Programmierung des Raspberry Pi Pico mit Micropython

Schritt-für-Schritt-Anleitung, um einige der vielen Möglichkeiten des Picos kennenzulernen.

LEDs schalten

Die interne LED

Der Pico hat eine interne LED, die folgendermaßen angesteuert werden kann:

```
import utime
from machine import Pin

#led=Pin("LED", Pin.OUT) # Für den Pico mit eingebautem WLAN
led=Pin(25, Pin.OUT) # Für den Pico ohne WLAN
led.value(1)
utime.sleep(1)
led.value(0)
```

Soll die LED unabhängig vom Programmablauf blinken, dann kann man das mit einem Timer realisieren.

```
from machine import Pin,Timer
# led=Pin("LED", Pin.OUT) # Für den Pico mit eingebautem WLAN
led=Pin(25, Pin.OUT) # Für den Pico ohne WLAN
timer = Timer()
timer.init(freq=2, mode=Timer.PERIODIC, callback=lambda t: led.toggle())
```

Soll der Timer beendet werden, so kann man das mit dem Befehl `timer.deinit()` erreicht werden.

Möchte man mehr Kontrolle haben oder komplexere Funktionen einbauen, dann geht das so:

```
import utime
from machine import Pin,Timer

# led=Pin("LED", Pin.OUT) # Für den Pico mit eingebautem WLAN
led=Pin(25, Pin.OUT) # Für den Pico ohne WLAN
led.value(1)
utime.sleep(1)
```

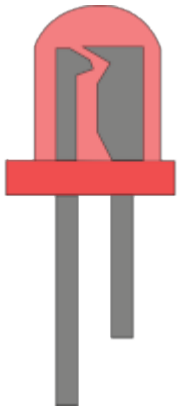
```
led.value(0)
def blink(timer):
    led.toggle()


timer = Timer()
Timer().init(freq=2, mode=Timer.PERIODIC, callback=blink)
```

Ein Timer läuft auch nach dem Ende des Programms weiter. Also nicht wundern, wenn es weiter blinkt.

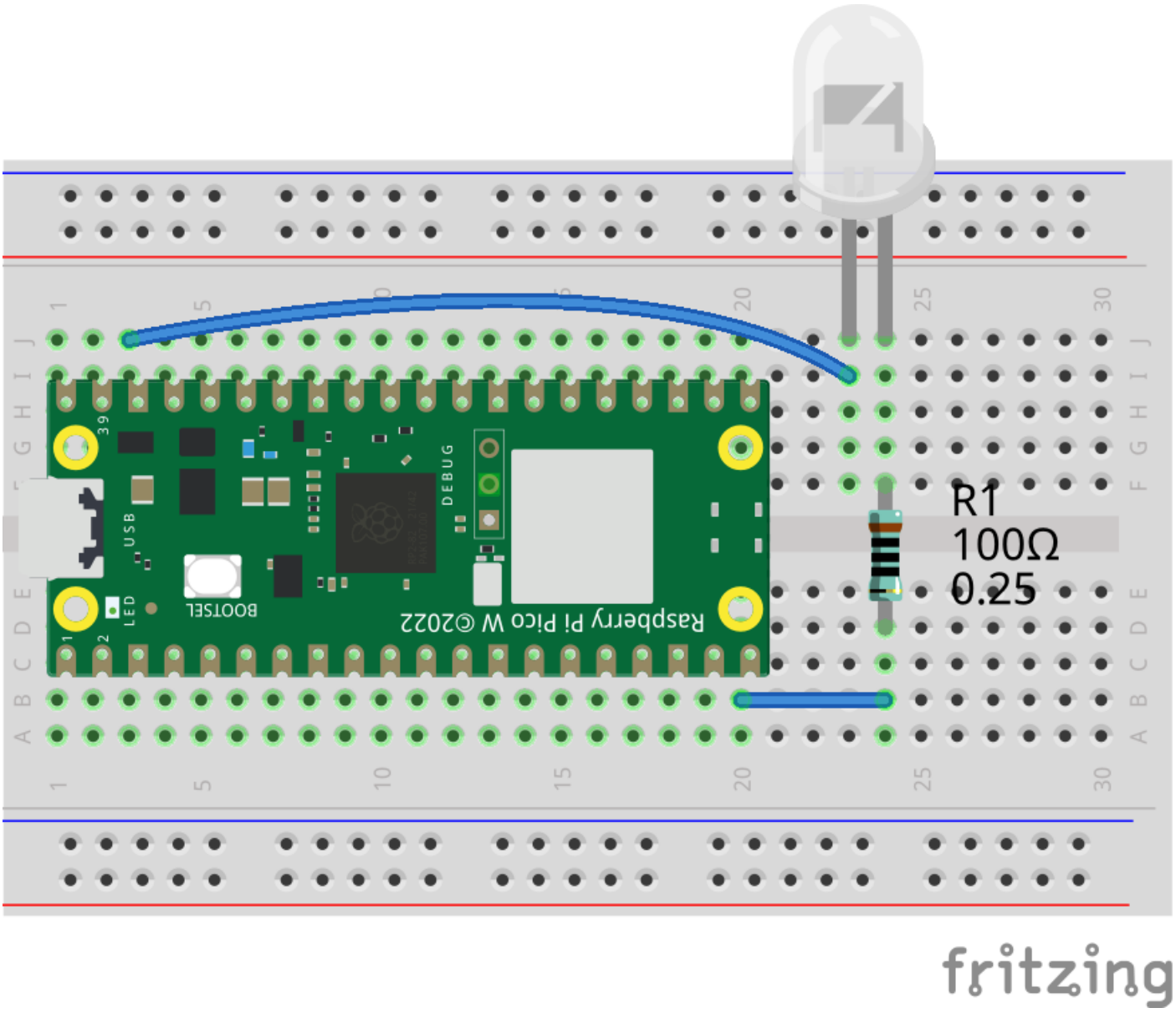
Aufgabe: Die LED zum Blinken bringen.

