31 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 －

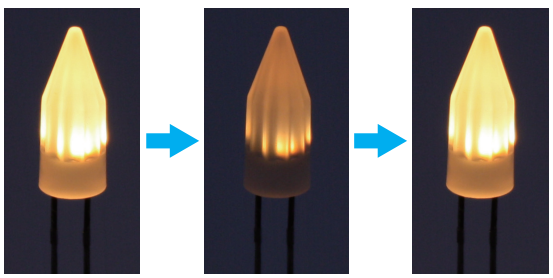
利用パーツ

●キャンドルLED×1（秋：111585） ●トランジスタ×1（秋：106477）
●キャンドルIC×1（L：5226） ●黄色LED×1（秋：111657）
●抵抗器330Ω×1（秋：125331）、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**）。ラズパイ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

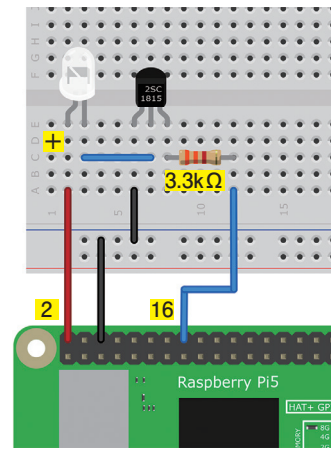
もし、任意のLEDをロウソクのように点灯したい場合には、「キャンドルIC」を利用する手もあります（**下図**）。キャンドルICに電源と点灯したいLEDを接続するだけで、キャンドルLEDと同じように光が揺らぐようになります。

図　任意の LED をロウソクのように光らせられる「キャンドル IC」の外観



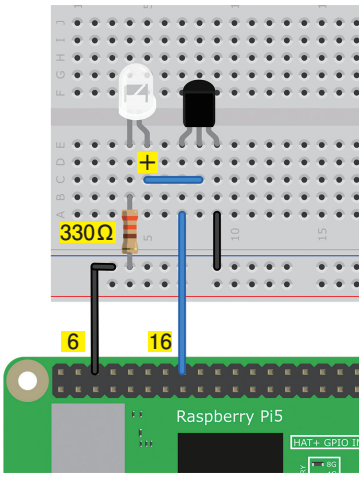
キャンドルLEDをラズパイから点灯制御するには**下図**のように接続します。ここで利用しているキャンドルLEDは5Vで動作するため、トランジスタを利用して5Vの電源を接続しています。

図　キャンドル LED の接続図



キャンドルICを使って黄色LEDをラズパイで点灯するには、**下図**のように接続します。このLEDには制限抵抗が必要になるので、忘れないよう注意しましょう。

図　キャンドル IC の接続図



以下に示すプログラム「candle_led.py」は、キャンドル LED、キャンドル IC のどちらでも動作します。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

```
import gpiozero
import time

LED_PIN = 23 ── キャンドルLEDを接続したGPIOの番号

led = gpiozero.DigitalOutputDevice( LED_PIN )

led.on()
time.sleep( 60 ) ── キャンドルLEDを点灯する時間(秒)

led.off()
```

32 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。

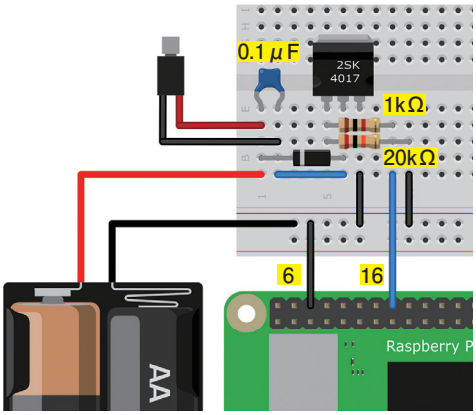
ここで紹介する振動モーター「LA3R5-480DE」は、回転軸に対して重りが偏って取り付けられています（**下図**）。この重りが回転すると、振動を引き起こします。

