26 点灯

キャンドルLED

デジタル出力

プログラム candle_led.py

ライブラリ －

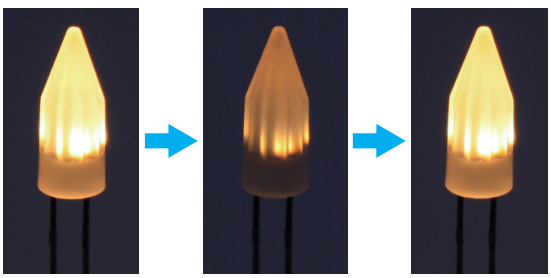
利用パーツ

●キャンドルLED×1（秋：111585） ●トランジスタ×1（秋：106477）
●抵抗器3.3kΩ×1（秋：125332）

ロウソクのように、明るくなったり暗くなったりする形でLEDを点灯したい場合、プログラムでPWMの出力をランダムに大きくしたり小さくしたりすることで実現できます。光を揺らがせるなど、よりリアルに見せたいなら、そうした工夫もプログラムに盛り込んでLEDの点灯制御をし続ける必要があり、手間がかかります。

「キャンドルLED」を利用すると、電源をつなぐだけでロウソクのような光り方でLEDが点灯します（**下図**）。Pico側からは点灯するか消灯するかを制御するだけで済み、プログラムを単純化できます。

図　ロウソクのように揺らいで光るキャンドルLED

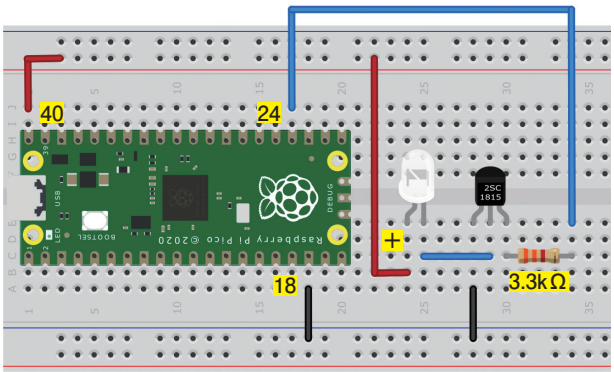


ロウソクのように明るくなったり暗くなったりする　（著者撮影）

キャンドルLEDをPicoから点灯制御するには**下図**のように接続します。ここで利用しているキャンドルLEDは5Vで動作するため、トランジスタを利用して5Vの電源を接続しています。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　キャンドル LED の接続図



（Fritzingで作成）

以下に示すプログラム「candle_led.py」を実行すると、1分間点灯した後、消灯します。

ソース　キャンドル LED の点灯、消灯を制御するプログラム

candle_led.py

```
from machine import Pin
import time

LED_PIN = 18 ── キャンドルLEDを接続したGPIOの番号

led = Pin( LED_PIN, Pin.OUT )

led.value( 1 )
time.sleep( 60 )

led.value( 0 )
```

27 動作

振動モーター

デジタル出力

プログラム vib.py

ライブラリ －

利用パーツ

●振動モーター ×1（秋：106784） ●FET×1（秋：107597） ●ダイオード×1（秋：100934）
●電池ボックス×1（秋：100327） ●コンデンサー 0.1μF×1（秋：100090）
●抵抗器1kΩ×1（秋：125102）、20kΩ×1（秋：103940）

振動モーターは、小さな振動を起こすことができる電子パーツです。スマホのバイブレーション機能のように、音を鳴らしたくない場面において、代わりに振動で通知するといった使い方ができます。

ここで紹介する振動モーター「LA3R5-480DE」は、回転軸に対して重りが偏って取り付けられています（**下図**）。この重りが回転すると、振動を引き起こします。

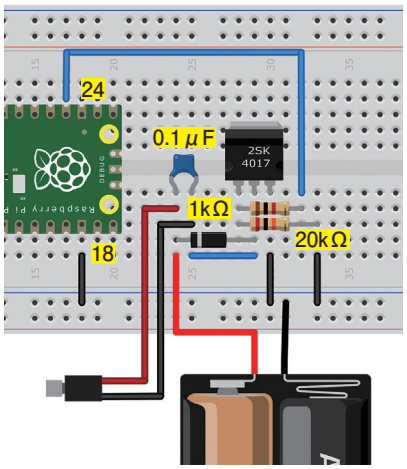
図　振動モーターの外観と振動する仕組み



振動モーターは小型のモーターですが、比較的大きな電流が流れます。LA3R5-480DEの場合、3Vで駆動すると100mAの電流が流れます。このため、DCモーター同様にFETを介して制御します（**下図**）。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　接続図



（Fritzingで作成）

デジタル出力をHighにすると回転し、Lowにすると停止します。スマホのバイブレーション機能と同じように、プログラムによって所定のパターンで振動させることが可能です。以下に示すプログラム「vib.py」では、PATTERNに設定したパターンで振動します。パターンは振動している時間、停止している時間、と交互に指定します。時間の単位は秒です。

ソース　振動モーターを所定のパターンで振動させるプログラム

vib.py

```
from machine import Pin
import time

M_PIN = 18

PATTERN = [ 1, 1, 2, 1, 3 ]
vib = Pin( M_PIN, Pin.OUT )

vib.value( 0 )

output = 1

for data in PATTERN:
    vib.value( output )
    time.sleep( data )
    if ( output == 1 ):
        output = 0
    else:
        output = 1

vib.value( 0 )
```

振動時間（秒）

振動のパターンを設定する

停止時間（秒）

パターンに従って振動させる

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

28 動作

デジタル出力

バイポーラー型ステッピングモーター

プログラム stepping.py

ライブラリ －

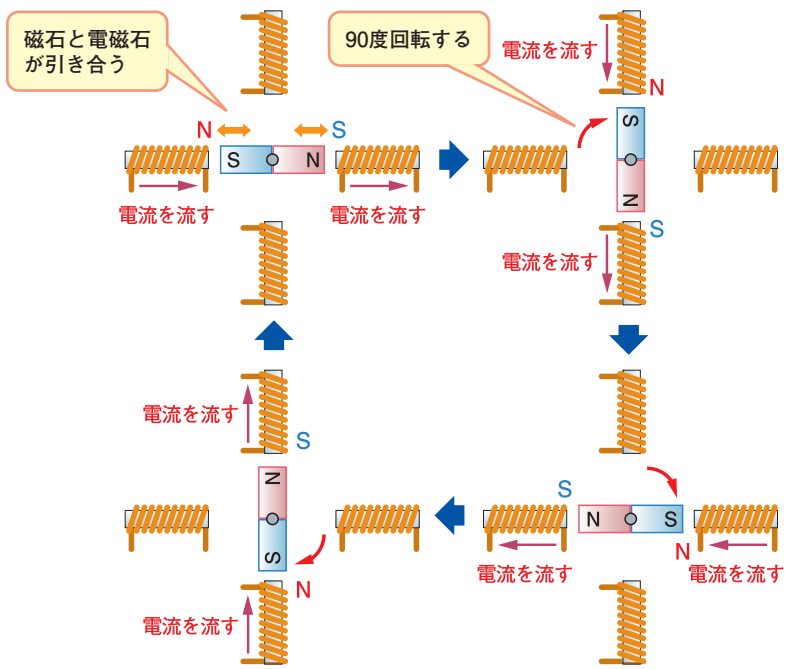
利用パーツ

●ステッピングモーター ×1（秋：105372） ●モータードライバー ×1（秋：109848）
●ACアダプター 5V×1（秋：111996） ●DCジャック×1（秋：108849）

本書の第4章の4-3で紹介した「DCモーター」は、回転するか停止するかを制御できますが、「指定した回数だけ回転したら停止する」といった正確な制御はできません。正確に回転を制御したい場合には「ステッピングモーター」を利用します。

ステッピングモーターは、**下図**のような原理で動作します。左右に配置した電磁石に電流を流すと磁石が横向きになります。次に上下の電磁石に電流を流すと、磁石が90度回転します。この手順を繰り返すことで、回転数や角度を正確に制御しながら回転させられます。実際の動作はもっと細かく、1回の切り替えで1度程度回転します（製品により異なります）。

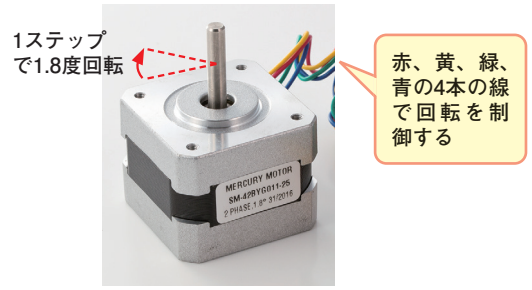
図　ステッピングモーターの動作原理



@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

ステッピングモーターには色違いの複数の線が付いています。ここに電源の＋極と－極を接続することで内部の電磁石の磁極を制御し、軸を回転させられます。ここで紹介するステッピングモーター「SM-42BYG011」は、赤、青、黄、緑の4本の線を備えています（**下図**）。

図　ステッピングモーターの外観



（日経BP撮影）

SM-42BYG011を回転させるには、各線に**下図**のように電源を接続します。Highは電源の＋極、Lowは－極を表します。時計回りに回転（正転）させるには、左表のようにまず赤、黄をHigh、緑、青をLowにします（①）。次に黄、緑をHigh、赤、青をLowにすると1ステップ分回転します（②）。SM-42BYG011の場合は、1ステップで1.8度回転します。続いて③の状態にするともう1ステップ回転します。このように、①～④の順に接続状態を切り替えることで1ステップずつ回転させられます（④まで達したら①に戻ります）。200回切り替えれば1回転になります（1.8度×200回＝360度）。

一方、右の表のように順次切り替えれば反時計回り（逆転）に回転させられます。

図　回転制御方法

順序	赤	黄	緑	青
①	High	High	Low	Low
②	Low	High	High	Low
③	Low	Low	High	High
④	High	Low	Low	High

時計回り（正転）

順序	赤	黄	緑	青
①	High	High	Low	Low
②	High	Low	Low	High
③	Low	Low	High	High
④	Low	High	High	Low

反時計回り（逆転）

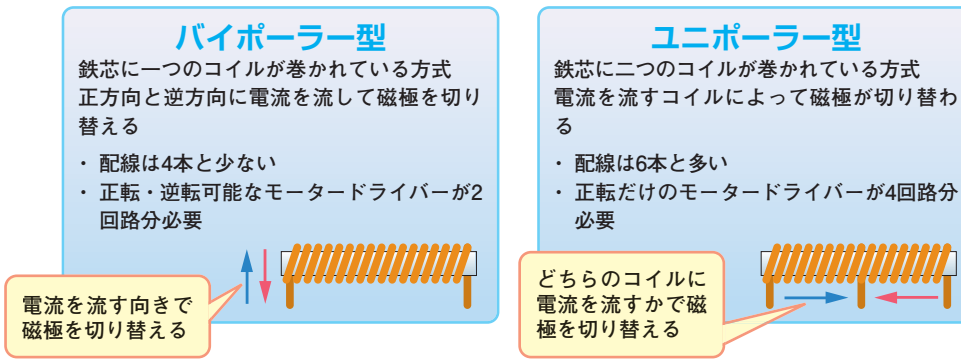
ステッピングモーターには「バイポーラー型」と「ユニポーラー型」の2種類があります（**下図**）。バイポーラー型は、電磁石が一つのコイルで構成されており、電流を流す向きによって磁極を切り替えます。このため本書の第5章のNo.4でDCモーターを制御する際に利用した、電流の向きを逆にできるフルブリッジ回路を搭載したモータードライバーが必要です。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

一方のユニポーラー型は、電磁石が二つのコイルで構成されています。どちらか一方に電流を流すことで磁極を切り替えます。流すか流さないかを切り替えるだけなのでフルブリッジドライバーを使わずに動かせます。半面、動かすには正転だけのモータードライバーが4回路分必要になります。

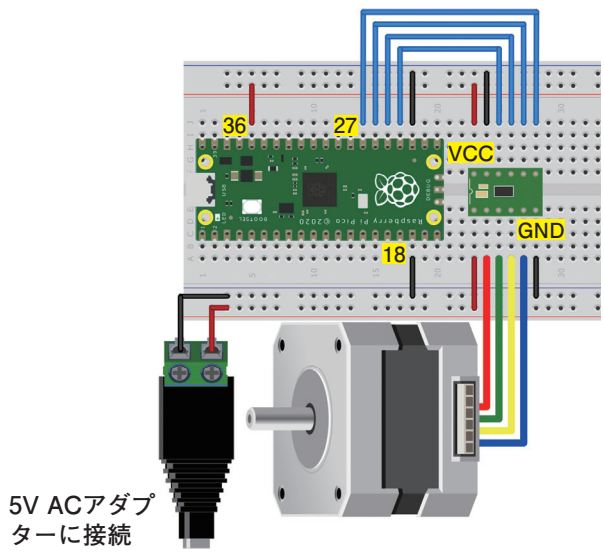
ここで利用するSM-42BYG011はバイポーラー型のステッピングモーターです。ユニポーラー型では回路の接続方法や制御方法が異なるので、購入する際には注意しましょう。

図　ステッピングモーターの種類



SM-42BYG011とPicoは**下図**のように接続します。このステッピングモーターは配線の赤と緑、青と黄のそれぞれに電磁石が接続されており、それぞれを別に制御します。このため、2系統を備えるモータードライバーのDRV8835を使って制御できます。モーターの制御電源としてACアダプターを利用して5Vを供給するようにしています。

図　接続図



（Fritzingで作成）

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

ステッピングモーターを制御するプログラム「stepping.py」を以下に示します。プログラム中の「STEPS」で指定したステップだけ回転し、その後停止します。「SPEED」の値を増減することで回転速度を変えられます。これは1ステップごとの待機時間（秒）で、値が大きいほど遅く、小さいほど速く回転します。「DIRECT」は回転方向です。1にすると正転、0にすると逆転します。

ソース　指定したスピードと回転方向でステッピングモーターを制御するプログラム

stepping.py

```
from machine import Pin
import time

ST_MOTOR_A0 = 18
ST_MOTOR_A1 = 19
ST_MOTOR_B0 = 20
ST_MOTOR_B1 = 21

STEPS = 200
SPEED = 0.01
DIRECT = 1

st_a0 = Pin( ST_MOTOR_A0, Pin.OUT )
st_a1 = Pin( ST_MOTOR_A1, Pin.OUT )
st_b0 = Pin( ST_MOTOR_B0, Pin.OUT )
st_b1 = Pin( ST_MOTOR_B1, Pin.OUT )

ST_MOTOR_OUT = [ 0b1100,
                 0b0110,
                 0b0011,
                 0b1001 ]

def to_bin( val ):
    if ( val == 0 ):
        return 0
    return 1

all_step = 0
finish = 0

while True:
    i = 0
    while ( i < len( ST_MOTOR_OUT ) ):
        if ( DIRECT  == 1 ):
            step_count = i
        else:
            step_count = len( ST_MOTOR_OUT ) - 1 - i

        a0 = to_bin( ST_MOTOR_OUT[ step_count ] & 0b1000 )
        a1 = to_bin( ST_MOTOR_OUT[ step_count ] & 0b0010 )
        b0 = to_bin( ST_MOTOR_OUT[ step_count ] & 0b0100 )
        b1 = to_bin( ST_MOTOR_OUT[ step_count ] & 0b0001 )

        print ( all_step, " : ", a0, a1, b0, b1 )

        st_a0.value( a0 )
        st_a1.value( a1 )
        st_b0.value( b0 )
        st_b1.value( b1 )

        time.sleep( SPEED )
        i = i + 1
        all_step = all_step + 1
        if ( STEPS <= all_step ):
            finish = 1
            break

    if ( finish == 1 ):
        break
```