Eine externe LED

Das war zwar schon spannend, aber nun wollen wir eine externe LED an den Pico anschließen. Dazu benötigen wir eine LED, einen 100 Ω Widerstand sowie zwei Kabel. Stecke die Schaltung genau so zusammen, wie abgebildet. Achte vor allem darauf, dass die LED richtig herum ist.

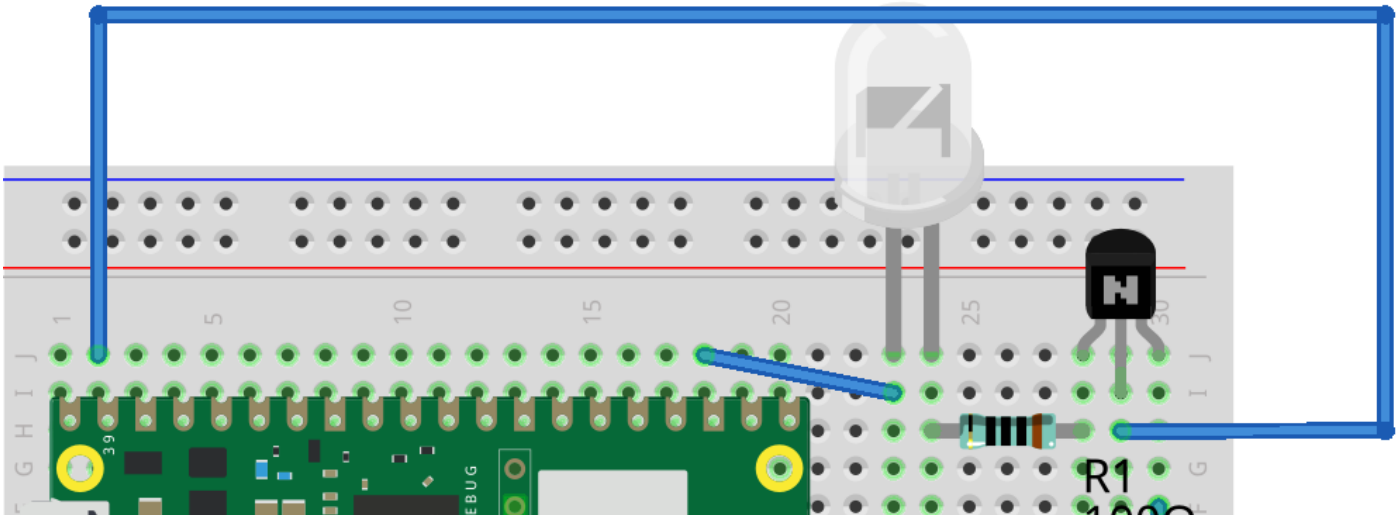


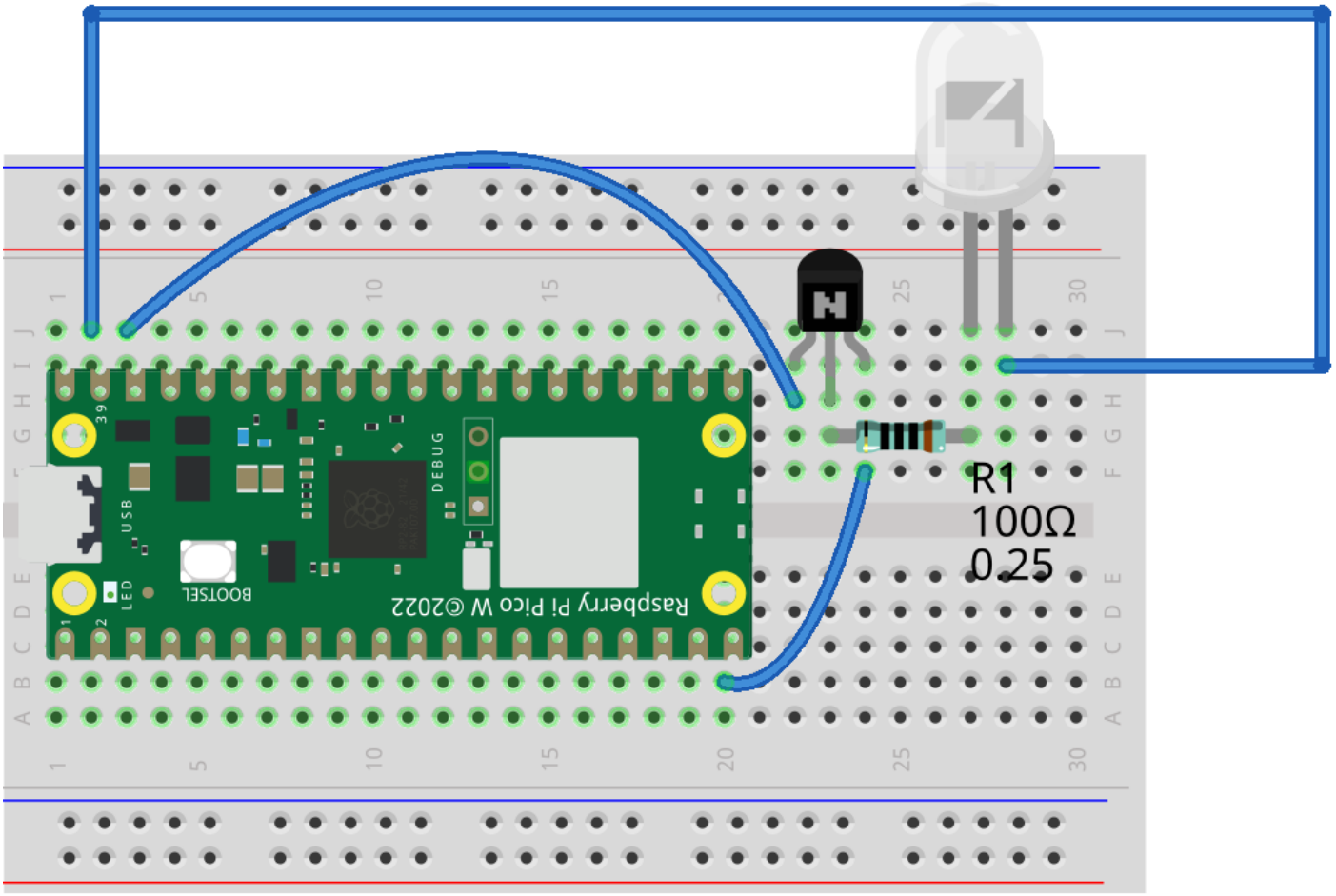
Hier ist der dazugehörige Schaltplan:



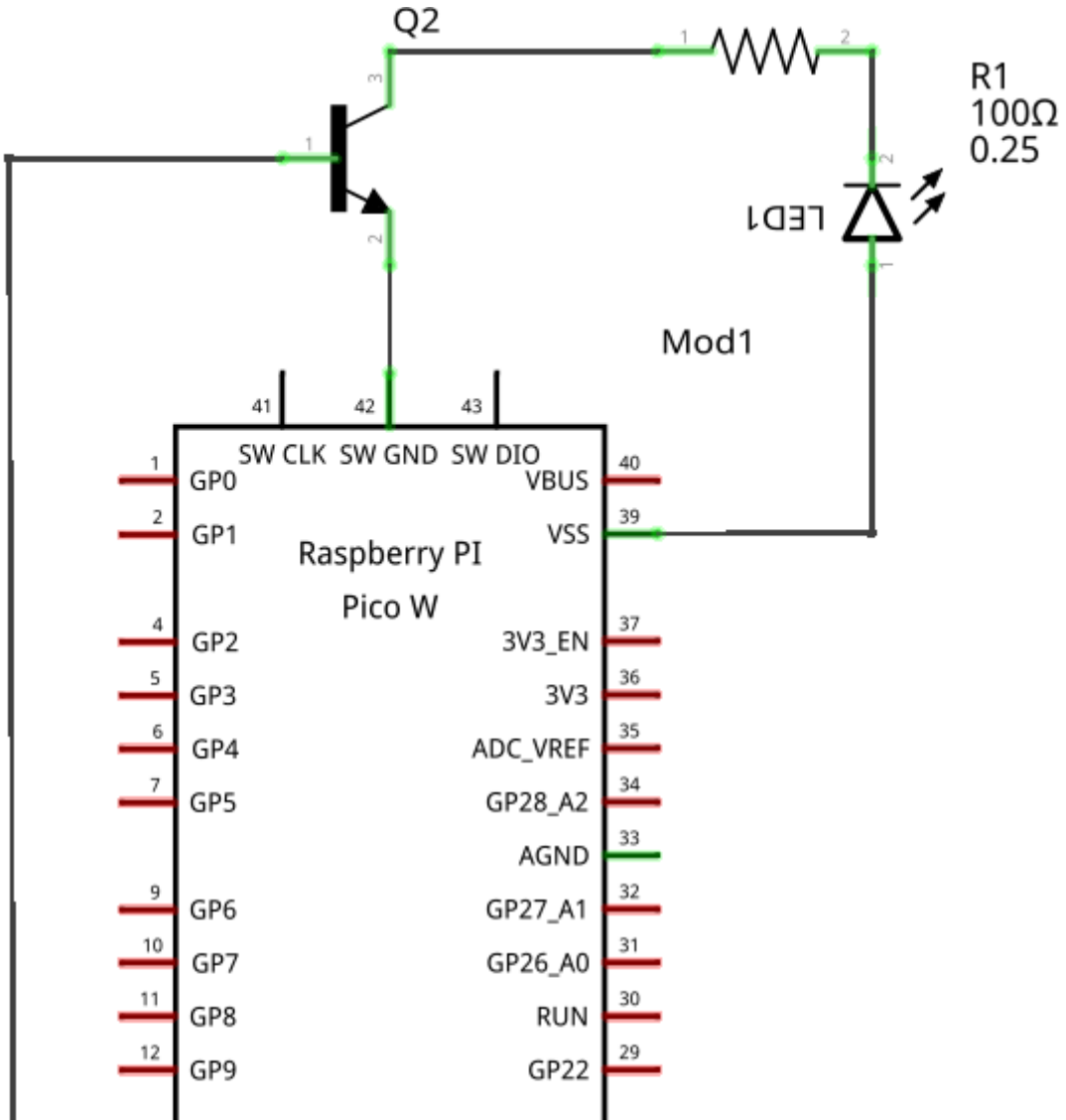
Externe LED mit Transistor

Der Pico liefert an den Pins nur eine Spannung von 3,3 Volt und die Stromstärke ist auch nicht sehr hoch. Es ist leicht möglich, die Anschlüsse zu überlasten, zwar nicht mit nur einer LED, jedoch bleibt es dabei ja nicht. Daher müssen Transistoren verwendet werden. Es gibt zwei mögliche Schaltungen für die LED mit Transistor. Der Unterschied ist nur, ob die Schaltung auf Anoden- oder auf Kathodenseite geschieht.




R1
100Ω
0.25
BOOTSEL
LED
USB
DEBUG
Raspberry Pi Pico W ©2022

fritzing


Q2
R1
100Ω
0.25
LED1
Mod1
41
42
43
SW CLK
SW GND
SW DIO
1
GP0
2
GP1
4
GP2
5
GP3
6
GP4
7
GP5
9
GP6
10
GP7
11
GP8
12
GP9
Raspberry PI