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**）。

図　接続図



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

```
import gpiozero
import time

M_PIN = 23

PATTERN = [ 1, 1, 2, 1, 3 ]
vib = gpiozero.DigitalOutputDevice( M_PIN )

output = 1

for data in PATTERN:
    if( output == 1 ):
        vib.on()
        output = 0
    else:
        vib.off()
        output = 1

    time.sleep( data )
```

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

33 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 ー

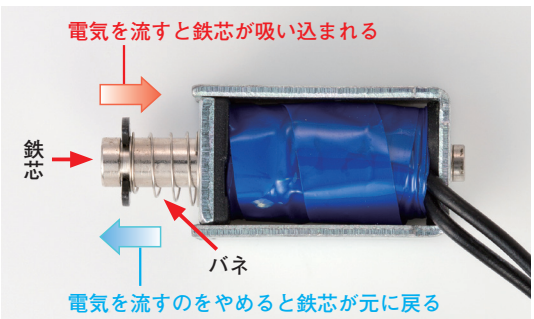
利用パーツ

●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。

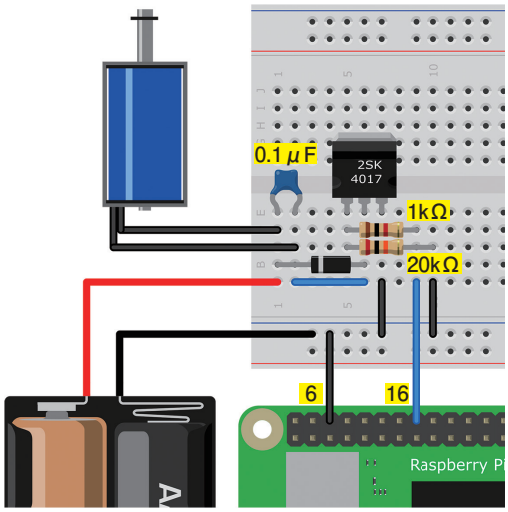
コイルの中に鉄芯が入っており、コイルに電気を流すと発生した磁力により鉄芯が引き込まれます。電気を流すのをやめると鉄芯に取り付けられているバネの力で元の位置に戻ります。

図　ソレノイドの外観



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、DCモーター同様にFETを介して制御します。ラズパイも含めた接続方法を**下図**に示します。

図　接続図



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。ラズパイのデジタル出力を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

```
import gpiozero
import time

S_PIN = 23 ── ソレノイドを接続したGPIOの番号

solenoid = gpiozero.DigitalOutputDevice( S_PIN )

solenoid.on() ── ソレノイドの鉄芯を吸い込む

time.sleep( 10 ) ── 保持する時間(秒)

solenoid.off() ── 鉄芯を元の状態に戻す
```