モータードライバーに接続したGPIOの番号

回転させるステップ数

1ステップごとの待機時間

回転方向。1は正転、0は逆転

29 動作

連続回転サーボモーター

PWM出力

プログラム rotservo.py

ライブラリ －

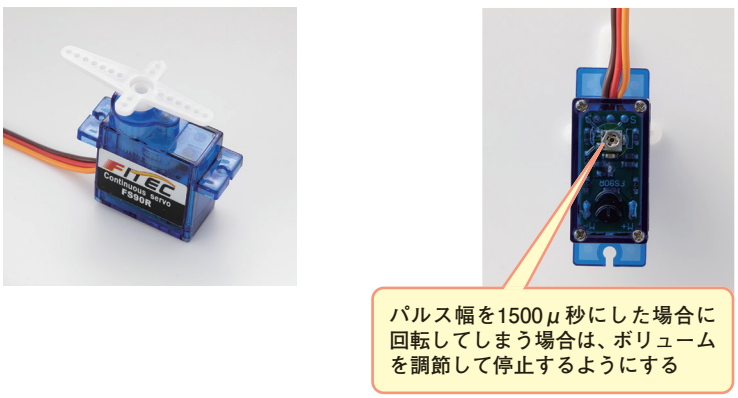
利用パーツ

●連続回転サーボモーター ×1（秋：113206）

「連続回転サーボモーター（ローテーションサーボモーター）」は、DCモーターのように連続して回転するサーボモーターです。DCモーターの代わりとして、模型の車を動かす目的などで利用できます（**下図**）。

連続回転サーボモーターは、本書の第5章のNo.5で紹介した通常のサーボモーターと同じ形状をしており、回転軸にホーンを付けて作品などに取り付けます。ホーンのほかに、回転軸に直接差し込めるタイヤも販売されています（秋：113207）。

図　連続回転サーボモーターの外観



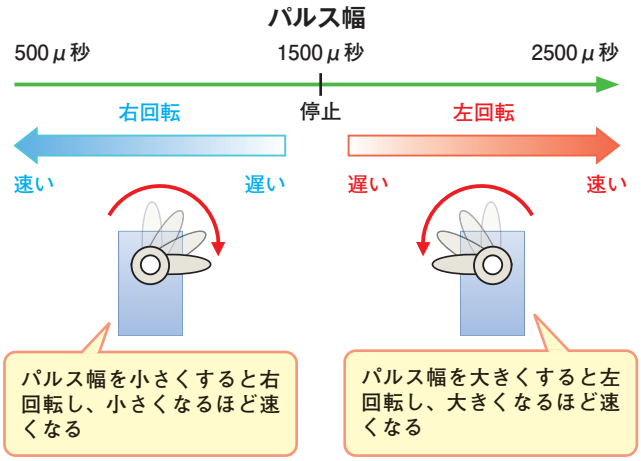
（日経BP撮影）

回転の制御には、通常のサーボモーターと同じPWM出力を使います。ここで利用する「FS90R」は、パルス幅を1500μ秒にすると停止し、それよりもパルス幅を大きくすると左回転、小さくすると右回転します（**下図**）。

回転速度の調節も可能で、1500μ秒に近い値だとゆっくり、1500μ秒より値が離れるほど速くなります。

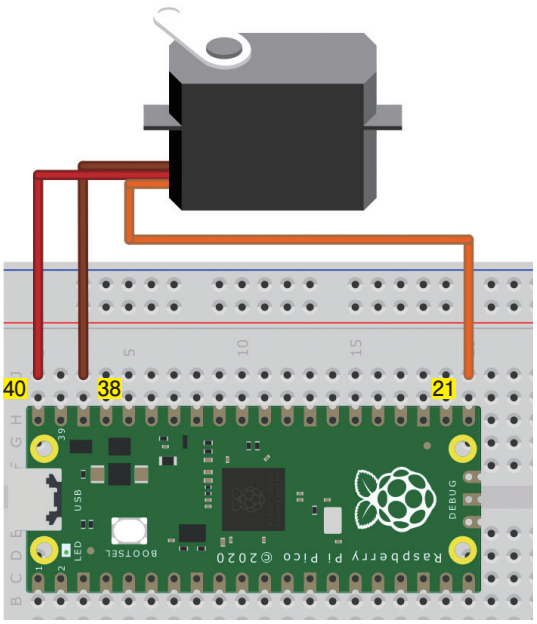
@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　連続回転サーボモーターの制御方法



Picoとの接続方法も通常のサーボモーターと変わりません（**下図**）。制御線をPicoのGPIOに接続し、電源は5V端子に接続します。

図　接続図



（Fritzingで作成）

連続回転サーボモーターの制御プログラム「rotservo.py」を以下に示します。本書の第5章のNo.5で紹介したサーボモーターと同様に、pulse_width()を使ってパルス幅をduty_u16()で利用できる値に変換しています。

プログラムを実行すると、右にゆっくり回転、右に速く回転、停止、左にゆっくり回転、左に速く回転、停止を繰り返します。

もし、停止（パルス幅が1500 μ秒）のところで回転が止まらない場合は、製品の個体差が原因です。底面にあるボリュームを回して停止するように調節してください。

ソース　連続回転サーボモーターの速度と回転方向を制御するプログラム

rotservo.py

```
from machine import Pin, PWM
import time

SERVO_PIN = 16 ── 信号線を接続したGPIOの番号
PWM_FREQ = 50

def pulse_width( val, freq = PWM_FREQ, resol = 65535 ):
    pulse = freq * val * 1e-6 * resol
    return int( pulse )

servo = PWM( Pin( SERVO_PIN ) )
servo.freq( PWM_FREQ )

while True:
    duty = pulse_width( 1400 )
    servo.duty_u16( duty )        ── ゆっくり右回転
    time.sleep( 1 )

    duty = pulse_width( 1000 )
    servo.duty_u16( duty )        ── 速く右回転
    time.sleep( 1 )

    duty = pulse_width( 1500 )
    servo.duty_u16( duty )        ── 停止
    time.sleep( 1 )

    duty = pulse_width( 1600 )
    servo.duty_u16( duty )        ── ゆっくり左回転
    time.sleep( 1 )

    duty = pulse_width( 2000 )
    servo.duty_u16( duty )        ── 速く左回転
    time.sleep( 1 )

    duty = pulse_width( 1500 )
    servo.duty_u16( duty )        ── 停止
    time.sleep( 1 )
```


@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

30 動作

ソレノイド

デジタル出力

プログラム solenoid.py

ライブラリ －

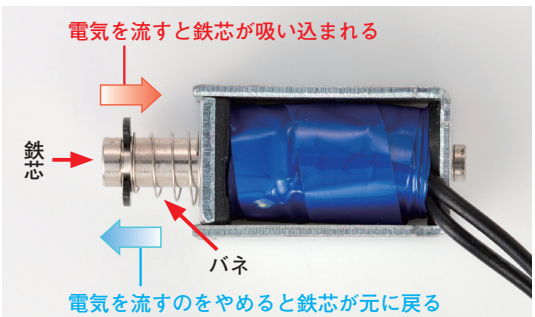
利用パーツ

●ソレノイド×1（秋：110761） ●FET×1（秋：107597） ●ダイオード×1（秋：100934）
●電池ボックス×1（秋：100327） ●コンデンサー 0.1μF×1（秋：100090）
●抵抗器1kΩ×1（秋：125102）、20kΩ×1（秋：103940）

「ソレノイド」は直線的な動作を発生させるための電子パーツです（**下図**）。例えば、ボタンを押したり、ドアをロックしたりする目的で使われます。

コイルの中に鉄芯が入っており、コイルに電気を流すと発生した磁力により鉄芯が引き込まれます。電気を流すのをやめると鉄芯に取り付けられているバネの力で元の位置に戻ります。

図　ソレノイドの外観

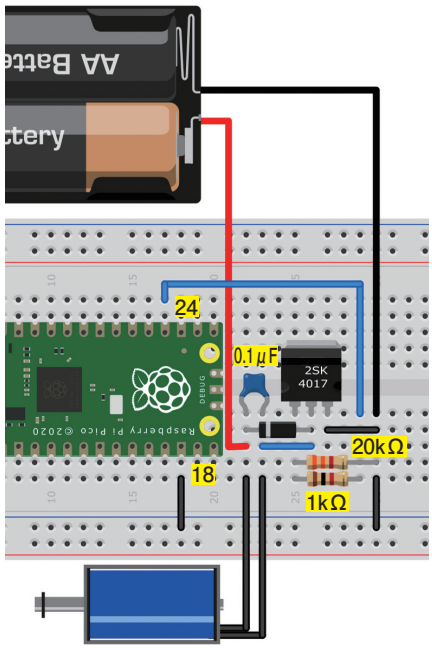


（日経BP撮影）

ソレノイドを動作させるには大きな電流が必要です。ここで紹介する「ZHO-0420S-05A4.5」は、数百mAの電流を必要とします。このため、第4章の4-3で紹介したDCモーターと同様にFETを介して制御します。Picoも含めた接続方法を**下図**に示します。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　接続図



（Fritzingで作成）

ソレノイドを制御するプログラム「solenoid.py」を以下に示します。Picoのデジタル出力をHighにするとソレノイドの鉄芯が吸い込まれ、Lowにすると元に戻ります。このプログラムでは、鉄芯を10秒間吸い込んだ状態にし、その後元に戻しています。

ソース　ソレノイドの鉄芯を10秒間吸い込むプログラム

solenoid.py

```
from machine import Pin
import time

S_PIN = 18 ── ソレノイドを接続したGPIOの番号
solenoid = Pin( S_PIN, Pin.OUT )

solenoid.value( 1 )
time.sleep( 10 )
solenoid.value( 0 )
```


@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

31 表示

バーLED

デジタル出力

プログラム barled.py

ライブラリ －

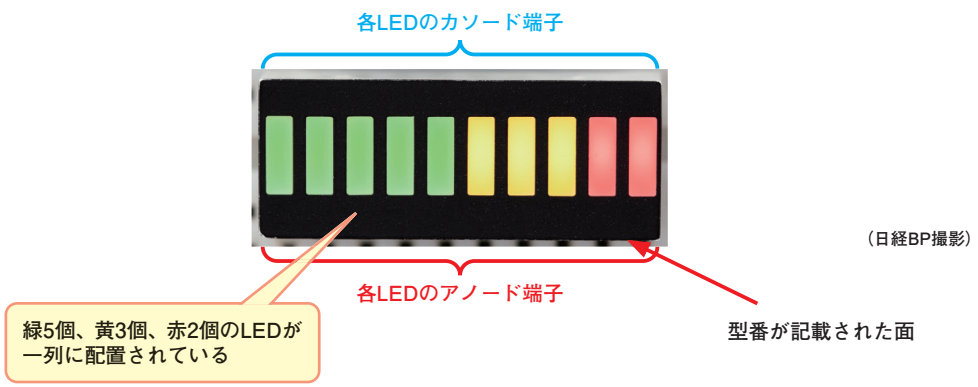
利用パーツ

●バー LED×1（秋：105347） ●抵抗器330Ω×10（秋：125331）

「バーLED」は、複数のLEDを一つの部品として詰め込む形で一列に配置した電子パーツです。音量の表示用インジケーターや、進捗を通知するプログレスバーなどの用途で使われます。ここでは緑色5個、黄色3個、赤色2個の計10個のLEDが配置された「OSX10201-GYR1」（**下図**）を使う方法を紹介します。

OSX10201-GYR1には、LEDごとにアノードとカソード端子が付いています。型番が記載されている面にある端子がアノードです。

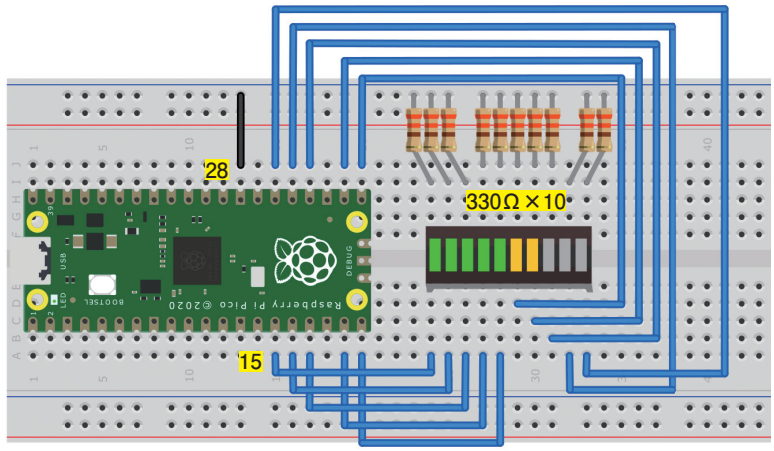
図　10個のLEDが一列に配置されたバーLEDの外観



それぞれのLEDをPicoのGPIOに接続し、デジタル出力で制御します（**下図**）。この際、制限抵抗を忘れないように接続してください。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　接続図



（Fritzingで作成）

指定した値で点灯するLEDの数を変えられるように制御するプログラム「barled.py」を以下に示します。プログラム中では、「OUTPUT」変数に0～10の数値を指定すると、緑色側のLEDから指定した数のLEDが点灯します。例えば、3の場合は3個の緑のLEDが点灯し、7の場合は緑5個、黄色2個のLEDが点灯します。

ソース　指定した数のLEDを点灯するプログラム

barled.py

```
from machine import Pin

OUTPUT = 7

LED_PIN = [ 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 ]

led = []

i = 0
while ( i < len( LED_PIN ) ):
    led.append( Pin( LED_PIN[ i ], Pin.OUT ) )
    led[ i ].value(0)
    i = i + 1

count = 0
for pin in LED_PIN:
    if ( count < OUTPUT ):
        led[ count ].value( 1 )
    else:
        led[ count ].value( 0 )
    count = count + 1
```

指定した数だけLEDが点灯する

各LEDに接続したGPIOの番号

すべてのLEDを消灯する

OUTPUTに指定した数分のLEDを点灯する

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

32 入力

マイクロスイッチ

デジタル入力

プログラム microsw.py
ライブラリ －

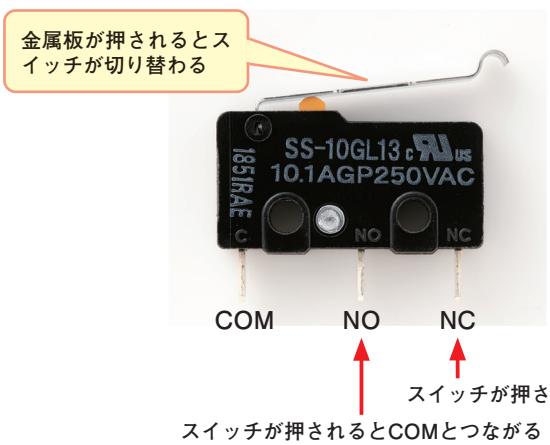
利用パーツ

●マイクロスイッチ×1（秋：114659）

「マイクロスイッチ」は、高い信頼性と耐久性を備えたスイッチです。所定の力が加わると、確実にスイッチが切り替わるようになっています。マウスのボタンや、動く作品が壁に衝突したことを検知する仕組みなどに利用されています。金属製のレバーが付いているタイプのマイクロスイッチを使うと、スイッチとして反応する面が広くなる上、てこの原理により、弱い力でもスイッチを押すことができるようになります。

ここで紹介するマイクロスイッチは「COM」「NO」「NC」という三つの端子を備えています（**下図**）。NO（Normally Open）はスイッチを押していない状態では何も接続されず、スイッチを押すとCOMと接続された状態になります。一方、NC（Normally Close）はNOと逆で、スイッチを押していない状態ではCOMと接続され、スイッチが押されると何も接続しない状態になります。用途に応じてNOとNCのどちらを利用するかを選択します。

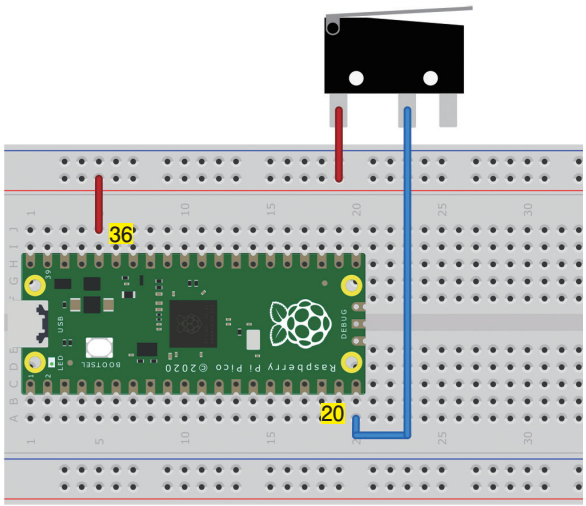
図　マイクロスイッチの外観



（日経BP撮影）

Picoとマイクロスイッチは**下図**のように接続します。ここではNO端子を利用し、マイクロスイッチが押されたらCOM（電源）と接続するようにしています。

図　接続図



（Fritzingで作成）

制御用プログラム「microsw.py」を以下に示します。このプログラムでは、マイクロスイッチが押されると「Hit」と表示します。また、マイクロスイッチを接続したGPIOについて、プルダウン抵抗が有効になるように設定しています。