Pico W
40
VBUS
39
VSS
37
3V3_EN
36
3V3
35
ADC_VREF
34
GP28_A2
33
AGND
32
GP27_A1
31
GP26_A0
30
RUN
29
GP22

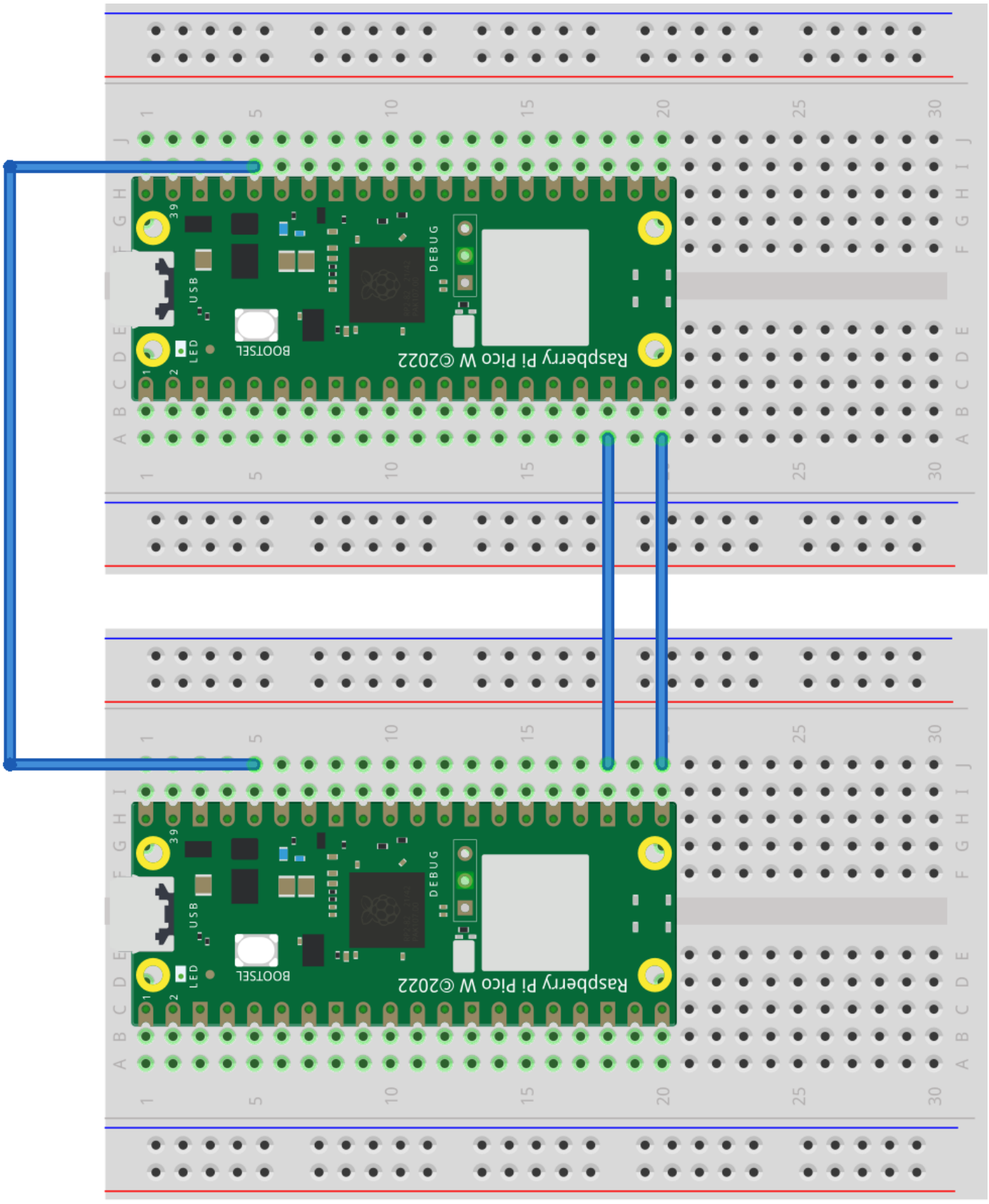
Regelung der Helligkeit

Egal, ob du die LED mit oder ohne Transistor betreibst, so ist die Helligkeit bislang immer die gleiche. Glühlampen lassen sich sehr einfach dimmen, indem man die Spannung regelt. Bei der LED funktioniert das nicht, da die Spannung an der LED immer dieselbe ist. Die LED ist nämlich kein ohmsches Bauteil. Außerdem haben wir mit dem Pico die Schwierigkeit, dass er ein digitales Gerät ist und bekanntermaßen kennen digitale Geräte nur 1 und 0 oder an und aus. Wie lässt sich damit also die Helligkeit regulieren?