34 表示

バーLED

デジタル出力

プログラム barled.py

ライブラリ –

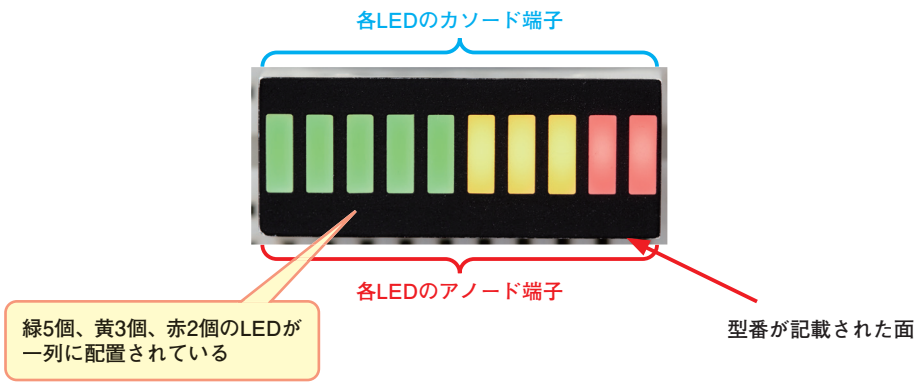
利用パーツ

●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。

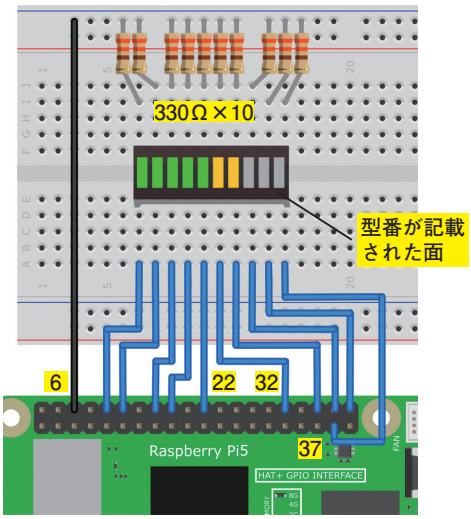
OSX10201-GYR1には、LEDごとにアノードとカソード端子が付いています。型番が記載されている面にある端子がアノードです。

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

図　接続図



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

```
import gpiozero
import time

OUTPUT = 7 ── 指定した数だけLEDが点灯する

LED_PIN = [ 15, 18, 23, 24, 25, ↵
12, 16, 20, 21, 26 ] ── 各LEDに接続したGPIOの番号

led = []

i = 0
while ( i < len( LED_PIN ) ):
    led.append( gpiozero.Digital↵
OutputDevice( LED_PIN[ i ] ) )↵
    led[ i ].off()
    i = i + 1
すべてのLEDを消灯する

count = 0
while len( LED_PIN ) > count:
    if ( count < OUTPUT ):
        led[ count ].on()
    else:
        led[ count ].off()
    count = count + 1
OUTPUTに指定した数分のLEDを点灯する
while True:
    time.sleep( 1 )
```

35 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。

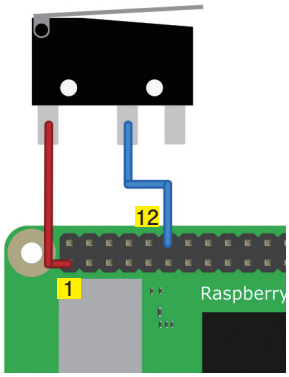
ここで紹介するマイクロスイッチは「COM」「NO」「NC」という三つの端子を備えています（**下図**）。NO（Normally Open）はスイッチを押していない状態では何も接続されず、スイッチを押すとCOMと接続された状態になります。一方、NC（Normally Close）はNOと逆で、スイッチを押していない状態ではCOMと接続され、スイッチが押されると何も接続しない状態になります。用途に応じてNOとNCのどちらを利用するかを選択します。

図　マイクロスイッチの外観



ラズパイ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



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は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

```
import gpiozero
import time

SW_PIN = 18 ── マイクロスイッチを接続したGPIOの番号
                                マイクロスイッチを接続したGPIOを入力モードにする
sw = gpiozero.DigitalInputDevice( SW_PIN, pull_up = False ) ┘
                                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 )
```

36 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）

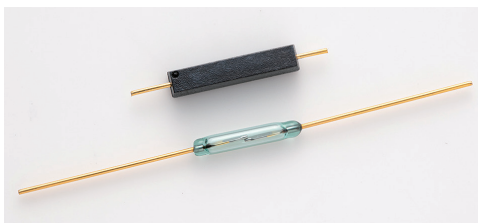
「リードスイッチ」は、磁石を近づけることでスイッチを切り替えることのできる電子パーツです（**下図**）。センサー自体に触れずに感知できるため、動くものの状態を調べたり、密封した空間内の状態を調べたりできます。ドアや窓などの開閉の状態を知りたい場合にも役立ちます。

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。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。