ソース　マイクロスイッチが押されたら「Hit」と表示するプログラム

microsw.py

```
from machine import Pin
import time

SW_PIN = 15 ── マイクロスイッチを接続したGPIOの番号

sw = Pin( SW_PIN, Pin.IN, Pin.PULL_DOWN ) ── プルダウン抵抗を有効にする

while True:
    if( sw.value() == 1 ):
        print( "Hit" ) ──┐
                         ├── 押されたら「Hit」と表示する
    time.sleep( 0.5 )
```


@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

33 入力

リードスイッチ

デジタル入力

プログラム reed.py

ライブラリ －

利用パーツ

●ドアスイッチ×1（秋：113371）

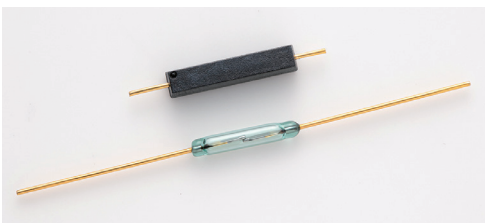
「リードスイッチ」は、磁石を近づけることでスイッチを切り替えることのできる電子パーツです（**下図**）。センサー自体に触れずに感知できるため、動くものの状態を調べたり、密封した空間内の状態を調べたりできます。ドアや窓などの開閉の状態を知りたい場合にも役立ちます。

下図の右側に示したように、リードスイッチの内部には、2枚の金属板が離れた状態で配置されています。磁石を近づけると、金属板がN極またはS極になります。N極とS極は引き合うので金属板がくっつき、導通状態になります。下図の左側下のガラス管を使った透明のリードスイッチを見ると、内部に2枚の金属版が配置されている様子を確認できます。

図　リードスイッチの外観と仕組み

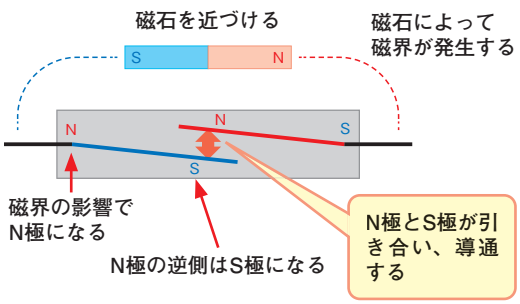
リードスイッチの外観

上はプラケース入り（秋：104209）、下はガラス管入り（秋：103676）のタイプ。



（日経BP撮影）

リードスイッチの仕組み



ドアなどの開閉状態を調べる目的で作られた、「ドアスイッチ」と呼ばれるリードスイッチの応用製品も市販されています（**下図**）。ドアスイッチは、二つの部品に分かれています。一つにはリードスイッチが、もう一つには磁石が入っており、二つの部品を近づけることでリードスイッチがオンの状態になる仕組みです。利用する際は、ドアの枠にリードスイッチ側を取り付けて配線し、ドア側に磁石を取り付けておきます。ドアを閉めると磁石がリードスイッチに近づき、スイッチがオンになります。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

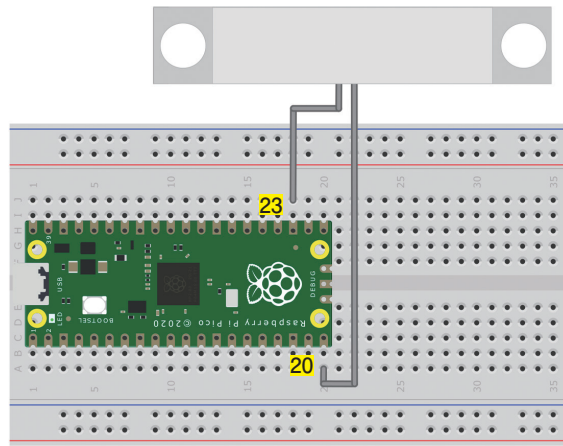
図　ドアの開閉を調べるのに使える「ドアスイッチ」の外観



（日経BP撮影）

リードスイッチとPicoは**下図**のように接続します。リードスイッチがオンになると、PicoのGPIOはLowに接続された状態になります。

図　接続図



（Fritzingで作成）

制御用プログラム「reed.py」を以下に示します。ドアスイッチを接続したGPIOはプルアップ抵抗を有効にしておきます。プログラムを実行すると、ドアが開いていれば「Open.」、閉まっていれば「Close.」と表示されます。

ソース　ドアスイッチを使って扉の状態を確認するプログラム

reed.py

```
from machine import Pin
import time

SW_PIN = 15 ── ドアイッチを接続した
               GPIOの番号

sw = Pin( SW_PIN, Pin.IN, Pin.PULL_UP )
↗

while True:
    if( sw.value() == 0 ):
        print( "Close." )
    else:
        print( "Open." )

    time.sleep( 0.5 )
```


@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

34 入力

デジタル入力

2進数出力ロータリースイッチ

プログラム rotlog.py

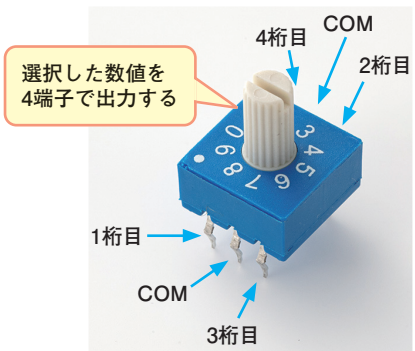
ライブラリ －

利用パーツ

●ロータリースイッチ0～9正論理×1（秋：102274）

本書の第5章のNo.11でも紹介した「ロータリースイッチ」は、接続先の回路を切り替えられる電子パーツです。AV機器などで出力先を切り替えるつまみがありますが、そのつまみなどに使用されています。このように単純に接続先の回路を切り替える用途以外にも、ロータリースイッチの中には、ダイヤル（つまみ）で選択した数値を出力する「正論理」と呼ぶタイプの製品もあります（**下図**）。合わせた数値によって内部のスイッチが切り替わり、各桁の端子の状態が変化します。

図　2進数出力ロータリースイッチの外観



（日経BP撮影）

ここで利用するロータリースイッチは、0～9の数値を選択でき、選んだ数値を4本の端子の出力状態で表現します（**下表**）。表中の「オフ」は何もつながっていない状態、「オン」はCOM端子とつながった状態です。

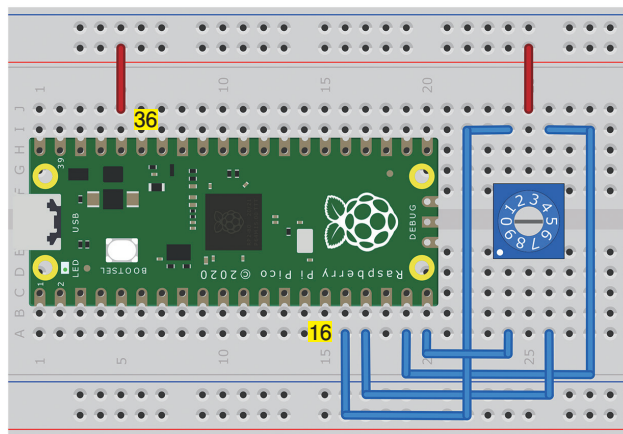
@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

表　各数値に合わせた際のロータリースイッチの端子の状態

数値	4桁目	3桁目	2桁目	1桁目
0	オフ	オフ	オフ	オフ
1	オフ	オフ	オフ	オン
2	オフ	オフ	オン	オフ
3	オフ	オフ	オン	オン
4	オフ	オン	オフ	オフ
5	オフ	オン	オフ	オン
6	オフ	オン	オン	オフ
7	オフ	オン	オン	オン
8	オン	オフ	オフ	オフ
9	オン	オフ	オフ	オン

Picoで数値を読み取るには、**下図**のようにCOM端子を3.3V、各桁の端子をGPIOに接続します。

図　接続図

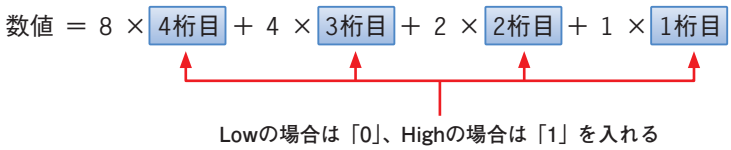


（Fritzingで作成）

プログラムで数値を読み取るには、各桁の端子に接続したPicoのGPIOの状態を調べ、Highの場合は「1」、Lowの場合は「0」とします。取得した値を**下図**の計算式に当てはめると、どの数値を選択したかが分かります。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　数値を求める計算式



作成した数値取得用プログラム「rotlog.py」を以下に示します。「ROT_PIN」には各桁の端子に接続したGPIOの番号を指定しておきます。

ソース　ロータリースイッチで合わせた数値を取得するプログラム

rotlog.py

```
from machine import Pin
import time

ROT_PIN = [ 15, 14, 13, 12 ]

sw = []
for i in ROT_PIN:
    sw.append( Pin( i, Pin.IN, Pin.PULL_DOWN ) )

data = [ 0, 0, 0, 0 ]

while True:
    i = 0
    while i < len( ROT_PIN ):
        data[i] = sw[i].value()
        i = i + 1

    value = data[3] * 8 + data[2] * 4 + data[1] * 2 + data[0]
    print ( "{} ({},{},{},{})".format( value, data[3], data[2], data[1], data[0] ) )

    time.sleep( 1 )
```

各桁に接続したGPIOの番号

プルダウン抵抗を有効にする

各桁の状態を読み取る

計算して数値を求める

35 入力

振動スイッチ

デジタル入力

プログラム fallsw.py

ライブラリ ー

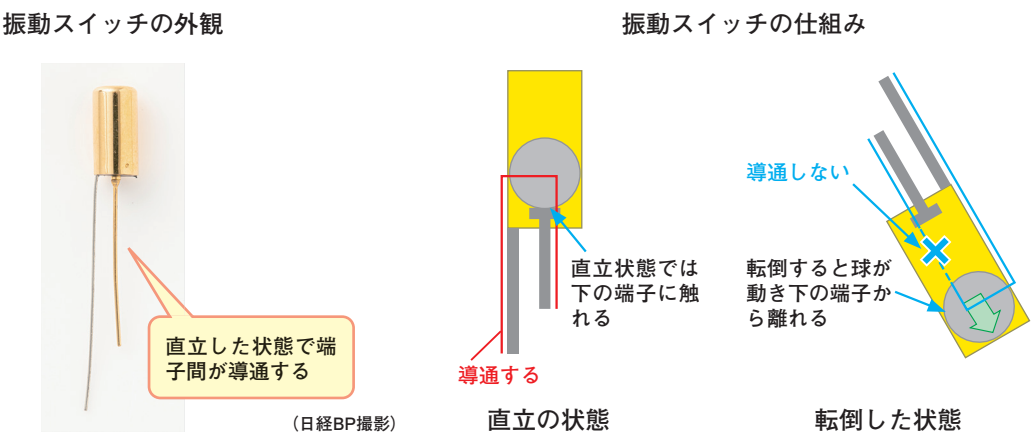
利用パーツ

●振動スイッチ×1（秋：102349）

「振動スイッチ」は、直立状態か転倒状態かによってオンとオフが切り替わるスイッチです（**下図**）。暖房器具が転倒したことを検知するなどの用途があります。

振動スイッチの内部には金属球が入っており、上下方向に自由に動きます。このため、直立状態では底面に配置された端子に金属球が触れて端子間が導通していますが、転倒すると金属球が端子から離れて導通しなくなります。

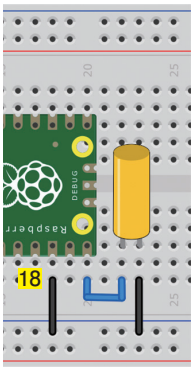
図　振動スイッチの外観と仕組み



振動スイッチとPicoは、**下図**のように接続します。直立時には振動スイッチがオンの状態となっているため、GPIOはGNDと接続されています。転倒するとオフとなり、GPIOには何も接続されていない状態になります。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　接続図



（Fritzingで作成）

転倒しているかどうかを表示するプログラム「fallsw.py」を以下に示します。振動スイッチを接続するGPIOは、プログラム中でプルアップ抵抗を有効にしておきます。

実行すると、直立状態であれば「Stability.」、転倒状態では「Fall down.」と表示します。

ソース　転倒しているかどうかを表示するプログラム

fallsw.py

```
from machine import Pin
import time

SW_PIN = 15 ── 振動スイッチを接続したGPIOの番号

sw = Pin( SW_PIN, Pin.IN, Pin.PULL_UP ) ── プルアップ抵抗を有効にする

while True:
    if( sw.value() == 1 ):
        print( "Fall down." ) ── 転倒した場合に表示
    else:
        print( "Stability." ) ── 直立状態の場合に表示

    time.sleep( 1 )
```

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

36 入力

キーパッド

デジタル入力

プログラム keypad.py

ライブラリ －

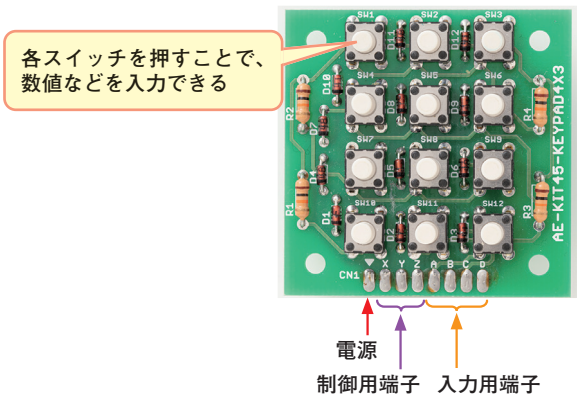
利用パーツ

●キーパッド×1（秋：112229）

「キーパッド」は、複数の押しボタンスイッチ（キー）が配置された電子パーツです。単純に複数個のスイッチとして使えるほか、それぞれのキーに文字などを割り当てることで、PCのキーボードのような文字入力用パーツとしても利用可能です。

ここで紹介するキーパッドは4行3列で合計12個のキーが配置されています（**下図**）。0～9とドット、決定キーを割り当てることで、テンキーのように数字の入力に使えます。

図　複数の押しボタンスイッチを配置したキーパッドの外観



（日経BP撮影）

キーパッドの読み取りには、本書の第5章のNo.8で紹介した「マトリクスLED」と同じようにダイナミック制御を使います。ここで利用するキーパッドには、読み取り対象列を選択するための3列分の制御用端子（X～Z列）と、押しているキーを判定するための4行分の入力用端子（A～D行）が用意されています（**下図**）。

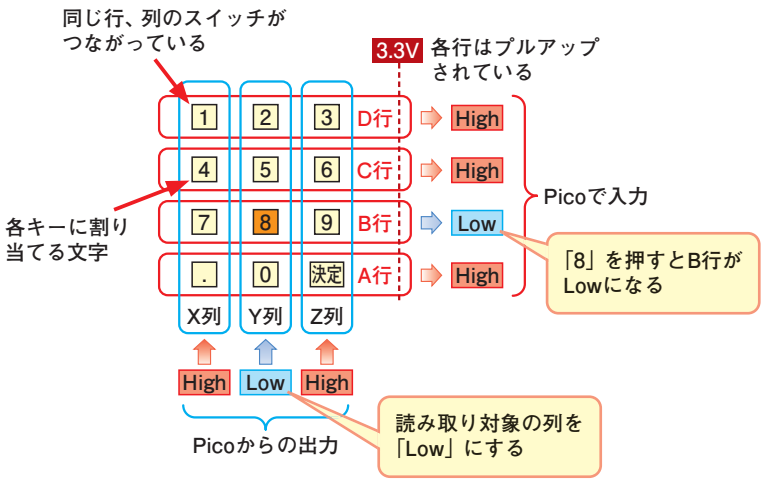
読み取り対象となる列をLow、他の列をHighに設定しておきます。この状態で対象列内のいずれかのキーが押されていると、該当する行の入力用端子がHighからLowに変化します。Picoのデジタル入力で各行の状態を読み取ると、どのキーが押されているかが分かります。

