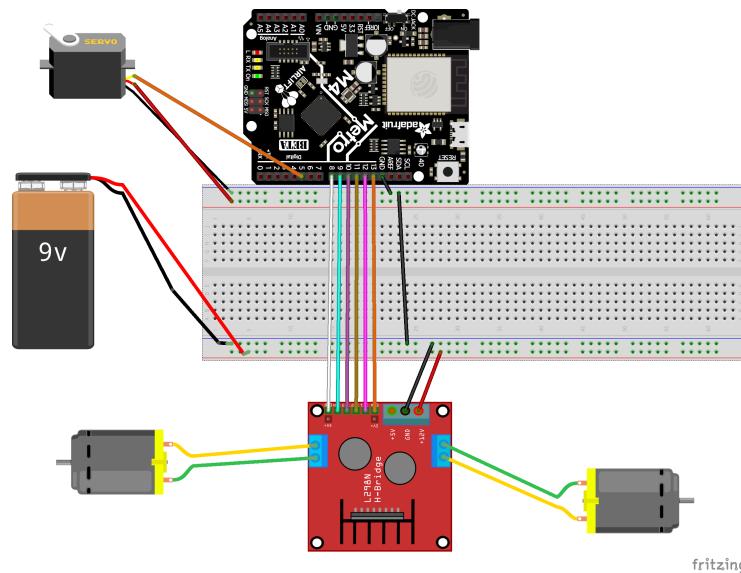
**Discussie:** De server kant is voor het grootste deel te vergelijken met 6.3. Het grootste verschil is wat voor data er binnen komt en hoe hier mee omgegaan wordt. De data wordt uitgelezen in de oneindige while loop met de recv() methode. We verwachten hier van de client kant een gecodeerde character of een string met characters, afhankelijk van hoeveel er gestuurd zijn en in de buffer staan. We zijn alleen geïnteresseerd in de laatste character, omdat deze de recentste toestand weergeeft en moeten deze destilleren van de buffer. De bytearray wordt eerst geconverteerd naar een string en vervolgens wordt de laatste character uit de string gehaald. Deze alfabetische letter wordt gedecodeerd met de UNPACK\_DICT. UNPACK\_DICT is een soort van de omgekeerde PACK\_DICT, alleen wordt a,b,c nu omgezet in de servo waarden 0, 90, 180, in plaats van de toetswaarden -1,0,1. Na het destilleren van de laatst gestuurde waarde kan gelijk de servo ingesteld worden.

## Hoofdstuk 7

# Remote Controlled boot

Alle delen om een RC boot te maken zijn nu behandeld, wat in dit hoofdstuk gebeurd. De client kant is een uitgebreide versie van 6.5.1 en de Server of boot kant is een combinatie van 5.5 en 6.5.2.

### 7.1 Circuit



Figuur 7.1: Circuit voor het aansturen van een boot

**Benodigdheden:**

- 1x Servo
- 2x DC motor
- External power source
- 1x L298N H-Bridge Motor Driver

## 7.2 Code

### 7.2.1 Computer client side

**Oplossing:**

```

1  ## RCBot_Client_v2.py
2
3  import socket
4  import time
5  import keyboard
6
7  #Host name and port number of server
8  HOST = '192.168.4.1'
9  PORT = 80
10
11 #Dictionary for coding tuples
12 PACK_DICT = {}
13 PACK_DICT[(-1,-1)] = 'a'
14 PACK_DICT[(-1,0)] = 'b'
15 PACK_DICT[(-1,1)] = 'c'
16 PACK_DICT[(0,-1)] = 'd'
17 PACK_DICT[(0,0)] = 'e'
18 PACK_DICT[(0,1)] = 'f'
19 PACK_DICT[(1,-1)] = 'g'
20 PACK_DICT[(1,0)] = 'h'
21 PACK_DICT[(1,1)] = 'i'
22
23 def key_handler(key_event):
24     #redirect to appropriate press or release function
25     if key_event.event_type == 'down': key_press_handler(key_event)
26     elif key_event.event_type == 'up': key_release_handler(key_event)
27
28
29 def key_press_handler(key_event):
30     if key_event.name == 'left': button_state['left'] = 1
31     elif key_event.name == 'right': button_state['right'] = 1
32     elif key_event.name == 'up': button_state['forward'] = 1
33     elif key_event.name == 'down': button_state['backward'] = 1
34
35 def key_release_handler(key_event):
36     if key_event.name == 'left': button_state['left'] = 0
37     elif key_event.name == 'right': button_state['right'] = 0
38     elif key_event.name == 'up': button_state['forward'] = 0
39     elif key_event.name == 'down': button_state['backward'] = 0
40