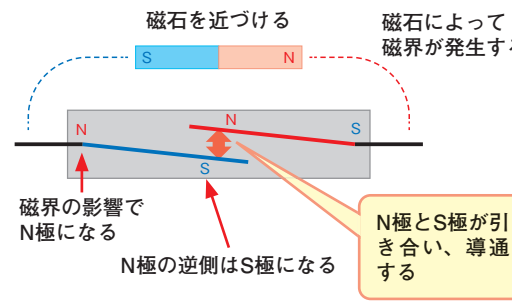
図　リードスイッチの外観と仕組み

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。

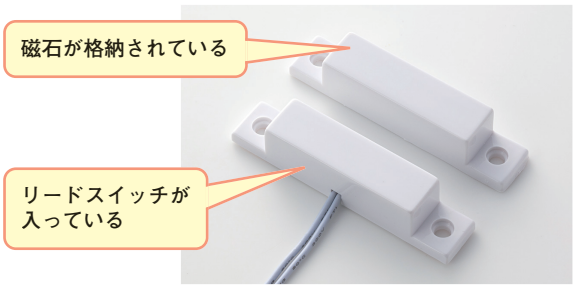


リードスイッチの仕組み



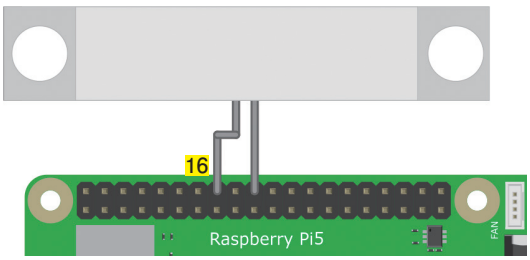
ドアなどの開閉状態を調べる目的で作られた、「ドアスイッチ」と呼ばれるリードスイッチの応用製品も市販されています（**下図**）。ドアスイッチは、二つの部品に分かれています。一つにはリードスイッチが、もう一つには磁石が入っており、二つの部品を近づけることでリードスイッチがオンの状態になる仕組みです。利用する際は、ドアの枠にリードスイッチ側を取り付けて配線し、ドア側に磁石を取り付けておきます。ドアを閉めると磁石がリードスイッチに近づき、スイッチがオンになります。

図　ドアの開閉を調べるのに使える「ドアスイッチ」の外観



リードスイッチとラズパイは**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、ラズパイのGPIOはGNDに接続された状態になり、Lowが入力されます。

図　接続図



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

```
import gpiozero
import time

SW_PIN = 23 ── ドアスイッチを接続したGPIOの番号

sw = gpiozero.DigitalInputDevice( SW_PIN, pull_up = True )
                                プルアップ抵抗を有効にする─┘
while True:
    if( sw.value == 0 ):
        print( "Close." ) ── 閉じている間は「Close.」と表示する
    else:
        print( "Open." ) ── 開いたら「Open.」と表示する

    time.sleep( 0.5 )
```

37 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

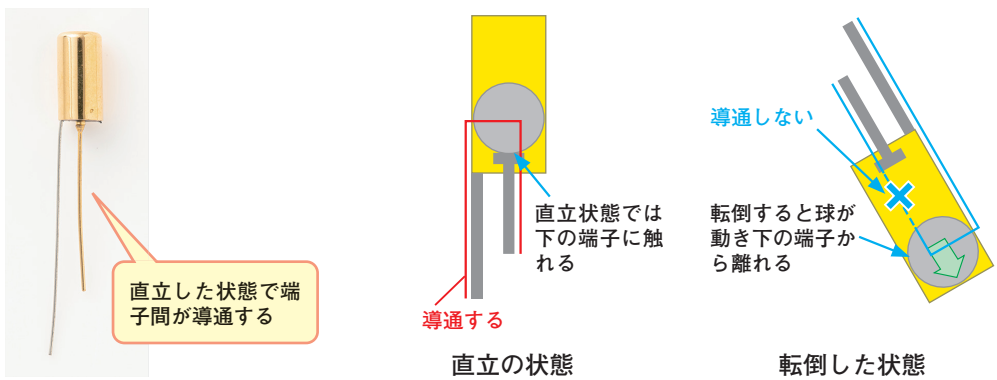
利用パーツ

●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。

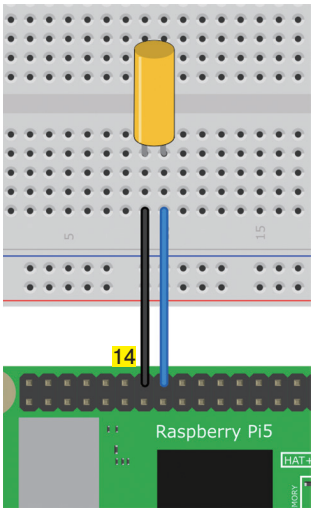
振動スイッチの内部には金属球が入っており、上下方向に自由に動きます。このため、直立状態では底面に配置された端子に金属球が触れて端子間が導通していますが、転倒すると金属球が端子から離れて導通しなくなります。