この処理をX、Y、Zと順にずらしながら短い時間で切り替えることで、すべてのスイッチの状

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

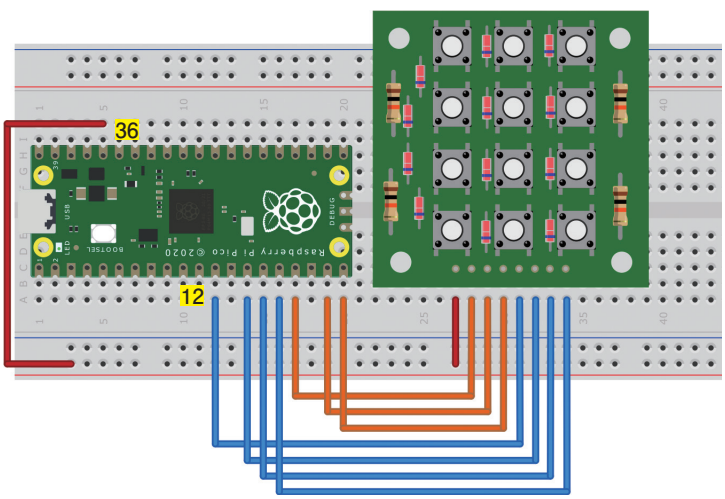
態を確認できます。なお、入力用端子はプルアップ抵抗を備えているため、Pico 側で内蔵プルアップ抵抗を有効にする必要はありません。

図　ダイナミック制御ですべてのキーの状態を読み取る



キーパッドは**下図**のようにPicoと接続します。行の端子に接続したGPIOはデジタル入力、列の端子に接続したGPIOはデジタル出力として使います。

図　接続図



（Fritzingで作成）

キーパッドから数値を入力するプログラム「keypad.py」を以下に示します。プログラム中の「KEY_MAP」で数を割り当てたキーを使って数値を入力し、決定キーを押すと入力した数値が表示されます。ドットは小数点として利用可能です。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

ソース　キーパッドから数値を入力するプログラム

keypad.py

```
from machine import Pin
import time
                    各端子に接続したGPIOの番号
IN_PIN = [ 9, 10, 11, 12 ]
SEL_PIN = [ 13, 14, 15 ]

KEY_MAP = [ [ '.', '7', '4', '1' ],
            [ '0', '8', '5', '2' ],
            [ '¥n','9', '6', '3' ] ]
                    各キーに割り当てる文字
sel_sw = []
in_sw = []

i = 0
while ( i < len( SEL_PIN ) ):
    sel_sw.append( Pin( SEL_PIN[ i ],
 Pin.OUT ) )
    sel_sw[ i ].value( 1 )
    i = i + 1

i = 0
while ( i < len( IN_PIN ) ):
    in_sw.append( Pin( IN_PIN[ i ],
Pin.IN ) )
    i = i + 1

keybuf = []

def keypad():
    global keybuf
    fg_in = 0
    i = 0
    while( i < len( SEL_PIN ) ):
        sel_sw[ i ].value( 0 )
        j = 0
        while( j < len( IN_PIN ) ):
            if ( in_sw[j].value( ) ==
 0 ):
                keybuf.append( KEY_MAP
[i][j] )
                fg_in = 1
                while ( in_sw[j].value
( ) == 0 ) :
                    time.sleep( 0.05 )
            j = j + 1
        sel_sw[ i ].value( 1 )
        i = i + 1
    return ( fg_in )

def bufread():
    global keybuf
    buf = ''
    i = 0
    point = 0
    while ( i < len( keybuf ) ):
        if ( keybuf[i] == '¥n' ):
            break
        elif( keybuf[i] == '.' ):
            point = point + 1
            if ( point > 1 ):
                break
            else:
                buf = buf + keybuf[i]
        else:
            buf = buf + keybuf[i]
        i = i + 1

    keybuf = []
    return ( float( buf ) )

while ( True ):
    f = keypad()
    if ( f == 1 ):
        if( keybuf.count('¥n') > 0 ):
            value = bufread()
            print( value )
```


@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

37 センサー

I²C

温湿度センサー

プログラム temphumi.py

ライブラリ aht25.py

利用パーツ

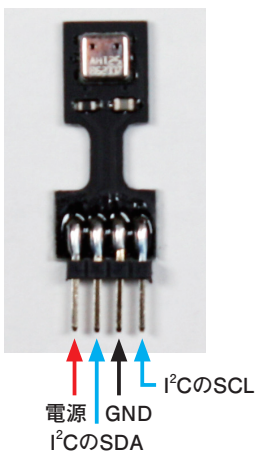
●温湿度センサー ×1（秋：116731）

室内が快適かどうかを知るには、「温湿度センサー」が利用できます。温湿度を計測することで、部屋の快適度を表す「不快指数」が求まります。この指数を基に、エアコンや加湿器を制御して、より快適な状態に近づけるといった応用も可能です。

「AHT25」は、温度と湿度を同時に計測できるセンサーです（**下図**）。温度は-40～80℃、湿度は0～100%の範囲で計測可能です。精度は、温度が±0.3℃、湿度が±2%となっており、比較的正確と言えるでしょう。

AHT25は、計測した温度や湿度をI²C通信を使って取得できます。I²Cアドレスは、0x38です。

図　AHT25の外観

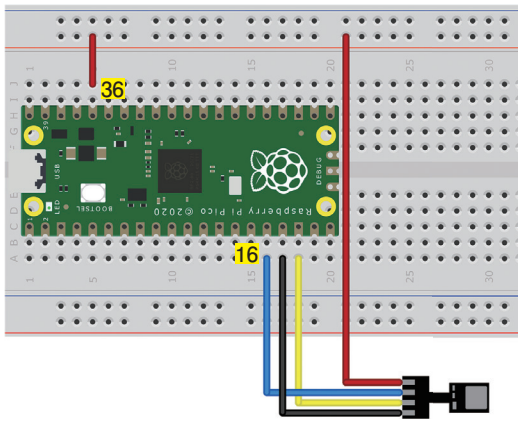


（著者撮影）

AHT25とPicoは**下図**のように接続します。付属のジャンパー線を利用します。ただし、ジャンパー線の色と機能が一致しないので注意が必要です（2025年4月現在）。電源の端子側を赤の線になるように接続した場合、SDAが青線、GNDは黄線、SCLが黒線になります。特にGNDが一般的に使われる黒線とならないので間違えないようにしましょう。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　接続図



（Fritzingで作成）

AHT25の制御には、本書のサポートサイトから入手できるライブラリ「aht25.py」を利用します。ダウンロードして展開したら、本書の第2章の2-3で解説した手順に従ってPicoに同じファイル名でライブラリを保存してください。

同ライブラリを使って、AHT25で温度、湿度を定期的に計測する計測プログラム「temphumi.py」を以下に示します。ライブラリの「sensor.get_temphumi()」関数を呼び出すことで、センサーが計測した温度と湿度を取得しています。

ソース　温度、湿度を定期的に計測するプログラム

temphumi.py

```
from machine import Pin, I2C
import time
import aht25

I2C_SDA = 12
I2C_SCL = 13
I2C_CH = 0

AHT25_ADDR = 0x38

i2c = I2C( I2C_CH, scl=Pin( I2C_SCL ), sda=Pin( I2C_SDA ), freq=100000)
i2c.scan()

sensor = aht25.aht25( i2c, AHT25_ADDR )

while True:
    ( temp, humi ) = sensor.get_temphumi()

    print( "T:{:.1f}C H:{:.1f}%".format( temp, humi ))

    time.sleep(1)
```

SDAとSCLのGPIO番号

I²Cチャンネル

I²Cアドレス

温度と湿度を取得する

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

38 センサー　SPI

熱電対

プログラム　thermocouple.py
ライブラリ　－

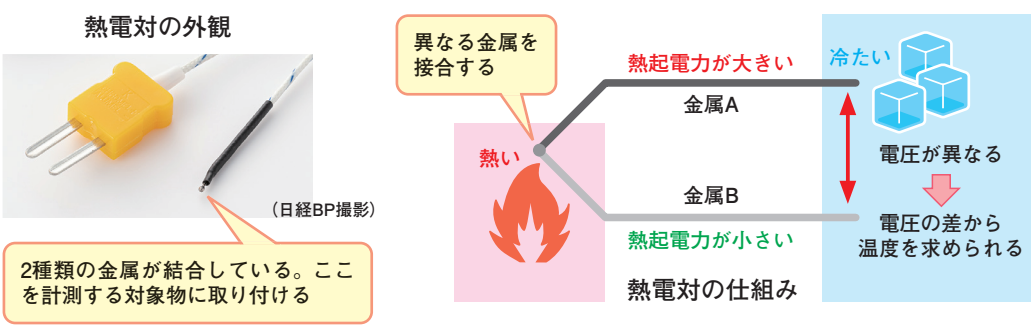
利用パーツ

●熱電対×1（秋：117400）　●熱電対アンプ×1（秋：108218）
●熱電対用コネクタ×1（秋：114971）

「熱電対」は、1000℃を超えるような高温を計測可能な温度センサーです（**下図**）。調理用の鍋や陶芸用の窯、電気炉など、高温かつ温度管理が必要となる場所や器具の温度を計測する目的でよく使われています。

金属は、両端の温度が異なると、内部を電荷が移動します。これを「熱起電力」と呼びます。熱起電力の大きさは金属によって異なります。熱電対は異なる二つの金属を接合したものです。接合した部分を温めると、温度差によりそれぞれの金属で熱起電力が発生し、他方の端との間で電圧が生じます。異なる金属では熱起電力が異なるため、それぞれの金属の端では電圧の差が生じます。この電圧の差を調べることによって温度を求めることができます。

図　熱電対の外観と仕組み



熱電対は、利用する金属によって計測できる温度が異なります。例えば、クロメル（ニッケルとクロムの合金）とアルメル（ニッケルとアルミニウムの合金）という2種類の金属を使った「K型熱電対」は、-270～1372℃の範囲の温度を計測できます。白金（プラチナ）とその合金である白金ロジウム（13%）を接合した「R型熱電対」なら、-50～1768℃とさらに高い温度まで計測可能です（**下表**）。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

表　主な熱電対と計測可能な温度範囲

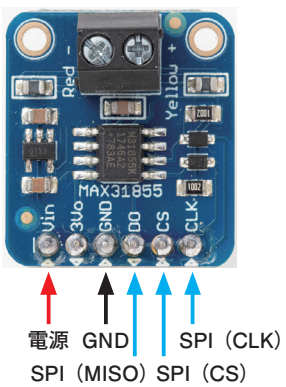
熱電対の型	利用金属（＋側）	利用金属（－側）	計測可能温度範囲*1
K型	クロメル	アルメル	-270～1372℃
N型	ナイクロシル	ナイシル	-270～1300℃
E型	クロメル	コンスタンタン	-270～1000℃
J型	鉄	コンスタンタン	-210～1200℃
T型	銅	コンスタンタン	-270～400℃
B型	白金・ロジウム（30%）	白金・ロジウム（6%）	0～1820℃
R型	白金・ロジウム（13%）	白金	-50～1768℃
S型	白金・ロジウム（10%）	白金	-50～1768℃

＊1　JIS規格で定義されている温度範囲

ここでは、秋月電子通商が販売するK型熱電対の利用方法を紹介します。-200～1250℃の温度計測に対応しています。この熱電対は、1℃の温度変化に対して電圧が約40.7 μ V上昇または下降します。このため、電圧を調べることで温度を計算できます。

ただし、熱電対が出力する電圧は非常に小さく、そのまま入力したのではPicoが搭載するADコンバーターでは読み取れません。そこで、熱電対専用のADコンバーターを別途用意して読み取ります。ここでは、米Adafruit Industries社の「MAX31855K」を搭載した熱電対用ADコンバーターを利用する方法を紹介します（**下図**）。熱電対から入力された電圧を増幅した後、14ビットのADコンバーターでデジタル値に変換します。変換した計測値はSPI通信を使ってPicoで読み取ります。

図　熱電対用ADコンバーター「MAX31855K」搭載モジュール

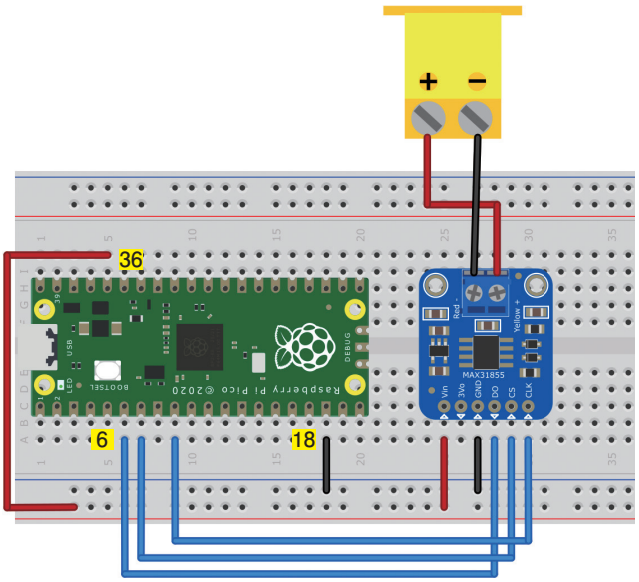


（日経BP撮影）

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

熱電対および熱電対用ADコンバーターは**下図**のようにPicoと接続します。熱電対用のコネクタを熱電対用ADコンバーターと接続します。ADコンバーターの「Yellow＋」と記載されている端子をコネクタの「＋」と記載された端子に接続します。接続したら、熱電対をコネクタに差し込みます。SPI通信をするためにMISOをGPIO 4（6番端子）、CLKをGPIO 6（9番端子）、CSをGPIO 5（5番端子）に接続します。MAX31855Kは出力しかしないため、MOSIを接続する必要はありません。

図　接続図



（Fritzingで作成）

熱電対で計測した温度を取得するプログラム「thermocouple.py」を以下に示します。get_thermo()関数を呼び出すことで、熱電対の計測値を取得します。

ソース　熱電対で計測した温度を取得するプログラム

thermocouple.py

```
from machine import Pin, SPI
import time

CS_PIN = 5
MISO_PIN = 4
SCLK_PIN = 6
SPI_CH = 0
SPI_SPEED = 10000000

cs = Pin( CS_PIN, Pin.OUT )
cs.value( 1 )

spi = SPI( SPI_CH, sck=Pin( SCLK_PIN ), miso=Pin( MISO_PIN ), baudrate =
SPI_SPEED )

def get_thermo():
    cs.value( 0 )
    data = spi.read( 4 )
    cs.value( 1 )

    thermo_data = ( data[0] << 6 ) | ( data[1] >> 2 )

    if ( thermo_data > 0b011111_11111111 ):
        thermo_data = thermo_data - 16384

    thermo_temp = thermo_data * 0.25

    return thermo_temp

while True:
    temp = get_thermo()

    print( "Temperature: {}C".format( temp ) )

    time.sleep( 1 )
```

SPIの各端子のGPIOの番号

SPIのチャンネル番号

SPIの通信速度

ADコンバーターから値を取得して温度を求める関数

計算で温度を求める

前述の関数を呼び出して温度を取得する

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

39 センサー

フォトインターラプター

デジタル入力

プログラム photo-int.py

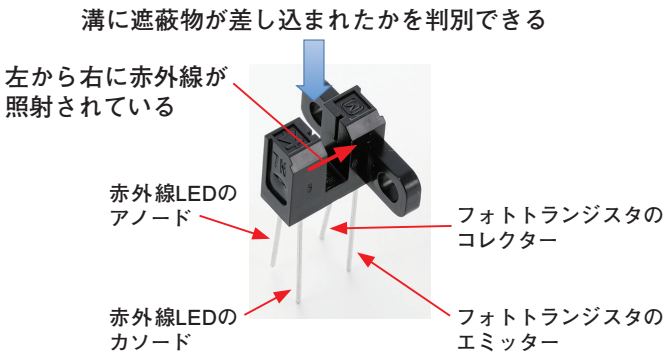
ライブラリ －

利用パーツ

●フォトインターラプター×1（秋：109668）
●抵抗器150Ω×1（秋：125151）、750kΩ×1（秋：125754）

「フォトインターラプター」は、赤外線を利用するセンサーモジュールです（**下図**）。モジュール上の溝を挟んで赤外線LEDと光センサー（フォトトランジスタ）が配置されており、この溝を通して赤外線が光センサーまで届くようになっています。溝に遮蔽物を差し込むと、赤外線が届かなくなり、光センサーの出力が変化します。