Ampelschaltung mit LEDs

Ampelschaltung

Verwende die LED-Ampel, um eine Ampelschaltung zu programmieren. Schaltet dann mehrere Ampeln zu einer Kreuzung zusammen, indem ihr die Picos miteinander kommunizieren lasst. Dazu müsst ihr einen Pin auf dem Pico, der Befehle erhält, als Eingangspin definieren.


Raspberry Pi Pico W ©2022
BOOTSEL
LED
USB
DEBUG
Raspberry Pi Pico W ©2022
BOOTSEL
LED
USB
DEBUG
fritzing

Empfange Daten auf Pin 16 und blinke mit der internen LED

```
from machine import Pin


sensor=Pin(16, Pin.IN, Pin.PULL_UP)
led=Pin(25, Pin.OUT)
while True:
  while sensor.value():
    led.value(1)
  led.value(0)
```

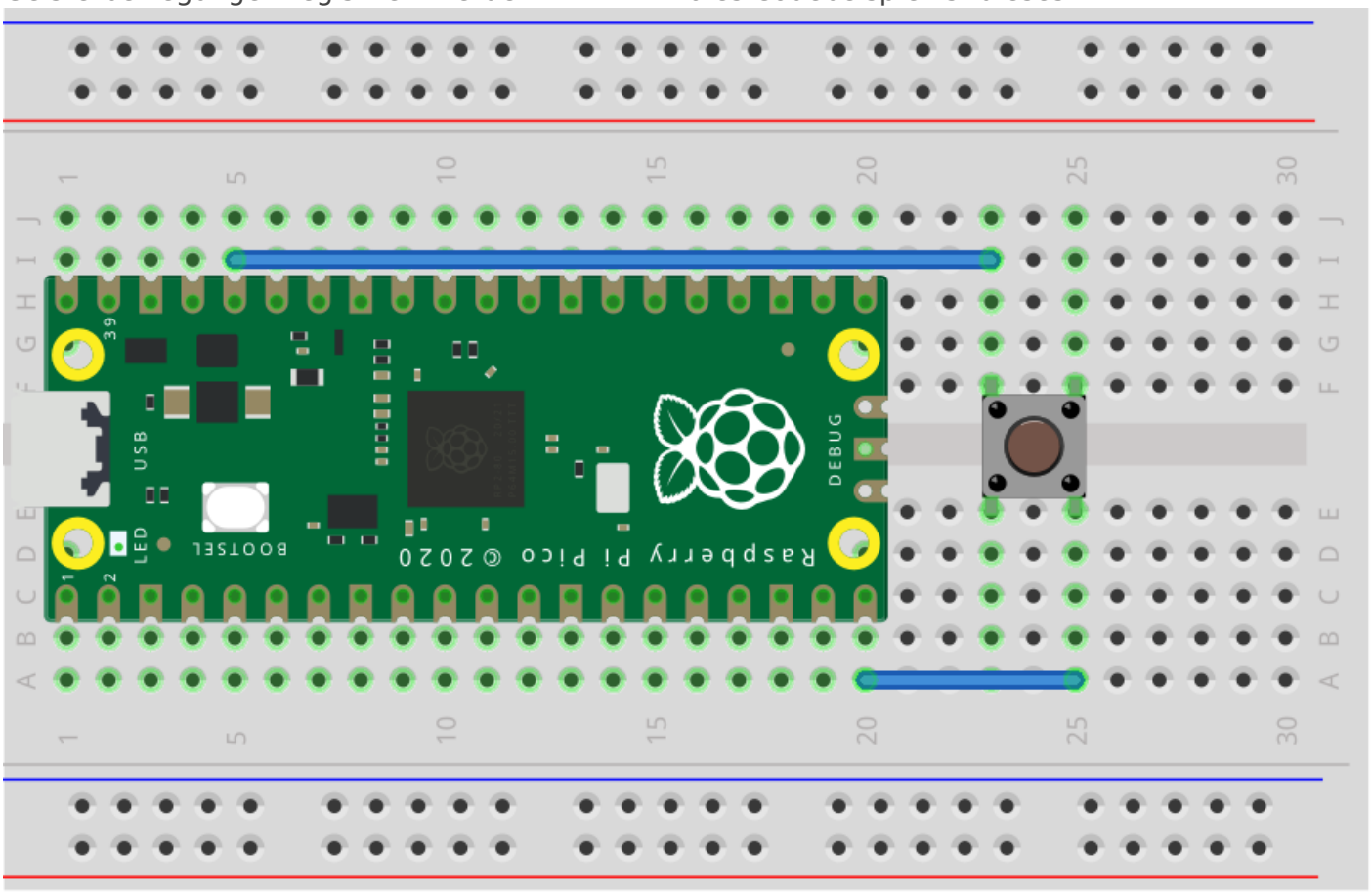
Sende Daten mit Pin 15 und blinke mit der internen LED

```
import utime
from machine import Pin


#led=Pin("LED", Pin.OUT) # Für den Pico mit eingebautem WLAN
led=Pin(25, Pin.OUT) # Für den Pico ohne WLAN
sender=Pin(15,Pin.OUT)
while True:
  led.value(1)
  sender.value(1)
  utime.sleep(1)
  led.value(0)
  sender.value(0)
  utime.sleep(1)
```

Nur einer der beiden Picos muss über USB an Strom angeschlossen werden. Beide Programm werden unter dem Namen "main.py" auf dem Pico abgespeichert, dann laufen die Programme automatisch, sobald die Picos Strom bekommen.