図　振動スイッチの外観と仕組み



振動スイッチとラズパイは、**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。

図　接続図



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

```
import gpiozero
import time

SW_PIN = 23 ── 振動スイッチを接続したGPIOの番号

sw = gpiozero.DigitalInputDevice( SW_PIN, pull_up = True ) ┐
                                    接続したGPIOを入力モードにし、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( 0.5 )
```

38 入力

デジタルジョイスティック

デジタル入力

プログラム d_joystick.py

ライブラリ －

利用パーツ

●デジタルジョイスティック×1（秋：115233）

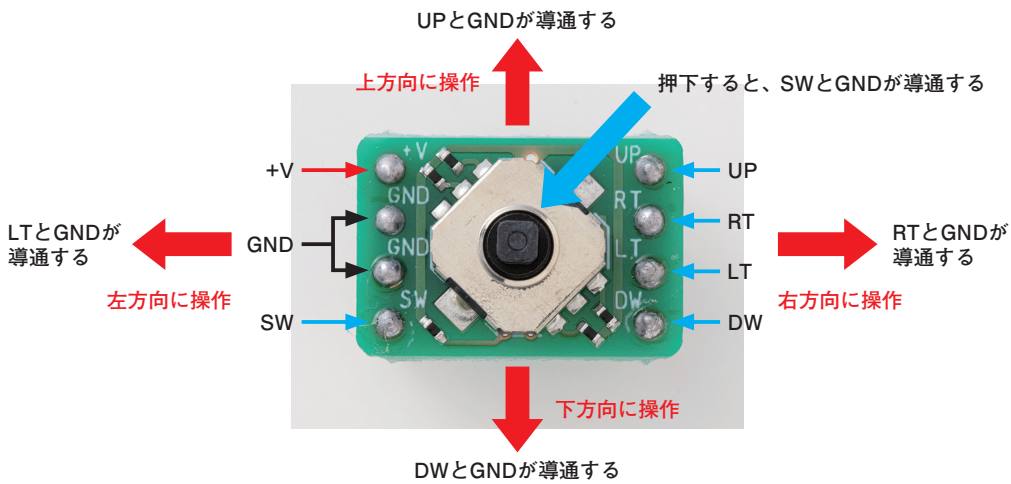
「ジョイスティック」は、搭載するスティックを上下、左右に動かすことで方向の入力ができる電子パーツです。リモコンで動かす模型やロボットアームの方向指示、メニュー画面の操作などに使えます。

ここで紹介するデジタルジョイスティックは、上下左右の４方向にスイッチが入っており、操作すると各スイッチがオンになります（**下図**）。加えて、スティックの根元に押し込むとオンにできる押しボタンスイッチも搭載しています。

各スイッチは一方がGND、もう一方がそれぞれ別々の端子に接続されています。操作すると対象の端子とGNDがつながった状態になります。例えば、上方向に操作するとUP端子とGND端子が導通します。各スイッチの端子をラズパイのGPIOに接続し、デジタル入力がLowになるかどうかを調べることで、スティックの入力方向が分かります。