図　フォトインターラプターの外観



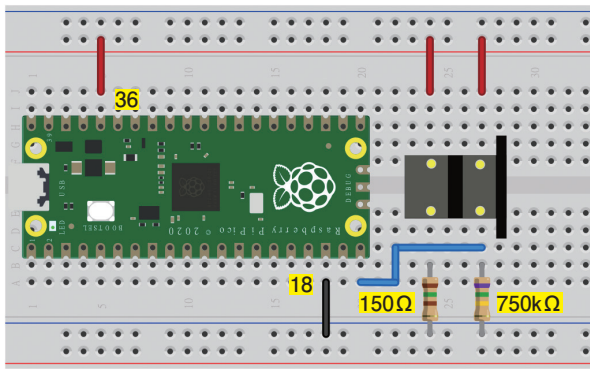
（日経BP撮影）

フォトインターラプターを使うと、例えばモーターが回転した回数を調べられます。モーターに穴が一つ開いた円盤を取り付けておきます。この円盤をフォトインターラプターの溝に差し込んでモーターを回転させると、穴が開いた部分が通過するたびに1回転したことを検出できます。

フォトインターラプターとPicoは**下図**のように接続します。フォトトランジスタの状態変化を調べるために、一つ前のフォトリフレクターと同じように抵抗器を接続して、フォトトランジスタと抵抗器の間の電圧を読み取ります。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　接続図



（Fritzingで作成）

遮蔽物が差し込まれているかどうかを判断するプログラム「photo-int.py」を以下に示します。遮蔽物が差し込まれていない場合は、赤外線がフォトトランジスタに届くため入力が「High」となり「None.」と表示します。遮蔽物を差し込むと、赤外線が届かなくなるため入力が「Low」となり「Block.」と表示します。

ソース　遮蔽物が差し込まれているかどうかを判断するプログラム

photo-int.py

```
from machine import Pin
import time

PINT_PIN = 15 ── エミッターに接続したGPIOの番号

sw = Pin( PINT_PIN, Pin.IN )

while True:
    value = sw.value() ── センサーの状態を読み取る
    if( value == 1 ):
        print ("None.")
    else:
        print ("Block.")

    time.sleep( 1 )
```

遮蔽物の有無を判別して表示する

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

40 センサー

カラーセンサー

I²C

プログラム color_sensor.py

ライブラリ apds9960.py

利用パーツ

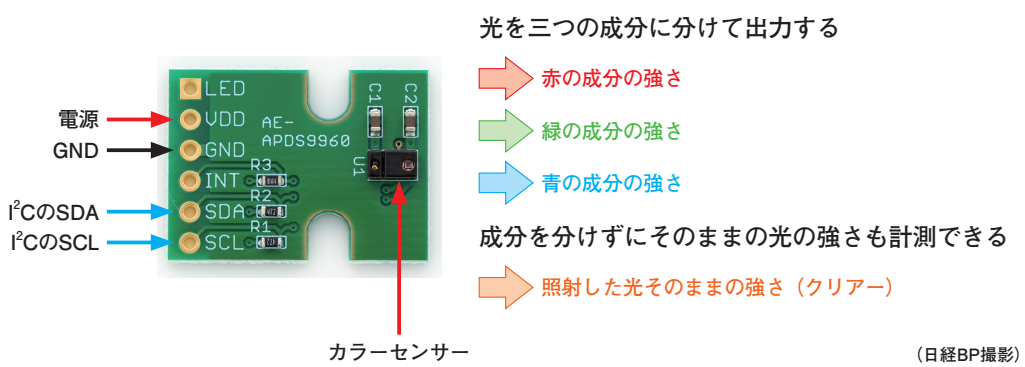
●カラーセンサー（秋：109754）

本書の第5章のNo.17とNo.18で紹介した「光センサー」（CdSセルやフォトトランジスタ）は、光の強さは分かるものの、どのような色の光が照射されているかは識別できません。光の色を知りたい場合には「カラーセンサー」が利用できます（**下図**）。カラーセンサーは、光の3原色である赤、緑、青の各成分を計測できる光センサー素子を搭載しており、受けた光を成分ごとに分けて計測できます。この計測結果から、光がどのような色であるかを判定可能です。例えば、赤と緑の成分が強ければ黄色であると分かります。

ここで紹介するカラーセンサー「APDS-9960」を搭載したモジュールは、中央に搭載したカラーセンサーチップで受けた光を3原色の成分ごとに計測できるほか、3原色に分けずにそのまま光の強さを計測する「クリアー」モードも搭載しています。

なお、APDS-9960は物体の動きを検知することができます。手をセンサーにかざして上下左右などに動かす事で、どちらに手を動かしたかが分かります。ただし、本書ではカラーセンサーとしての使い方を紹介し、動きの検知については説明しません。

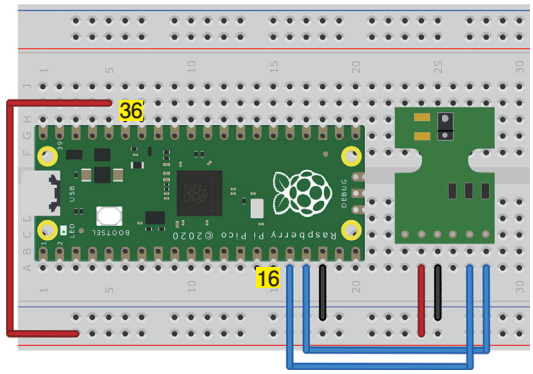
図　カラーセンサーの外観



（日経BP撮影）

APDS-9960は、I²C通信を使ってモジュールから計測結果を取得できます。I²Cアドレスは「0x39」です。Picoとは**下図**のように接続します。

図　接続図



（Fritzingで作成）

APDS-9960の制御には、本書のサポートサイトから入手できるライブラリ「apds9960.py」を利用します。ダウンロードして展開したら、本書の第2章の2-3の手順に従ってPicoに同じファイル名でライブラリを保存してください。

同ライブラリを使って、照射した光の成分を計測するプログラム「color_sensor.py」を以下に示します。ライブラリの「sensor.get_value()」関数を呼び出すことで、各色成分およびクリアーの計測値を取得できます。

物体の色を判別したい場合は、別途白色LEDを点灯して物体に照射し、その反射光をセンサーで読み取るようにします。

ソース　照射した光の成分を計測するプログラム

color_sensor.py

```
from machine import Pin, I2C
import time
import apds9960

APDS9960_ADDR = 0x39 ── I²Cアドレス

I2C_SDA = 12
I2C_SCL = 13 ┐── SDAとSCLのGPIO番号
I2C_CH = 0 ── I²Cチャンネル

i2c = I2C( I2C_CH, scl=Pin( I2C_SCL ), sda
=Pin( I2C_SDA ), freq=100000 )
i2c.scan()

sensor = apds9960.apds9960( i2c, APDS9960_
ADDR )

while True :
    ( red, green, blue, clear ) = sensor.
get_value() ── 各色の計測値を取得する

    print( "Red:{}  Green:{}  Blue:{}
Clear:{}".format( red, green, blue, clear ) )

    time.sleep( 1 )
```


@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

41 センサー

I²C

地磁気センサー

プログラム compass.py

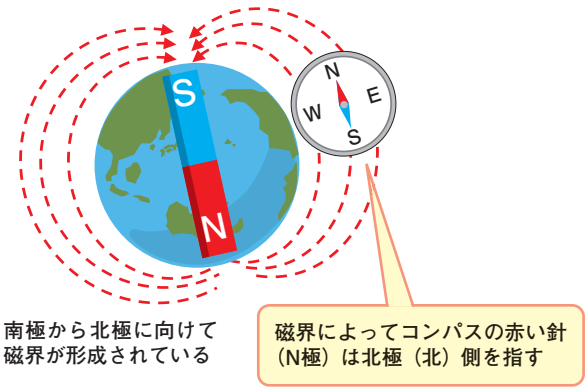
ライブラリ lis3mdl.py

利用パーツ

●地磁気センサー ×1（ス:6266、マ:M3446-6241）

地球は南極をN極、北極をS極とした大きな磁石のようになっており、南極から北極に向けて磁場が形成されています。このためコンパス（方位磁石）の針は、地球の磁場を受けてN極（赤い方）が北極方向、S極が南極方向を指し示し、その結果自分が現在どちらの方角を向いているかが分かります（**下図**）。

図　地球は南極から北極に向けて磁場が形成されている

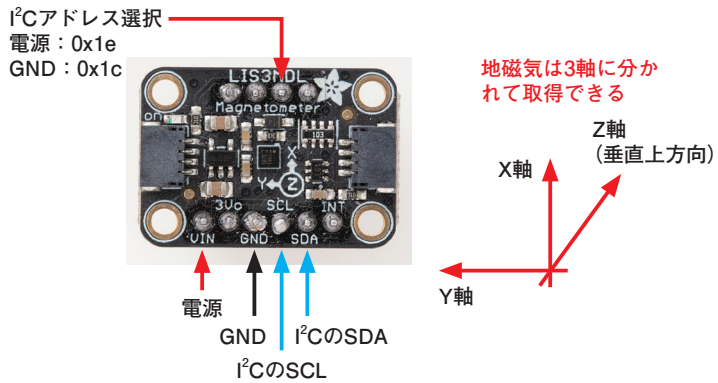


「地磁気センサー」は、この地球の磁場を計測できるセンサーです。磁場はX軸、Y軸、Z軸の成分に分けて計測されます。ここで紹介する地磁気センサー「LIS3MDL」は、各軸の向きが基板上に記載されています。**下図**のように置いた場合、上側がX軸、左側がY軸、基板表面に垂直な方向がZ軸となっています。

取得した磁場のX軸成分とY軸成分を利用することで、磁場がどちらの方向に向いているかを導き出せます。取得した値は、N極からS極に向けて正の値となっており、X軸とY軸の成分を合成して取得した方角が北向きとなります。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

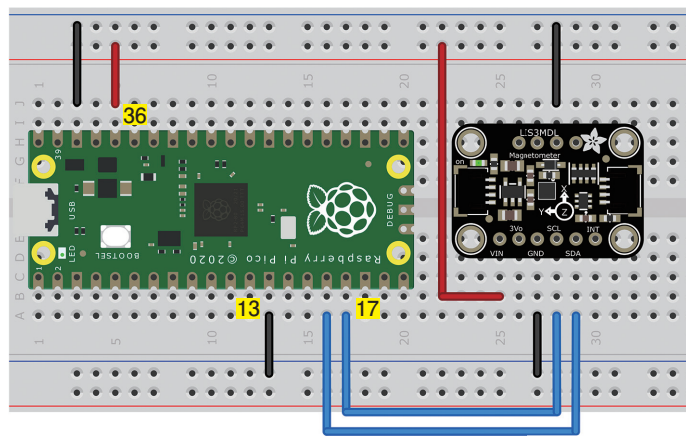
図　地磁気センサーの外観



（日経BP撮影）

LIS3MDLはI²CまたはSPIで通信可能です。ここではI²Cを利用する方法を説明します。I²Cアドレスは「0x1c」を使います。Picoとは**下図**のように接続します。

図　接続図



（Fritzingで作成）

LIS3MDLの制御には、本書のサポートサイトから入手できるライブラリ「lis3mdl.py」を利用します。ダウンロードして展開したら、本書の第2章の2-3の手順に従ってPicoに同じファイル名でライブラリを保存してください。

方位を精度高く計測するには、センサーの計測値に対して、現在地に合わせた補正を加える必要があります。そこで、まずは補正値を取得するプログラム「compass_calib.py」を実行します。compass_calib.pyは本書のサポートサイトから入手できます。同プログラムを実行すると、CALIB_X、CALIB_Yの補正値が表示されます。この状態で、水平の状態を保ちながらセンサーを数周回転させてください。CALIB_XとCALIB_Yが特定の値で安定したら、その補正値をメモしておきます。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

方位の計測プログラム「compass.py」を以下に示します。compass_calib.pyで取得した補正値をプログラム内の「CALIB_X」と「CALIB_Y」に設定してください。実行すると、方位が北を0度として0～360度の範囲で表示されます。

なお、ここで利用しているlis3mdl.pyライブラリに含まれる「sensor.get_mag()」関数を使うと、各軸の磁場の強さを値（単位はGauss：ガウス）で取得することも可能です。

ソース　方位を計測するプログラム

compass.py

```
from machine import Pin, I2C
import time
import lis3mdl

CALIB_X = 13 ┐
CALIB_Y = 9  ┘ 補正値を設定する

I2C_SDA = 12 ┐
I2C_SCL = 13 ┘ SDAとSCLのGPIO番号
I2C_CH = 0 ── I²Cのチャンネル

LIS3MDL_ADDR = 0x1c ── I²Cアドレス

MAG_GAIN = 4 ── ゲイン。4は±4Gauss、8は±8Gauss、12は±12Gauss、16は±16Gauss

i2c = I2C( I2C_CH, scl=Pin( I2C_SCL ), sda=Pin( I2C_SDA ), freq=100000)
i2c.scan()

sensor = lis3mdl.lis3mdl( i2c, LIS3MDL_ADDR )
sensor.set_gain( MAG_GAIN )
sensor.calib_x = CALIB_X
sensor.calib_y = CALIB_Y

while True:
    deg = sensor.get_compass_360() ── 方位を取得する
    print ( "Compass : {:.0f}deg".format( deg ) )

    time.sleep( 1 )
```

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

42 センサー

9軸モーションセンサー

I²C

プログラム 9axis.py
ライブラリ lsm9ds1.py

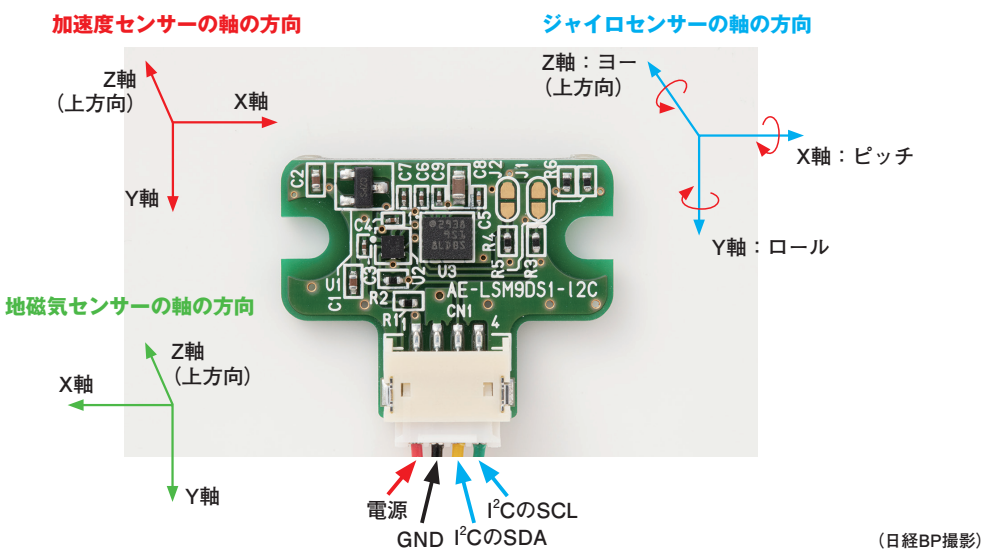
利用パーツ

●9軸モーションセンサー（秋：115381）

「9軸モーションセンサー」は、加速度、ジャイロ（角速度）、地磁気の三つをまとめて測れるセンサーです。加速度、ジャイロ、地磁気のそれぞれを測定できるセンサーは、ドローンなどの姿勢を調べる際によく使われます。加速度センサーでは、機体がどの程度傾いたかや移動速度が分かります。ジャイロセンサーは機体の回転を計測できるので、向きや傾きを調べられます。地磁気センサーは、地球の磁場を計測して向いている方向を取得できます。これらの情報を統合することで、機体の姿勢を正確に把握して制御できます。