```

```

41 | def sendData(data):
42 |     #convert to string, encode and send
43 |     dataString = PACK_DICT[data]
44 |     dataEncoded = dataString.encode()
45 |     s.sendall(dataEncoded)
46 |
47 |
48 | def checkStateChange(keyDirection):
49 |     if current_direction == keyDirection:
50 |         return 0
51 |     return 1
52 |
53 |
54 | #Initialize dictionary to keep track of pressed buttons
55 | button_state = {
56 |     'left':0,
57 |     'right':0,
58 |     'forward':0,
59 |     'backward':0
60 | }
61 |
62 | #To keep track of the current direction
63 | current_direction = (0,0)
64 |
65 |
66 | #convert to host and port to address
67 | addr = socket.getaddrinfo(HOST, PORT)[0][4]
68 | print(socket.getaddrinfo(HOST, PORT))
69 |
70 | #connect to address
71 | s = socket.socket()
72 | s.connect(addr)
73 | print(s)
74 |
75 | #hook keyboard listerer to key_handler function
76 | keyboard.hook(key_handler)
77 |
78 |
79 | while True:
80 |     #check dictionary
81 |     leftRightDirection = button_state['right'] - button_state['left']
82 |     upDownDirection = button_state['forward'] - button_state['backward']
83 |     keyDirection = (leftRightDirection, upDownDirection)
84 |     #check if keyDirection is changed
85 |     stateChange = checkStateChange(keyDirection)
86 |
87 |     #if it is changed:
88 |     if stateChange:
89 |         current_direction = keyDirection      #change direction
90 |         sendData(current_direction)          #send it
91 |         time.sleep(50/1000)

```

**Discussie:** De code is eigenlijk precies hetzelfde als recept 6.5.1. Er is weer de key\_handler() functie die wordt aangeroepen bij een keyboard event. De functie

stuurt het event door naar de key\_press\_handler() als een toets is ingedrukt en naar de key\_release\_handler als een toets is losgelaten. Er is de functie sendData() voor het coderen en sturen van data en een functie checkStateChange() om te controleren of er een verandering is ten opzichte van de laatst gestuurde richting. Het enige verschil is dat de PACK\_DICT, button\_state en functies uitgebreid zijn met de üpén "down"toetsen. Als de code niet duidelijk is lees dan recept 6.5.1 nog een keer.

### 7.2.2 Boat server side

---

**Oplossing:**

```

1  ## RCBoat_Server-v2.py
2
3  ##imports
4  import busio
5  import digitalio
6  import time
7  import board
8  import pulseio
9
10 from micropython import const
11
12 from adafruit_esp32spi import adafruit_esp32spi
13 import adafruit_esp32spi.adafruit_esp32spi_socket as socket
14 from adafruit_motor import servo
15
16
17 #constant definitions
18 NO_SOCK_AVAIL = const(255)
19 PORT = 80
20 SSID = b'ESP32'
21 PASSWORD = b'JohanJohan'
22
23 ##Pin definitions
24 #DC motor pins
25 ENA_PIN = board.D13
26 IN1_PIN = board.D12
27 IN2_PIN = board.D11
28 ENB_PIN = board.D8
29 IN3_PIN = board.D10
30 IN4_PIN = board.D9
31
32 #Servo pin
33 SERVO_PIN = board.D5
34
35 #Dictionary for Decoding data
36 UNPACKDICT = {}
37 UNPACKDICT['a'] = (0,-1)
38 UNPACKDICT['b'] = (0,0)
39 UNPACKDICT['c'] = (0,1)
40 UNPACKDICT['d'] = (90,-1)
41 UNPACKDICT['e'] = (90,0)
42 UNPACKDICT['f'] = (90,1)
43 UNPACKDICT['g'] = (180,-1)
44 UNPACKDICT['h'] = (180,0)
45 UNPACKDICT['i'] = (180,1)
46
47 #Define DC motor class
48 class DC_motor:
49     def __init__(self, EN_pin, IN1_pin, IN2_pin):
50         self.EN = pulseio.PWMOut(EN_pin)
51         self.IN1 = digitalio.DigitalInOut(IN1_pin)
52         self.IN1.direction = digitalio.Direction.OUTPUT
53         self.IN2 = digitalio.DigitalInOut(IN2_pin)
54         self.IN2.direction = digitalio.Direction.OUTPUT
55