なお、各スイッチはプルアップ抵抗を搭載しているため、プログラムでラズパイの内蔵プルアップ抵抗を有効にする必要はありません。

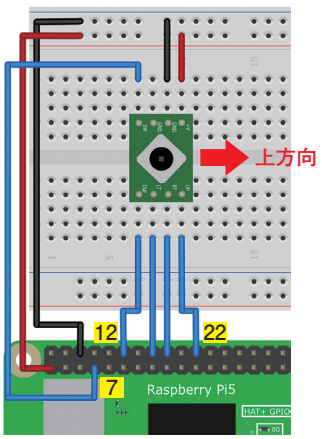
図　デジタルジョイスティックの外観



デジタルジョイスティックとラズパイは**下図**のように接続します。各スイッチは、上方向をGPIO 25、下方向をGPIO 18、左方向をGPIO 23、右方向をGPIO 24、押しボタンをGPIO 4に接続しています。

この図では、ジョイスティックを右に90度回して差し込んでいます。このため、図中に矢印で示した方向が上方向の操作となるので注意してください。

図　接続図



デジタルジョイスティックの操作を読み取るプログラム「d_joystick.py」を以下に示します。ジョイスティックに接続した各GPIOの状態をチェックし続けて、Lowになったらその方向へのスティックの入力があったと判断し、方向を表示します。

ソース　デジタルジョイスティックの操作を読み取るプログラム

d_joystick.py

```
import gpiozero
import time

up_pin = 25
dw_pin = 18
lt_pin = 23
rt_pin = 24
sw_pin = 4

up = gpiozero.DigitalInputDevice( up_pin )
dw = gpiozero.DigitalInputDevice( dw_pin )
lt = gpiozero.DigitalInputDevice( lt_pin )
rt = gpiozero.DigitalInputDevice( rt_pin )
sw = gpiozero.DigitalInputDevice( sw_pin )
```

ジョイスティックに接続したGPIOの番号

```
while True:
    if ( up.value == 0 ):
        print( "UP" )
    elif ( dw.value == 0 ):
        print( "DOWN" )
    elif ( lt.value == 0 ):
        print( "LEFT" )
    elif ( rt.value == 0 ):
        print( "RIGHT" )
    elif ( sw.value == 0 ):
        print( "PUSH" )
    else:
        print( "NONE" )
    time.sleep( 0.5 )
```

ジョイスティックを操作したらその方向を表示

何も操作していない場合に表示

39 センサー

SPI

熱電対

プログラム thermocouple.py

ライブラリ ー

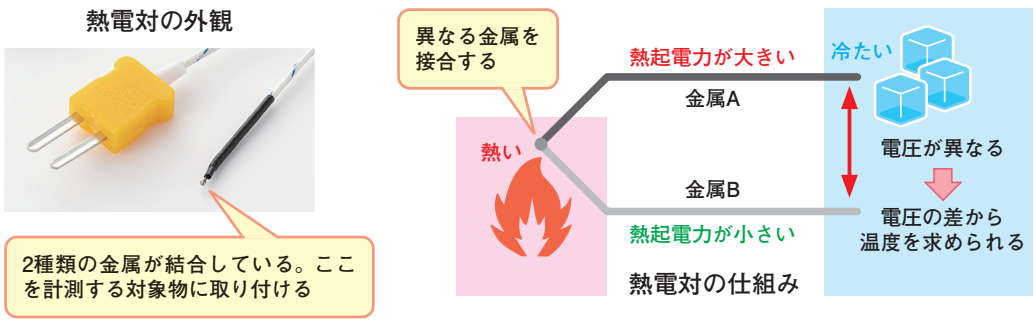
利用パーツ

●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**）。

表　主な熱電対と計測可能な温度範囲

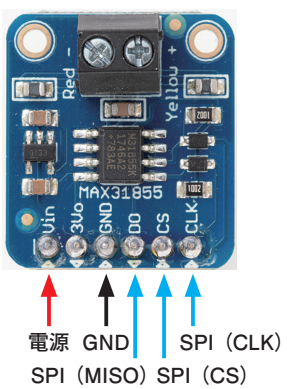
熱電対の型	利用金属（＋側）	利用金属（－側）	計測可能温度範囲*1
K型	クロメル	アルメル	-270～1372℃
N型	ナイクロシル	ナイシル	-270～1300℃
E型	クロメル	コンスタンタン	-270～1000℃
J型	鉄	コンスタンタン	-210～1200℃
T型	銅	コンスタンタン	-270～400℃
B型	白金・ロジウム（30%）	白金・ロジウム（6%）	0～1820℃
R型	白金・ロジウム（13%）	白金	-50～1768℃
S型	白金・ロジウム（10%）	白金	-50～1768℃