ここで紹介する9軸モーションセンサー「LSM9DS1」は、X軸、Y軸、Z軸またはロール、ヨー、ピッチの要素に分けて加速度、ジャイロ、地磁気それぞれのセンサーの計測値が出力されます（**下図**）。このように3種類の計測値がそれぞれ3軸／3要素に分けて出力されることから、9軸モーションセンサーあるいは単に9軸センサーと呼ばれています。

図　9軸モーションセンサーの外観
地磁気センサーのみ図の左方向がX軸の正の向きとなっている点に注意。

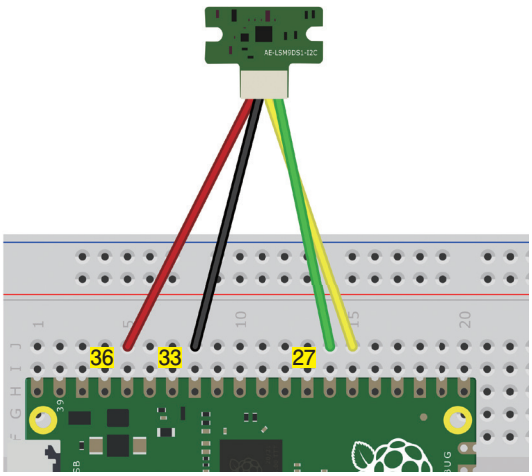


（日経BP撮影）

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

LSM9DS1から各計測値を取得するには、I²Cを使って通信します。Picoとの接続には**下図**のようにセンサーに付属するコネクタ付きケーブルを用いて接続します。

図　接続図



（Fritzingで作成）

LSM9DS1の計測には、本書のサポートサイトから入手できるライブラリ「lsm9ds1.py」を利用します。ダウンロードして展開したら、本書の第2章の2-3の手順に従ってPicoに同じファイル名でライブラリを保存してください。

加速度、地磁気、ジャイロの各計測値を表示するプログラム「9axis.py」を以下に示します。「sensor.get_accel()」で加速度、「sensor.get_gyro()」でジャイロ、「sensor.get_mag()」で地磁気の計測値を取得できます。

プログラム中のACCEL_RANGE、GYRO_SCALE、MAG_GAINの値を変えることで、各センサーの感度を調節可能です。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

ソース　加速度、地磁気、ジャイロの各計測値を表示するプログラム

9axis.py

```
from machine import Pin, I2C
import time
import lsm9ds1

I2C_SDA = 20 ┐
I2C_SCL = 21 ┘ SDAとSCLのGPIO番号
I2C_CH = 0 ── I²Cのチャンネル

LSM9DS1_ACCEL_ADDR = 0x6a ── 加速度、ジャイロセンサーのI²Cアドレス
LSM9DS1_MAG_ADDR = 0x1c ── 地磁気センサーのI²Cアドレス

ACCEL_RANGE = 2 ── 加速度センサーのレンジ。2は±2G、4は±4G、16は±16G
GYRO_SCALE = 245 ── ジャイロセンサーのレンジ。245は±245dps、500は±500dps、
MAG_GAIN = 4 ┐      2000は±2000dps
    地磁気センサーのゲイン。4は±4Gauss、8は±8Gauss、12は±12Gauss、16は±16Gauss
i2c = I2C( I2C_CH, scl=Pin( I2C_SCL ), sda=Pin( I2C_SDA ), freq=100000)
i2c.scan()

sensor = lsm9ds1.lsm9ds1( i2c, LSM9DS1_ACCEL_ADDR, LSM9DS1_MAG_ADDR )

sensor.set_accel_range( ACCEL_RANGE )
sensor.set_gyro_scale( GYRO_SCALE )
sensor.set_mag_gain( MAG_GAIN )

while True:
    ( accel_x, accel_y, accel_z ) = sensor.get_accel() ── 加速度を取得する
    ( gyro_x, gyro_y, gyro_z ) = sensor.get_gyro() ── 角速度を取得する
    ( mag_x, mag_y, mag_z ) = sensor.get_mag() ── 地場の強さを取得する

    print( "Accelerometer  X:{:.2f}G Y:{:.2f}G Z:{:.2f}G".format( accel_
x, accel_y, accel_z ) )
    print( "Gyroscope      P:{:.2f}dps R:{:.2f}dps Y:{:.2f}dps".format
( gyro_x, gyro_y, gyro_z ) )
    print( "Magnetic       X:{:.2f}Gauss Y:{:.2f}Gauss Z:{:.2f}Gauss".
format( mag_x, mag_y, mag_z ) )
    print( "--------------------" )

    time.sleep( 0.5 )
```


@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

43 センサー

曲げセンサー

アナログ入力

プログラム bend.py

ライブラリ －

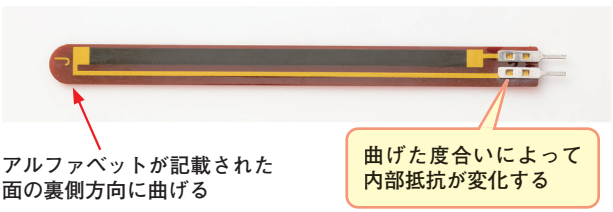
利用パーツ

●曲げセンサー×1（秋：116410） ●抵抗器10kΩ×1（秋：125103）

「曲げセンサー」は、取り付けた物体がどれくらい曲げられたかを計測できるセンサーです。フィルム内に抵抗素子が入っており、曲げることで抵抗値が変化します。この抵抗値を計測することで曲がり度合いを把握できる仕組みです。例えば、ロボットの腕の関節などに曲げセンサーを取り付けておけば、どの程度腕を曲げているかが分かります。

ここで紹介する曲げセンサーは、片面にアルファベットが記載されています（**下図**）。この面とは逆方向に曲げることで内部抵抗値が変化します。伸ばした状態では内部抵抗は小さく、曲げていくことで内部抵抗が徐々に増えていきます。

図　曲げセンサーの外観

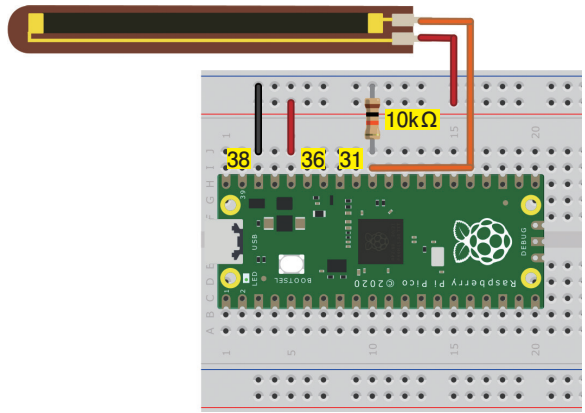


（日経BP撮影）

曲げセンサーとPicoは**下図**のように接続します。曲げセンサーの内部抵抗値を読み取るには、図中に示したように抵抗器を直列に接続して分圧回路を作り、抵抗器にかかる電圧をPicoのアナログ入力で読み取ります（分圧回路については本書の第4章の4-4を参照）。ここで示した回路では、伸ばした状態では電圧が高く、曲げることで電圧が低くなります。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　接続図



（Fritzingで作成）

曲げた度合いを調べるプログラム「bend.py」を以下に示します。このプログラムは、曲げた度合いを割合で表示します。そのために必要となるセンサーの校正情報を、BEND_MIN変数とBEND_MAX変数にあらかじめ設定しておく必要があります。

校正情報は、次のように取得します。bend.pyプログラムを実行すると、Valueに曲げセンサーの計測値が表示されます（配布したプログラムにはMIND_MINとBEND_MAX変数に値が指定してありますが、そのままの状態で実行して問題ありません）。この値は1秒間隔で逐次更新されるようになっています。実行したら、(1) 曲げセンサーをまっすぐにした状態と、(2) 180度曲げた状態のそれぞれの値を読み取ります。この値が校正情報となります。必ずメモしておいてください。

校正情報を取得できたら、bend.pyのBEND_MIN変数とBEND_MAX変数の値を、取得した校正情報で書き換えます。具体的には、曲げセンサーをまっすぐにした状態の値をBEND_MIN変数に、180度曲げた状態の値をBEND_MAX変数に設定します。プログラムを再度実行すると、曲げた度合いが正しく表示されるようになります。

なお、180度曲げる際は、センサーに折り目が付く形で折り曲げないよう注意してください。端同士を重ねてループ形状にして計測します（このため校正しても曲げ度合いはおおまかな値となります）。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

ソース　曲げた度合いを表示するプログラム

bend.py

```
import machine
import time

BEND_MIN = 18692 ── 伸ばした状態の値(取得した校正情報で書き換える)
BEND_MAX = 2192 ── 180度曲げた状態の値(取得した校正情報で書き換える)

VREF = 3.3 ── 電源電圧
ADC_PIN = 0 ── センサーを接続したアナログ入力の番号

analog = machine.ADC( 0 )

while True:
    value = analog.read_u16() ── 計測値を取得する
    ratio = 1 - ( value - BEND_MAX ) / ( BEND_MIN - BEND_MAX )
    ratio = round ( ratio * 100 ) ── 曲げた度合を求める

    if ( ratio < 0 ):
        ratio = 0
    if ( ratio > 100 ):
        ratio = 100

    print ( "Ratio: {}%  Value :{}".format( ratio, value )  )

    time.sleep( 1 )
```

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

44 センサー

圧力センサー

アナログ入力

プログラム pressure.py

ライブラリ －

利用パーツ

●圧力センサー×1（秋：104003） ●抵抗1kΩ×1（秋：125102）

「圧力センサー」は、指などで押されたかを判断できるセンサーです（**下図**）。指で押すだけでなく、人や物体がセンサー上に乗ることで判定するのにも利用できます。どの程度の力で押されているかまで判断できるため、軽く触れた程度の力の場合は、押されていないと判断することもあります。

図　圧力センサーの外観

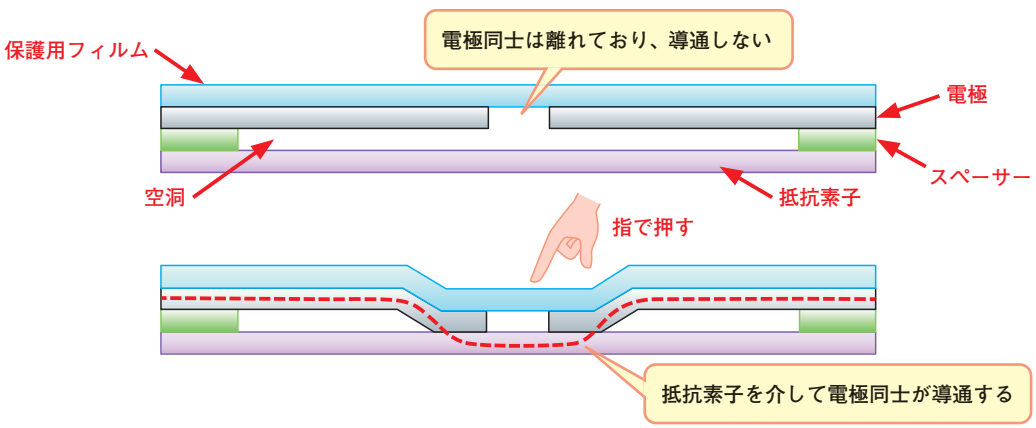


指などで押した強さによって内部抵抗が変化する

（日経BP撮影）

圧力センサーは**下図**のように、抵抗素子の上にすき間を挟んで二つの電極シートが配置された構成になっています。上から指で圧力を加えると、離れていた電極が抵抗素子を介して接続され、電気が流れるようになります。圧力の大小によって電極が触れる面積が変わるため、抵抗値が変化します。

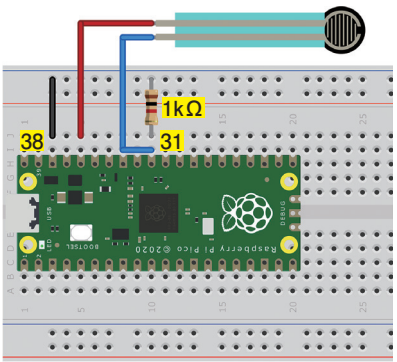
図　圧力センサーの仕組み



@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

圧力センサーの内部抵抗を読み取るには、**下図**のように抵抗を直列に接続して分圧回路を作ります。接続した抵抗にかかる電圧をPicoのアナログ入力で読み取ります。センサーを押していないと電圧は0Vとなり、押す力によって電圧が上昇します。

図　接続図



（Fritzingで作成）

押されているかを調べるプログラム「pressure.py」を以下に示します。プログラムの「TH」には、押されているかを判断するしきい値（0～65535）を指定します。

プログラムを実行し、センサーを押していない状態では「Free」と表示されます。センサーを指などで押し、しきい値以上の力が加わると「Press」と表示されます。なお、センサーの計測値も表示されるため、この値を確認してしきい値を決めると良いでしょう。

ソース　圧力センサーを押したかを判断するプログラム

pressure.py

```
from machine import Pin, ADC
import time

SENSOR_PIN = 0 ── センサーを接続したアナログ入力の番号

TH = 10000 ── 押したと判断するしきい値（0 ～ 65535）

analog = ADC( SENSOR_PIN )

while True:
    value = analog.read_u16() ── 計測した値を取得する

    if ( value > TH ): ── しきい値よりも高いかを調べる
        print( "Press    Value:{}".format( value ) ) ── 押されている場合に表示
    else:
        print( "Free     Value:{}".format( value ) ) ── 押されていない場合に表示

    time.sleep( 0.5 )
```

45 センサー

UART

ドップラーセンサー

プログラム doppler.py

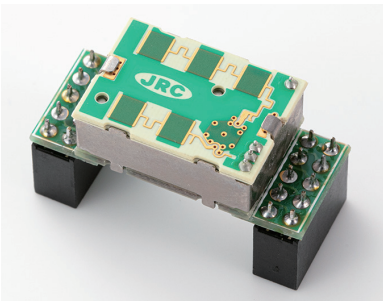
ライブラリ ―

利用パーツ

●ドップラーセンサー ×1（秋：107776）

「ドップラーセンサー」は、人や物体が近づいたり遠ざかったりする動きを検知できるセンサーです（**下図**）。後ろから車が近づいているのを検出したら注意を促すといった使い方ができます。

図　ドップラーセンサーの外観



（日経BP撮影）

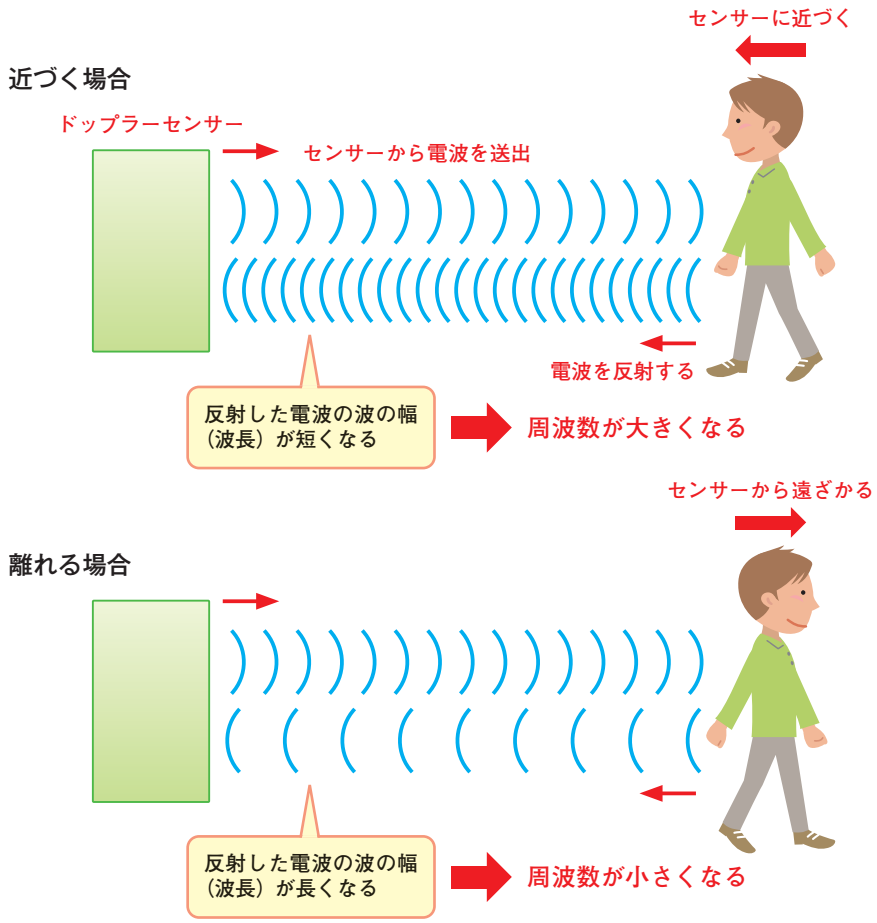
ドップラーセンサーは、電波を送出しています。1秒間に発生する電波の波の数を周波数と呼び、Hz（ヘルツ）という単位で表します。ここで紹介するドップラーセンサーは周波数24GHzの電波を送出しています。