Knopfsteuerung der Ampel

Für die Programmierung eines Ampelknopfes, muss man den Knopf *entprellen*, damit keine Geisterbewegungen registriert werden. Ein minimales Codebeispiel ist dieses:




```
from machine import Pin
import utime
button = Pin(15, Pin.IN, Pin.PULL_DOWN)
pressed = False
num_pressed = 0
last_pressed = 0
DEBOUNCE_WAIT = 100
def button_handler(pin):
    global pressed, num_pressed, last_pressed #mit dem Befehl global teilt man Python mit, dass man die Variabel verwenden möchte, die außerhalb der Funktion initialisiert wurde.
    while utime.ticks_diff(utime.ticks_ms(), last_pressed) < DEBOUNCE_WAIT: # Hier wird verhindert,dass mehrere Auslösungen hintereinander registriert werden.
        pass
    last_pressed = utime.ticks_ms()
    if not pressed:
        while utime.ticks_diff(utime.ticks_ms(), last_pressed) < DEBOUNCE_WAIT:
            pass
```

```
        if pin.value() == 1:
            pressed=True #Damit kann im Programmablauf der Knopfdruck registriert werden.
            num_pressed +=1
            print(pin.value(), "number presses: ", num_pressed)
            last_pressed = utime.ticks_ms()
            pressed=False # Dies hier eher im weiteren Programmablauf verwenden.

button.irq(trigger=Pin.IRQ_RISING, handler=button_handler)

# Hier weiterer Programmablauf
while True:
    pass
```

Programmiergrundkurs in Python

Der Motor

Ein Motor wird genauso gesteuert wie eine LED. Man kann ihn entweder einfach an- und ausschalten, oder mithilfe der Pulsweitenmodulation die Geschwindigkeit regeln. Probiert es einfach einmal aus. Da ein Motor deutlich mehr Leistung hat als eine LED, kann man den Raspberry Pi Pico schnell überlasten. Daher betreiben wir den Motor nur über einen Transistor. Es sorgt dafür, dass das Steuersignal des Picos verstärkt von der 9V Batterie an den Motor geleitet wird. Aber Achtung: Der Motor ist für dauerhaft 6V ausgerichtet und sollte daher nicht zu lange mit 9V betrieben werden.

Für diese Schaltung könnt ihr die Schaltung der LED mit Transistor ohne den Widerstand verwenden.

Der Code für diese Steuerung ist genauso wie für eine LED. In dem Schaltbild wird der Pin 15 benutzt.

Richtungssteuerung

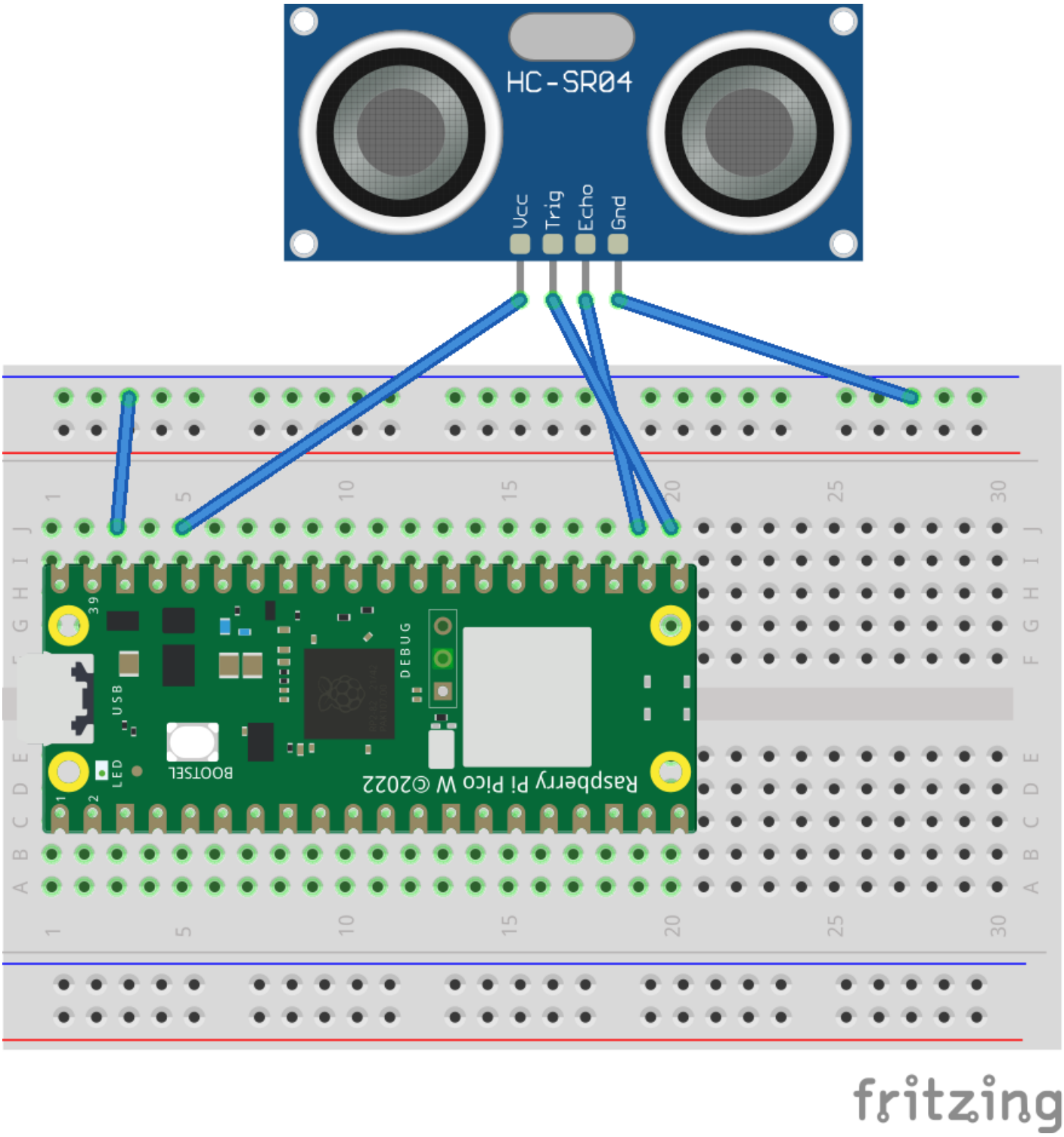
Wie könnte man nun die Drehrichtung des Motors ändern, ohne die Kabel umzustecken?