＊1　JIS規格で定義されている温度範囲

ここで利用する熱電対（秋月電子通商で販売）は、K型熱電対を用いた製品で、-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40.7μV上昇または下降します。この電圧を調べることで温度を計算できます。

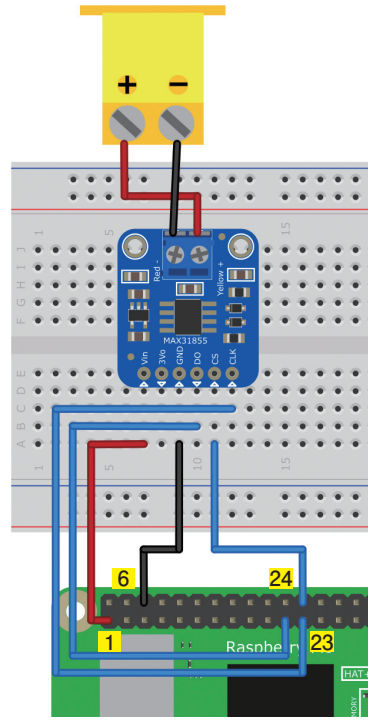
ただし、熱電対が出力する電圧は非常に小さいため、そのまま入力したのではラズパイが搭載するADコンバーターでは読み取れません。そこで、熱電対専用のADコンバーターを別途用意して読み取ります。ここでは、米Adafruit Industries社の「MAX31855K」を搭載した熱電対用ADコンバーターを利用する方法を紹介します（**下図**）。熱電対から入力された電圧を増幅した後、14ビットのADコンバーターでデジタル値に変換します。変換した計測値はSPI通信を使ってラズパイで読み取ります。

図　熱電対用ADコンバーター「MAX31855K」搭載モジュール



熱電対は**下図**のようにラズパイと接続します。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 9（21番端子）、CLKをGPIO 11（23番端子）、CSをGPIO 8（24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



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

```
import spidev
import time

SPI_CH = 0 ── SPIのチャンネル
CE = 0
SPI_SPEED = 10000000 ── SPIの通信速度

spi = spidev.SpiDev()
spi.open( SPI_CH , CE )

spi.max_speed_hz = SPI_SPEED

def get_thermo(): ── ADコンバーターから値を取得して温度を求める関数
    data = spi.xfer2( [ 0x00, 0x00, 0x00, 0x00 ] )

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

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

    thermo_temp = thermo_data * 0.25 ── 計算で温度を求める

    return thermo_temp

while True:
    temp = get_thermo() ── 前述の関数を呼び出して温度を取得する

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

    time.sleep( 1 )
```