```

```

56     def forward(self , DutyC):
57         self.EN.duty_cycle = 0
58         self.IN1.value = True
59         self.IN2.value = False
60         self.EN.duty_cycle = int(DutyC*(2**16-1))
61
62     def backward(self , DutyC):
63         self.EN.duty_cycle = 0
64         self.IN1.value = False
65         self.IN2.value = True
66         self.EN.duty_cycle = int(DutyC*(2**16-1))
67
68     def setMotor(self , DutyC):
69         if DutyC>0:
70             self.forward(DutyC)
71         else:
72             self.backward(abs(DutyC))
73
74
75 ##setup
76 #initialize servo
77 pwm = pulseio.PWMOut(SERVO_PIN, duty_cycle=2 ** 15, frequency=50)
78 my_servo = servo.Servo(pwm)
79 my_servo.angle=90
80
81 #initialize motors
82 motor1 = DC_motor(ENA_PIN, IN1_PIN ,IN2_PIN )
83 motor2 = DC_motor(ENB_PIN, IN3_PIN ,IN4_PIN )
84
85 #Obtain control over esp32
86 esp32_cs = digitalio.DigitalInOut(board.ESP_CS)
87 esp32_ready = digitalio.DigitalInOut(board.ESP_BUSY)
88 esp32_reset = digitalio.DigitalInOut(board.ESP_RESET)
89
90 spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
91 esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready ,
92                                         esp32_reset)
93
94 #create access point
95 esp.create_AP(SSID, PASSWORD)
96
97 #Host server
98 socket.set_interface(esp)
99 server_socket = socket.socket()
100 esp.start_server(PORT, server_socket.socknum)
101 ip = esp.pretty_ip(esp.ip_address)
102 print("Server available at {0}:{1}".format(ip , PORT))
103
104 #initiate client socket
105 client_sock = socket.socket(socknum=NO_SOCK_AVAIL)
106
107 while True:
108     #wait for client connection
109     while client_sock.socknum == NO_SOCK_AVAIL:
110         client_sock_num = esp.socket_available(server_socket .
                                          socknum)

```

```

111     client_sock = socket.socket(socknum = client_sock_num)
112     if client_sock.socknum != NO_SOCK_AVAIL: print('connected')
113
114     #once connected, recieve data
115     if client_sock.connected() and client_sock.available()!= 0:
116         byteData = client_sock.recv()                      #recieve data
117         dataString = byteData.decode()                     #decode
118         dataTuple = UNPACK_DICT[dataString[-1]] #last char to tuple
119         print(dataTuple)
120         (rudderAngle, motorvalue) = dataTuple    #unpack tuple
121
122     #set motors
123     my_servo.angle = rudderAngle
124     motor1.setMotor(motorvalue)
125     motor2.setMotor(motorvalue)
126
127
128
129     # once disconnected replace client socket for
130     # empty socket and wait for new client.
131     elif not client_sock.connected():
132         print('connection lost, looking for new connection... ')
133         client_sock = socket.socket(socknum = NO_SOCK_AVAIL)

```

**Discussie:** Deze code is een combinatie van recept 5.5 en recept 6.5.2. Eerst worden imports en definities gedaan. Daarna wordt de DC\_motor() class gedefinieerd die is hergebruikt van recept 5.5. Na de definities worden de servo, dc motoren en het netwerk geïnitialiseerd en gaat het programma de oneindige while loop in. Zodra het een connectie heeft leest het de gestuurde data uit, converteert de data naar een string, haalt de laatst gestuurde character uit de string en zoekt de bijbehorende tuple op in de dictionary. Vervolgens wordt de tuple en uitgepakt in een rudderAngle en een motorvalue die tot slot worden gebruikt om de servo en motoren in te stellen.