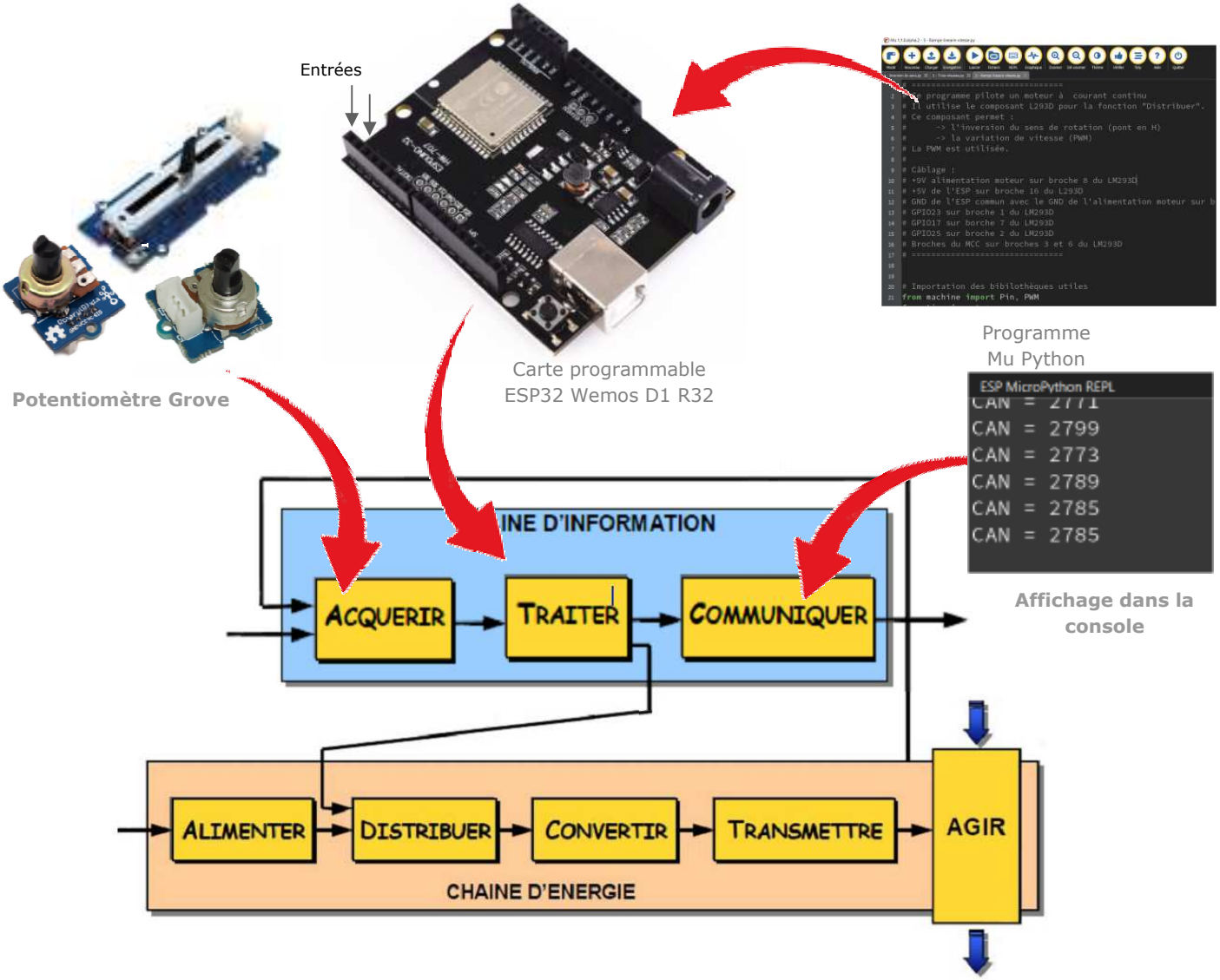
MISE EN ŒUVRE

→ **TRAITER : ESP32 WEMOS (EDI MU)**

→ **ACQUERIR : Potentiomètre Grove**

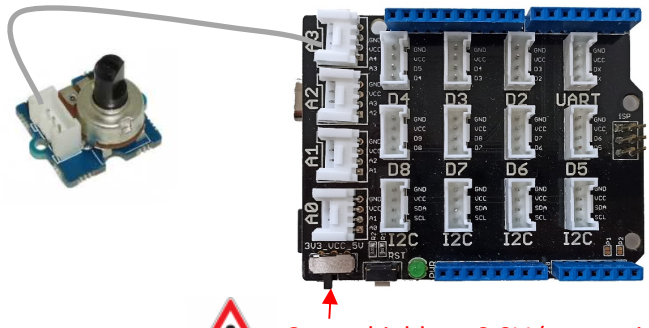
1 – Mise en situation



2 – Plan de câblage / Montage

Raccorder avec la carte ESP et son shield grove :

Potentiomètre connectée
sur le GPIO34 (A3)



Carte shield sur 3,3V (sous peine de détruire la carte ESP32)

3 – Schéma du BP et explications

Schéma

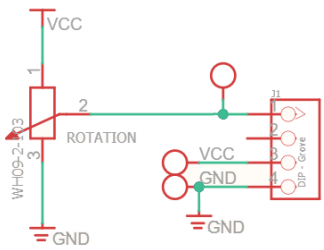
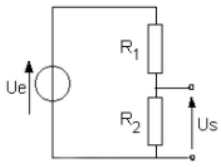


Schéma de principe



<u>Valeurs caractéristiques :</u>

Valeur du potentiomètre P = R1 + R2 = 10kΩ - Course angulaire : 300°

<u>Principe de fonctionnement :</u>

R_1 : résistance entre les bornes 1 et 2 (curseur) du potentiomètre

R_2 : résistance entre les bornes 2 (curseur) et 3 du potentiomètre

U_e : tension d'alimentation du potentiomètre : U_e = VCC (3,3V pour la carte ESP32 5V pour carte ARDUINO)

U_s : tension délivrée par le potentiomètre selon de principe du pond diviseur de tension : $U_s = U_e \cdot \frac{R_2}{R_1+R_2}$

La **tension analogique** issue du potentiomètre est convertie en une **valeur numérique** par le **CAN** de l'ESP :

L'ESP32 possède plusieurs entrées avec un **CAN de 12 bits**. La tension d'entrée entre 0 et 3.3V est convertie en une valeur numérique comprise entre **0 et 4095**. La résolution peut être changée par programmation. Sur l'ESP32, la fonctionnalité ADC est disponible sur les broches 32-39.