Überlegt erst einmal und dann schaut ihr euch das nächste Bauteil an:

Die H-Brücke

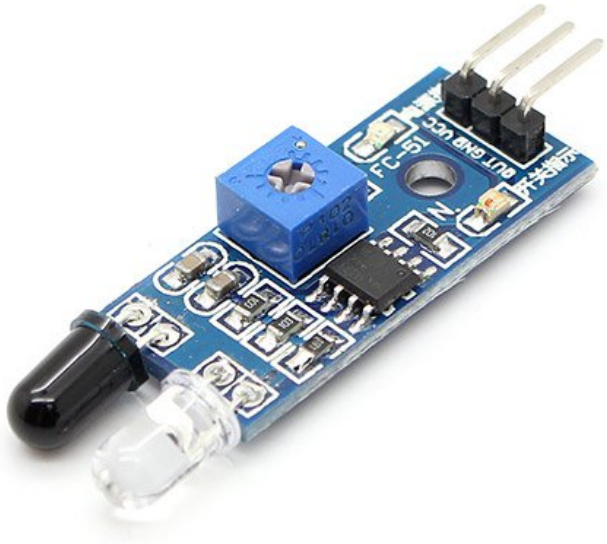
Der Ultraschallsensor

Schließt den Ultraschallsensor an den Pico an:

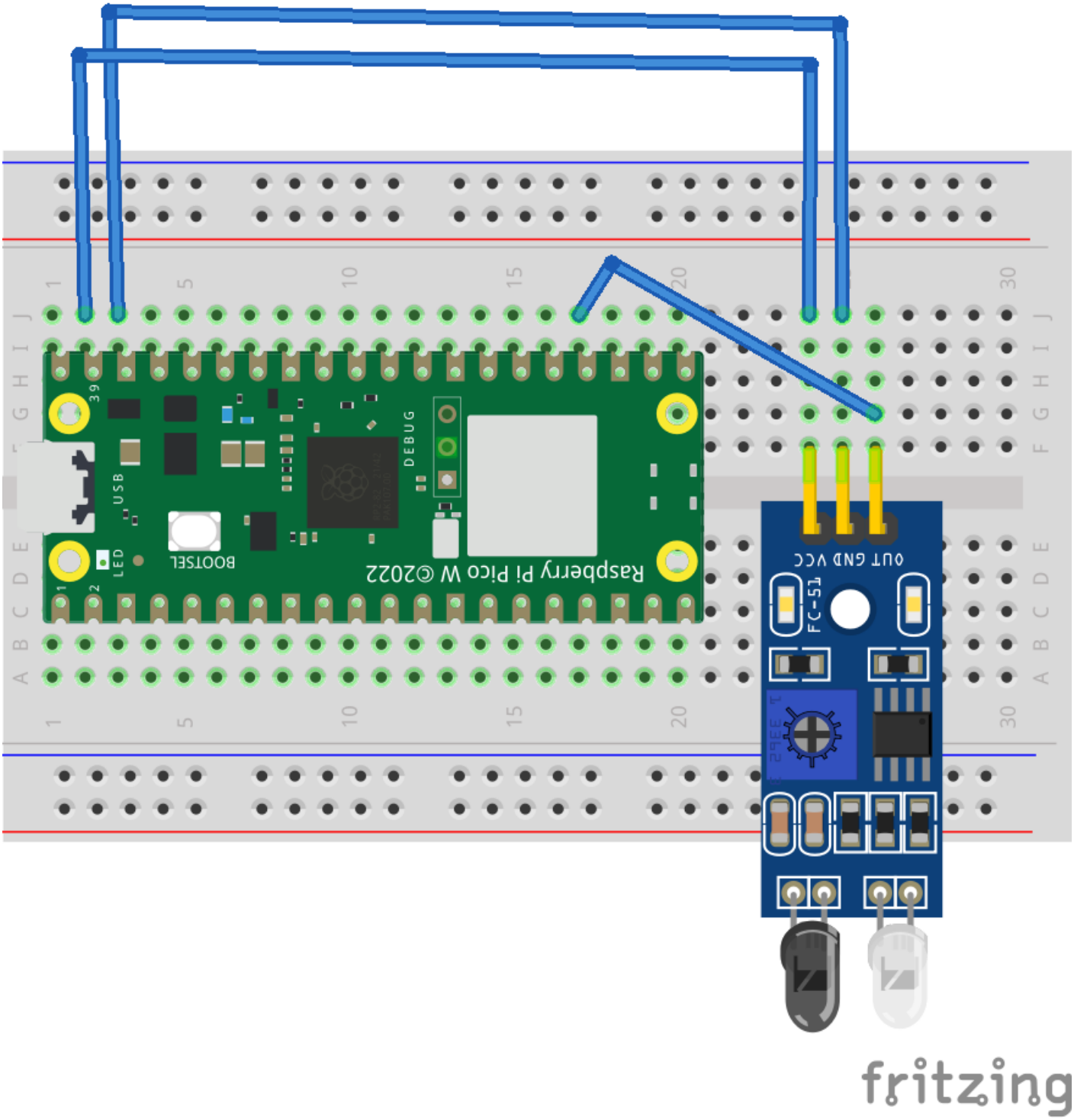


Die Softwarebibliothek für den Ultraschallsensor

Der Infrarotsensor



Der IR-Sensor wird folgendermaßen an den Pico angeschlossen. Der OUT-Pin kann natürlich auch geändert werden.



So wird der Infrarotsensor ausgelesen:

```
from machine import *
import utime

# Der Pin für den Infrarotsensor wird initialisiert.
ir=Pin(16,Pin.IN,Pin.PULL_UP)

while True:
```

```
print(ir.value()) #Es wird einmal der Wert ausgegeben, der am Pin anliegt.
while ir.value() == 0: # Solange der Wert 0 bleibt, ändert sich die Anzeige nicht.
    utime.sleep_ms(50)
print(ir.value()) # Ist/Wird der Wert 1, wird erneut auf der Konsole ausgegeben.
while ir.value() == 1:
    utime.sleep_ms(50) # Solange der Wert 1 bleibt, ändert sich die Anzeige nicht.
```

Codebeispiele

Theo III

Best Practise

Ausschalten der Motoren bei Unterbrechung des Programms

Anfangs wird man sehr viel an dem Roboterprogramm testen müssen. Dabei wird das Programm dann häufig abstürzen. Damit die Motoren nicht weiter in dem Zustand laufen, in dem sie dabei geschaltet waren, kann man mit einem try/excepts arbeiten:

```
try:
  # Hier läuft das Programm
except Exception as err:
    print(err) # Nötig, um Fehlermeldungen angezeigt zu bekommen.
    r.emergency_stop() # Roboter anhalten, hier ein Beispiel mit der robotlibrary.
    print("Robot stopped") # Damit es ganz deutlich ist.
except KeyboardInterrupt:
    r.emergency_stop()
    print("Keyboard interrupt")
```

Effizienter Code

Auch wenn die Rechenleistung der Picos ausreichen sollte, ergibt es Sinn, sich über Effizienz Gedanken zu machen, da schwer zu erkennen ist, ob manche Probleme durch Überlastung des Prozessors hervorgerufen werden.

```
while True:
    robot.drive()
    if us.get_dist() > min_distance:
        # stop or turn
        robot.stop()
```

In diesem Beispiel wird in der Schleife der Befehl `drive()` mit jedem Durchlauf aufgerufen, was nicht sonderlich effizient ist, da die Motoren weiterfahren, auch wenn das Programm gerade andere Befehle ausführt. Eine bessere Variante wäre diese:

```
robot.drive()
while us.get_dist() > min_distance:
    pass
robot.turn()
```

Hier wird nur die Entfernung zum nächsten Hindernis überprüft. Sobald der Roboter zu nahe gekommen ist, wird die Schleife beendet und der Code wird weiter ausgeführt.

Fehlertoleranter Code

Die Sensoren, die wir benutzen, liefern nicht immer zuverlässige und korrekte Ergebnisse. Daher kann man sich nicht darauf verlassen, dass eine Messung ausreicht. Ist man auf genauere Ergebnisse angewiesen, kann es sinnvoll sein, die Ergebnisse von Sensormessungen (insbesondere des Ultraschallsensors) zu filtern. Dazu kann gehören, Extremwerte, die im vorliegenden Fall unwahrscheinlich sind, zu ignorieren oder Mittelwerte von mehreren Messungen zu bilden.

Beschleunigung mit Entfernungsmessung

```
obstacle_detected = False
new_speed = 100
speed_now = 0
min_distance = 15
while speed_now <= new_speed and not obstacle_detected:
    #Set the speed for the motors, f. ex. motor.set_speed(speed_now)
    utime.sleep_ms(10+int(speed_now/2))
    speed_now += 1
    if us.get_dist() < min_distance: # Adjust the code to your needs.
        obstacle_detected = True
if obstacle_detected:
    # Stop or turn or whatever
    obstacle_detected = False
else:
    # keep going
    pass
```