ドップラーセンサーから送出された電波は、対象物に当たると反射してセンサーに戻ってきます。この際、対象物がセンサーに近づく方向に動いていると、センサーが受け取る1秒当たりの波の数が多くなる、つまり、周波数が高くなります（**下図**）。

一方、センサーから離れる方向に動くと、1秒当たりの波の数が少なくなり、周波数が低くなります。この周波数の変化を見ることで、近づいているか離れているかを判断できます。これがいわゆる「ドップラー効果」で、救急車が近づいてくる場合にサイレンが高い音（周波数が高い）で聞こえ、遠ざかると低い音（周波数が低い）で聞こえるのと同じ原理です。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

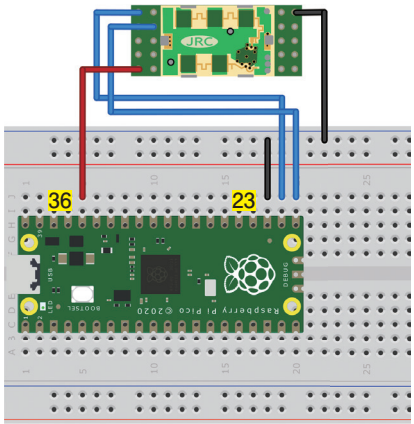
図　ドップラー効果を用いて物体が近づいているか遠ざかっているかを判別する



ドップラーセンサーとPicoは**下図**のように接続します。ここで利用するドップラーセンサーには基板とピンソケットが付属しており、あらかじめはんだ付けしておく必要があります。特にセンサー本体と基板のはんだ付けについては端子間の間隔が狭いため、はんだでつながってしまう「ブリッジ」にならないよう注意してください。はんだ付けが終わったら、オス-オス型ジャンパー線をドップラーセンサーのピンソケットに直接差し込んで接続します。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　接続図



（Fritzingで作成）

物体が近づいているか遠ざかっているかを判別するプログラム「doppler.py」を以下に示します。ここで利用するドップラーセンサーとは、UARTで通信して計測結果を取得できます。取得した値が「67」の場合は物体がセンサーに近づいており、「76」の場合は遠ざかっていると判別できます。

ソース　物体が近づいているか遠ざかっているかを判別するプログラム

doppler.py

```
from machine import Pin, UART
import time

UART_CH = 0 ── UARTのチャンネル番号
UART_TX = 16 ── UARTのTxDのGPIO番号
UART_RX = 17 ── UARTのRxDのGPIO番号

uart = UART( UART_CH, baudrate=9600, tx=Pin( UART_TX ), rx=Pin( UART_RX ) )

while( True ):
    buf = bytes()
    flag = False
    while( uart.any() > 0 ):
        buf += uart.read(1)
        flag = True

    if ( flag == True ):
        if ( len(buf) > 1 ):
            ans = int( buf[1] )
            if( ans == 67 ):
                print("Approach.")  ── 値が67(近づいている)の場合に表示する
            elif( ans == 76 ):
                print( "Move away." )  ── 値が76(遠ざかっている)の場合に表示する
    time.sleep( 0.1 )
```


@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

46 センサー

赤外線距離センサー

I²C

プログラム ir_length.py

ライブラリ gp2y0e03.py

利用パーツ

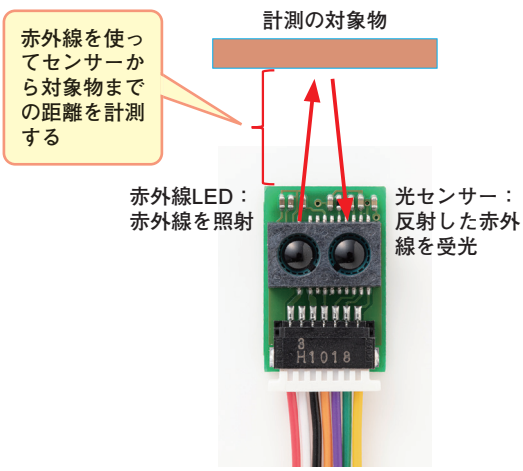
●赤外線距離センサー ×1（秋：107547）

「赤外線距離センサー」は、センサーから対象物までの距離を赤外線を使って計測するセンサーです。本書の第5章のNo.23で紹介した超音波距離センサーと同じような使い方ができます。サイズが小さな対象物でも距離を測りやすいことが、超音波距離センサーよりも優れている点です。

赤外線距離センサーは、対象物に照射した赤外線の反射光を受光して距離を計測します（**下図**）。ここで紹介する赤外線距離センサー「GP2Y0E03」は、4～50cmの距離の計測が可能です。二つのレンズが取り付けられており、一方が赤外線を照射し、もう一方が反射した赤外線を受光します。

GP2Y0E03が計測した距離データをPicoで取得するにはI²C通信を使います。I²Cアドレスは「0x40」です。このセンサーは、I²Cで計測値を送るだけでなく、計測した距離に対応する電圧を2番端子から出力します。この電圧をPicoのADコンバーターで読み取って計算することで距離を求める方法もありますが、ここではI²C通信で取得する方法を説明します。

図　赤外線距離センサーの外観



（日経BP撮影）

端子の用途

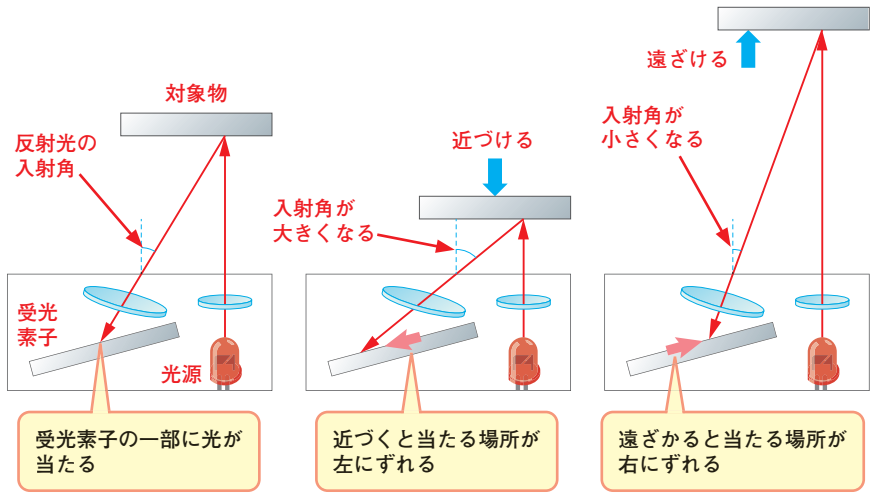
番号	線色	用途
1	赤	電源
2	白	出力（電圧変化）
3	黒	GMD
4	橙	I/O電圧
5	紫	アクティブの切り替え
6	緑	I²CのSCL
7	黄	I²CのSDA

I²Cアドレス：0x40

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

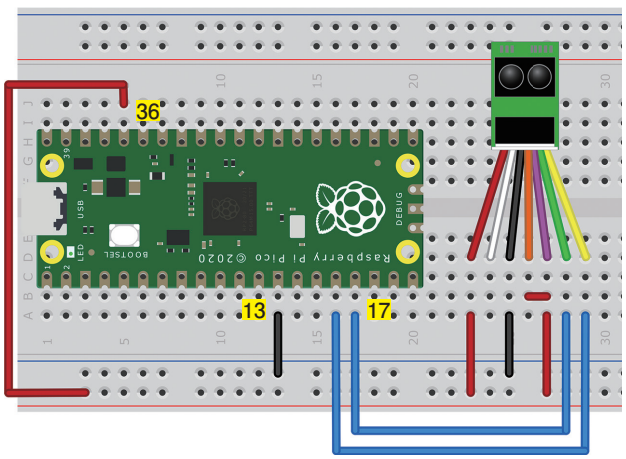
GP2Y0E03は、「PSD（Position Sensing Detector）」と呼ぶ方式で距離を計測します（**下図**）。PSD方式は、対象物との距離によって、反射光のレンズへの入射角度が変わることを利用します。このとき、内部に配置された光センサーにレンズで集束された光が当たる位置も変わります。下図に示した配置では、対象物が近いほどセンサーの左寄りに、遠いほど右寄りに光が当たります。この当たった位置を計測して、対象物までの距離を求める仕組みです。

図　赤外線距離センサーの仕組み



GP2Y0E03とPicoは**下図**のように配線します。GP2Y0E03は、付属するコネクタ付きケーブルを使ってブレッドボードに取り付けます。

図　接続図



（Fritzingで作成）

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

GP2Y0E03を制御するには、本書のサポートサイトから入手できるライブラリ「gp2y0e03.py」を利用します。ダウンロードして展開したら、本書の第2章の2-3の手順に従ってPicoに同じファイル名でライブラリを保存してください。

同ライブラリを使って、赤外線距離センサーで対象物までの距離を計測するプログラム「ir_length.py」を以下に示します。ライブラリの「sensor.measure()」関数を呼び出すことで、対象物までの距離をcm単位で取得します。

ソース　赤外線距離センサーで対象物までの距離を計測するプログラム

ir_length.py

```
from machine import Pin, I2C
import time
import gp2y0e03

I2C_SDA = 12
I2C_SCL = 13
I2C_CH = 0

GP2Y0E03_ADDR = 0x40

i2c = I2C( I2C_CH, scl=Pin( I2C_SCL ), sda=Pin( I2C_SDA ), freq=100000)
i2c.scan()

sensor = gp2y0e03.gp2y0e03( i2c, GP2Y0E03_ADDR )

while True:
    distance = sensor.measure()
    print ( "Discance : {:.1f} cm".format( distance ) )
    time.sleep( 1 )
```

SDAとSCLのGPIO番号

I²Cのチャンネル

I²Cアドレス

距離を取得する

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

47 センサー

近接センサー

I²C

プログラム approach.py
ライブラリ vcnl4010.py

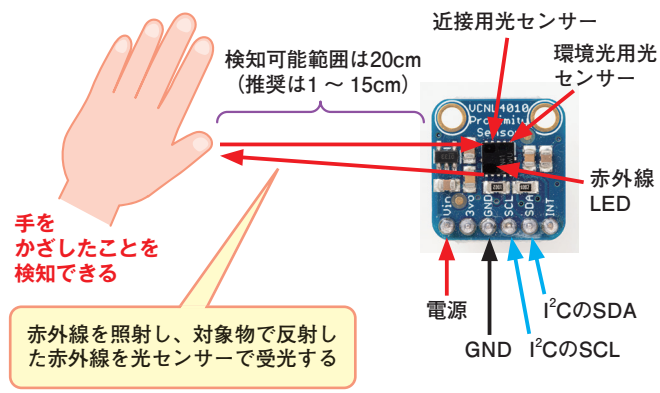
利用パーツ

●近接センサー ×1（ス：2640）

「近接センサー」は、手などをかざしたかどうかを検知できるセンサーです。十数cmの距離以内の範囲にかざした場合のみ反応します。例えば、手をかざすだけで水や石けん水を流す装置などに利用できます。蛇口のレバーに触れずに済むため衛生的に使えます。

ここで紹介する近接センサー「VCNL4010」は、約20cm以内に対象物があるかどうかを調べられます（**下図**）。赤外線を照射し、対象物で反射した赤外線を光センサーで受けることで、手がかざされたかどうかを検知します。環境光を計測する光センサーも搭載しており、周囲の明るさも計測できます。

図　近接センサーの外観

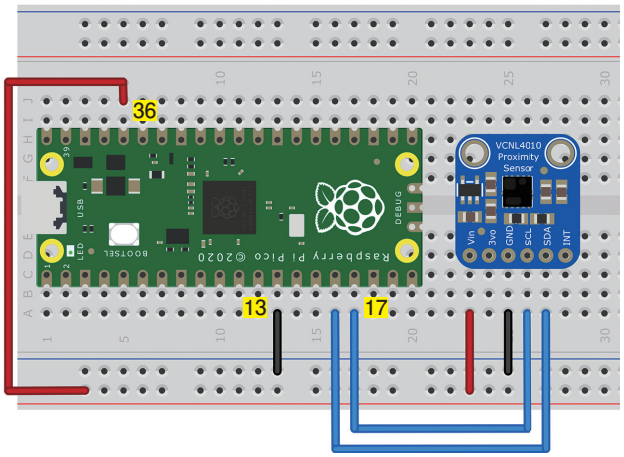


（日経BP撮影）

PicoでVCNL4010の計測値を受信するには、I²C通信を使います。I²Cアドレスは「0x13」です。Picoとは**下図**のように接続します。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

図　接続図



（Fritzingで作成）

VCNL4010を制御するには、本書のサポートサイトから入手できるライブラリ「vcnl4010.py」を利用します。ダウンロードして展開したら、本書の第2章の2-3の手順に従ってPicoに同じファイル名でライブラリを保存してください。

同ライブラリを使って、手などをかざしたかどうかを検知するプログラム「approach.py」を以下に示します。このプログラムは、センサーからの入力値が「TH」変数で設定した値以上になったら手が近づいたと見なし、「Approach.」と表示します。THの初期値として4000を代入していますが、各自の環境に合わせて調節可能です。まず1回プログラムを実行し、手をかざして近づいたと検知させたい位置まで近づけていき、表示された値をTHに設定してください。

ソース　手などをかざしたかどうかを検知するプログラム

approach.py

```
from machine import Pin, I2C
import vcnl4010
import time

TH = 4000 ── 近接したかを判断するしきい値

I2C_SDA = 12 ┐
I2C_SCL = 13 ┘── SDAとSCLのGPIO番号
I2C_CH = 0 ── I²Cのチャンネル

VCNL4010_ADDR = 0x13 ── I²Cアドレス

i2c = I2C( I2C_CH, scl=Pin( I2C_SCL ),
 sda=Pin( I2C_SDA ), freq=100000)
i2c.scan()

sensor = vcnl4010.vcnl4010( i2c,
VCNL4010_ADDR )

while True:
    value = sensor.read_proximity()
                                 計測値を取得する
    if( value >= TH ):
        message = "Approach. "
                                 近接したかを判断する
    else:
        message = ""

    print ( "{}Value : {}".format
( message, value ) )
    time.sleep( 1 )
```

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

48 センサー　　シリアル通信

ロードセル

プログラム　weight.py

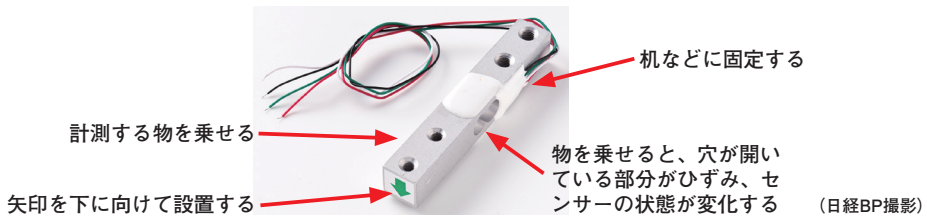
ライブラリ　https://github.com/SergeyPiskunov/micropython-hx711/

利用パーツ

●ロードセル×1（秋：113042）　●ロードセル用ADコンバーター ×1（秋：112370）
●タクトスイッチ×1（秋：103647）

「ロードセル」は、重さを計測できるセンサーモジュールです（**下図**）。中央付近にひずみセンサーが貼り付けてあります。物体をロードセルに乗せると、中央の穴の部分でひずみが生じ、ひずみセンサーの内部抵抗値が変化します。この内部抵抗値の変化を読み取ることで重さを計測できます。ロードセルに取り付けられた赤と黒の配線を電源に接続すると、白と緑の配線に電圧が生じます。この電圧を読み取ることで重さが分かります。