4 – Programme

ESP32 Micropython programme qui affiche dans la console le résultat de la conversion analogique numérique de la tension sur la broche 34 (ADC1_CH6, repère A3 shield base 1)

```
from machine import ADC, Pin
from time import *

can = ADC(Pin(34))                # crée un objet ADC sur la broche 34 (Shield Grove A3)
can.atten(ADC.ATTN_11DB)          # étendue totale : 3.3V
#can.width(ADC.WIDTH_10BIT)       # change la résolution du convertisseur à 10bits

while True:
    pot = can.read()          # conversion analogique-numérique 0-4095
    print("CAN =", pot)       # affichage sur la console REPL de la valeur numérique
    sleep_ms(100)
```

Remarque : Pour changer la résolution du convertisseur, il faut utiliser la méthode can.width(nb_bit) avec comme paramètre nb_bit :

- ADC.WIDTH_9BIT : range 0 à 511
- ADC.WIDTH_10BIT : range 0 à 1023
- ADC.WIDTH_11BIT : range 0 à 2047
- ADC.WIDTH_12BIT : range 0 à 4095

ex : can.width(ADC.WIDTH_10BIT)

Remarque : Pour changer la plage pleine échelle, il faut utiliser la méthode can.atten(attenuation) avec comme paramètre attenuation :

- ADC.ATTN_0DB : plage 0 – 1,2V
- ADC.ATTN_2_5 : plage 0 – 1,5V
- ADC.ATTN_6DB : plage 0 – 2V
- ADC.ATTN_11DB : plage 0 – 3,3V

ex : can.atten(ADC.ATTN_11DB)