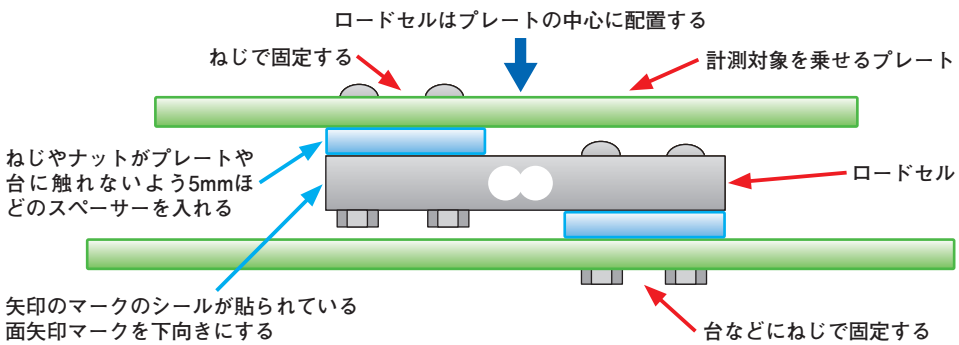
図　ロードセルの外観



（日経BP撮影）

ロードセルは、記載されている矢印が下向きになるように配置し、底面を台座などに固定します。この状態で天面に物を乗せることで、重さを計測できます。設置例を**下図**に示します。

図　ロードセルの設置例

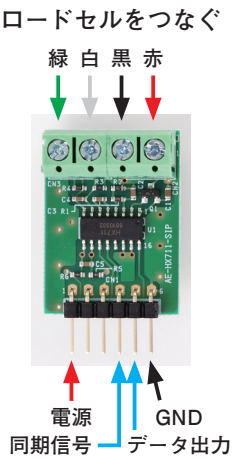


@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

ロードセルは製品によって計測できる重さの範囲が異なるため、計測したい対象物に合わせて利用するロードセルを選択する必要があります。ここでは、10kgまで計測可能なロードセルを使う場合を紹介します。

ロードセルは、出力する電圧の変化が非常に小さいため、Picoが搭載するADコンバーターでは変化を十分には読み取れません。そこで、別途ロードセル用のADコンバーターを用意し、これを介してロードセルが出力した電圧を読み取ります。ここで利用するロードセル用ADコンバーター「HX711」は分解能が24ビットと高く、非常に小さな電圧の変化も読み取り可能です（**下図**）。

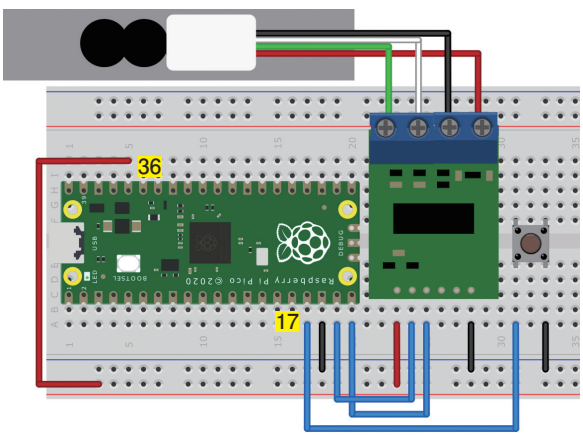
図　ロードセル用ADコンバーターの外観



（日経BP撮影）

ロードセルおよびロードセル用ADコンバーターとPicoは**下図**のように接続します。計測値を0にリセットするためのタクトスイッチも接続しておきます。

図　接続図



（Fritzingで作成）

上記構成で重さを計測するには、Sergey Piskunov氏が公開しているHX711用のライブラリ「hx711.py」を利用するのがお手軽でオススメです。GitHubにある同ライブラリのページ（https://github.com/SergeyPiskunov/micropython-hx711/）にアクセスし、ページ内の「Code」-「Download ZIP」の順にクリックしてダウンロードします（より詳細な手順は本書の第5章のNo.3「シリアルLED」の記事を参照）。ダウンロードしたファイルを展開し、格納されている「hx711.py」をPicoに保存してください（ライブラリをPicoに保存する方法は本書の第2章の2-3を参照）。

同ライブラリを利用し、重さを計測するプログラム「weight.py」を以下に示します。プログラムを実行したら、まずタクトスイッチを押してください。すると、重さが0にリセットされます。その後、ロードセルに計測対象物を乗せると重さが表示されます。

ソース　重さを計測するプログラム

weight.py

```
from machine import Pin
import time
from hx711 import HX711

DATA_PIN = 14
CLOCK_PIN = 15
TARE_PIN = 13

TARE_TIMES = 10
MEASURE_TIMES = 2

RATED_VOLT = 0.001
RATED_WEIGHT = 20000

VDD = 3.3
GAIN = 128
RESO_BIT = 24

sw = Pin( TARE_PIN, Pin.IN, Pin.PULL_UP )
tare = 0

ADC = HX711( d_out = DATA_PIN, pd_sck = CLOCK_PIN )

while True:
    if( sw.value() == 0 ):
        count = 0
        sum = 0
        while ( count < TARE_TIMES ):
            value = ADC.read()
            sum = sum + value
            count = count + 1

        tare = sum / TARE_TIMES

        while( sw.value() == 0 ):
            time.sleep( 0.1 )
        print( "Tare set. {:.0f}".format( tare ) )
    count = 0
    sum = 0
    while ( count < MEASURE_TIMES ):
        value = ADC.read()
        sum = sum + value
        count = count + 1
        time.sleep( 0.1 )

    value_ave = sum / MEASURE_TIMES
    volt = ( value_ave - tare ) * VDD / ( 2 ** RESO_BIT ) / GAIN
    weight = volt / ( RATED_VOLT * VDD / RATED_WEIGHT )

    print( "Weight : {:.1f}g".format( weight ) )
```

同期信号端子のGPIO番号

データ出力端子のGPIO番号

タクトスイッチのGPIO番号

ADコンバーターから値を取得して重さを算出する

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

49 センサー

ガスセンサー

アナログ入力

プログラム smell.py

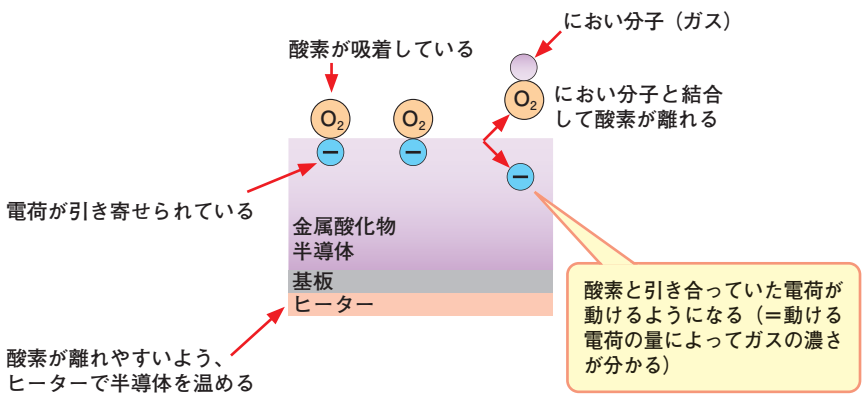
ライブラリ ー

利用パーツ

●においセンサー ×1（秋:100989） ●トランジスタ×2（秋:106477）
●抵抗器27Ω×1（秋:114273）、100Ω×1（秋:125101）、1kΩ×2（秋:125102）

「ガスセンサー」を使うと、空気中に含まれる特定の種類のガスを検出できます。ここで利用するガスセンサー「TGS2450」は、メチルメルカプタンや硫化水素といった口臭の主な原因となるガス成分（硫黄化合物系ガス）を検出可能な金属酸化物半導体を使ったガスセンサーです（**下図**）。ガスセンサーの周りにこれら対応するガスがあると、金属酸化物半導体に吸着している酸素がガスによってはぎ取られます。その結果、酸素に引き寄せられていた金属酸化物半導体内の電荷が自由に動けるようになって内部抵抗値が下がります。この変化を計測することでガスを検出できる仕組みです。

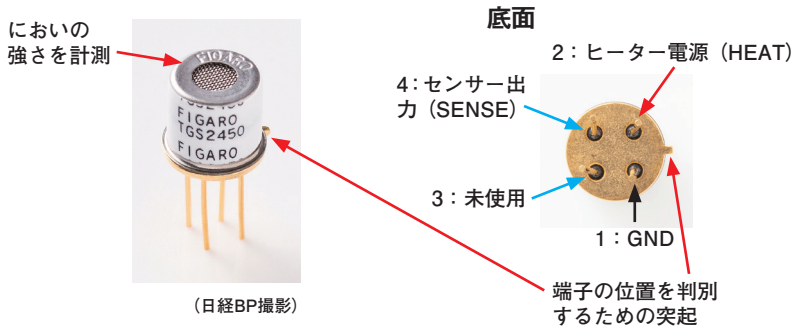
図　ガスセンサーの仕組み



TGS2450は下面に四つの端子を備えています（**下図**）。1番端子がGND、2番端子が金属酸化物半導体を温めるためのヒーター用電源端子、4番端子がセンサー出力端子となっています（3番端子は未使用）。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

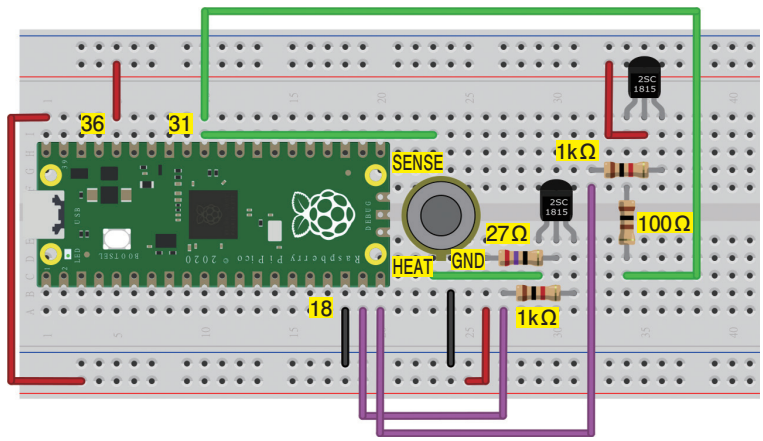
図　TGS2450の外観と各端子の役割



（日経BP撮影）

TGS2450とPicoは**下図**のように接続します。検出に当たっては、金属酸化物半導体から酸素が離れやすくなるように事前にヒーターで加熱する処理が必要です。このヒーターの温め処理と計測処理は同時には実行できません。そこで、トランジスタを利用してヒーターの温め処理と計測処理を切り替えられるようにしています。

図　接続図



（Fritzingで作成）

TGS2450を使ってにおいを計測するプログラム「smell.py」を以下に示します。ガスセンサーの計測結果はアナログ入力への電圧値として取得できるので、これを「volt」変数に格納しています。この値が所定の値（TH）よりも小さい場合は、においが強いと判断して「BAD.」と表示します。

THをどの程度にするかは、検出対象のガスがない状態でプログラムを実行して通常時の電圧を記録しておきます。続いて、口臭がすると感じる息をガスセンサーに吹きかけて電圧を記録します。通常時の電圧と息を吹きかけたときの電圧の中央値をTHに設定してください。

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

ソース　においを計測するプログラム

smell.py

```
import machine
import time

TH = 2.0 ── においが強いと判断するしきい値

HEATER_PIN = 14 ┐
SENSOR_PIN = 15 ┘── センサー制御用のGPIO番号
ADC_PIN = 0 ── センサーに接続したアナログ入力の番号
VREF = 3.3 ── 電源電圧

analog = machine.ADC( ADC_PIN )

heater = machine.Pin( HEATER_PIN, machine.Pin.OUT )
sensor = machine.Pin( SENSOR_PIN, machine.Pin.OUT )
heater.value( 0 )
sensor.value( 0 )

while ( True ):
    i = 0 ┐
    while ( i < 5 ):                    │
        time.sleep( 0.242 )             │
        heater.value( 1 )               ├── ヒーターでセンサーを温める
        time.sleep( 0.008 )             │
        heater.value( 0 )               │
        i = i + 1 ┘
    time.sleep( 0.237 )
    sensor.value( 1 ) ── センサーの状態を読み取れるようにする
    time.sleep( 0.0025 )

    value = analog.read_u16() ┐
    volt = value / 65536 * VREF ┘── センサーの計測値を取得する
    if( volt > TH ):                    ┐
        print ( "OK. Volt: {:.2f} V".format( volt ) )   ├── においが強いかを判別する
    else:                               │
        print ( "BAD. Volt: {:.2f} V".format( volt ) ) ┘

    time.sleep( 0.0025 )
    sensor.value( 0 ) ── センサーの読み取りを無効にする

    time.sleep( 1 )
```

50 センサー

UART

CO2センサー

プログラム co2.py

ライブラリ ー

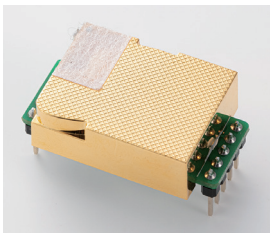
利用パーツ

●CO2センサー×1（秋：116142）

「CO2センサー」は、二酸化炭素（CO2）の濃度を計測できるセンサーです（**下図**）。冬場は窓を開けることが少なくなるため、室内のCO2濃度が高くなりがちです。そこで、CO2センサーでその濃度を計測し、所定の濃度を超えたら換気を促すといった使い方ができます。

ここで紹介するCO2センサー「MH-Z19C」は、400～5000ppmのCO2濃度を計測可能です。Picoで計測値を読み取るにはUARTを使います。

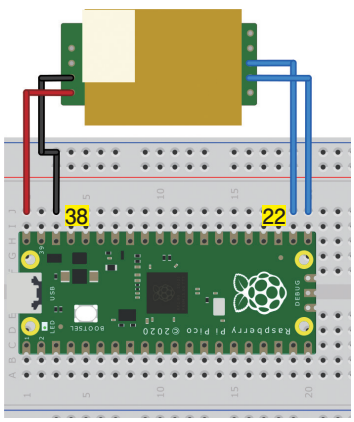
図　CO2センサーの外観



（日経BP撮影）

MH-Z19Cは、一般的なブレッドボードと端子間隔が異なっているため直接差し込めません。そこで、**下図**のようにオス-メス型ジャンパー線を使ってブレッドボードと接続します。

図　接続図



（Fritzingで作成）

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.

CO2濃度の計測プログラム「co2.py」を以下に示します。PicoとMH-Z19CはUARTの0チャンネルを使って通信します。MH-Z19Cに対してプログラムで所定の文字列を送り込むことでセンサーの設定や計測値を取得できます。計測値を取得するには、16進数で「ff」「01」「86」「00」「00」「00」「00」「00」「79」と文字列を送ることで値が返ってきます。取得した値の3文字目（s[2]）と4文字目（s[3]）が計測値です。3文字目を256倍して4文字目を足すことで、CO2濃度を求められます。

ソース　CO2濃度を計測するプログラム

co2.py

```
from machine import Pin, UART
import time

AUTO_CALIB = True

UART_CH = 0 ── UARTのチャンネル番号
UART_TX = 16 ── UARTのTxDのGPIO番号
UART_RX = 17 ── UARTのRxDのGPIO番号

uart = UART( UART_CH, baudrate=9600, tx=Pin( UART_TX ), rx=Pin( UART_RX ) )

if ( AUTO_CALIB == True ):
    uart.write( b'¥xff¥x01¥x79¥xa0¥x00¥x00¥x00¥x00¥xe6' )
else:
    uart.write( b'¥xff¥x01¥x79¥x00¥x00¥x00¥x00¥x00¥x86' )

time.sleep(1)

while( True ):
    uart.write(b'¥xff¥x01¥x86¥x00¥x00¥x00¥x00¥x00¥x79')
    s = uart.read(9)
    if len(s) >= 4 and s[0] == 0xff and s[1] == 0x86:
        print(f'{s[2] * 256 + s[3]} ppm')
    else:
        print('Failed to read CO2 concertration')
    time.sleep(1)
```

計測値を取得する

計測値を表示する

@2025 Kazuhiro Fukuda, Nikkei Business Publications, Inc.