40 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

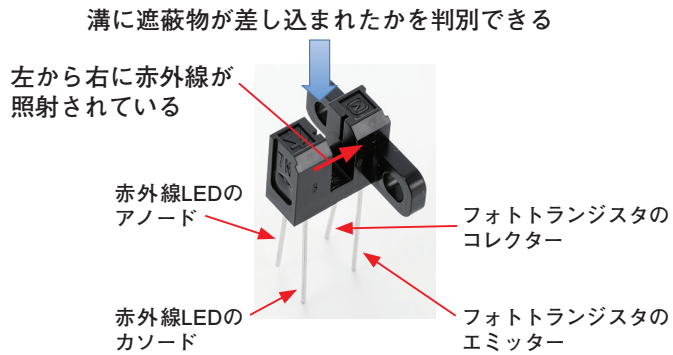
ライブラリ ー

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。

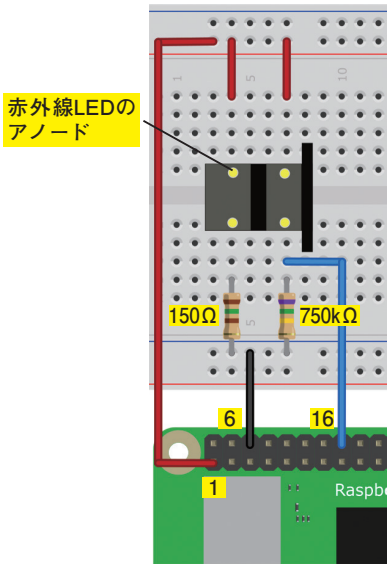
図　フォトインターラプターの外観



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ラズパイは**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。

図　接続図



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

```
import gpiozero
import time

PINT_PIN = 23 ── エミッターに接続したGPIOの番号

pint = gpiozero.DigitalInputDevice( PINT_PIN, pull_up = None, active_state
 = True )

while True:
    value = pint.value ── センサーの状態を読み取る

    if( value == 1 ):
        print ("None.")
    else:
        print ("Block.")  ── 遮蔽物の有無を判別して表示する

    time.sleep( 1 )
```

41 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 s11059.py

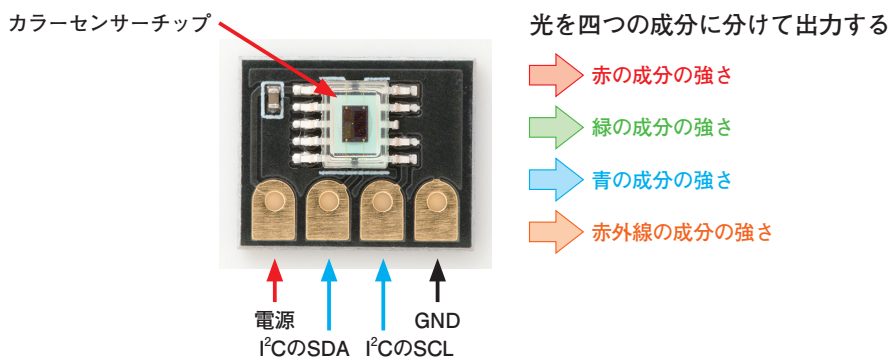
利用パーツ

●カラーセンサー（秋:108316）

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。

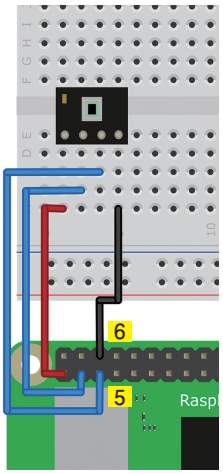
ここで紹介するカラーセンサー「S11059-02DT」を搭載したモジュールは、中央に搭載したカラーセンサーチップで受けた光を3原色の成分ごとに計測できるほか、赤外線の計測も可能です。

図　カラーセンサーの外観



S11059-02DT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「0x2a」です。ラズパイとは**下図**のように接続します。

図　接続図



S11059-02DT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「s11059.py」を利用します。プログラムと同じフォルダー内に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、各色成分の計測値を取得できます。

ソース　照射した光の成分を計測するプログラム

color_sensor.py

```
import smbus2
import time
import s11059

S11059_ADDR = 0x2A ── センサーのI²Cアドレス
I2C_CH = 1

i2c = smbus2.SMBus( I2C_CH )

sensor = s11059.s11059( i2c, S11059_ADDR )

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

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

    time.sleep( 0.5 )
```

42 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 －

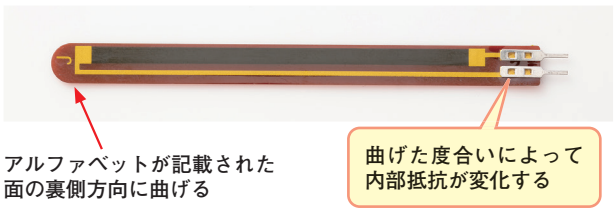
利用パーツ

●曲げセンサー ×1（秋：116410） ●ADコンバーター ×1（秋：100238）
●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。

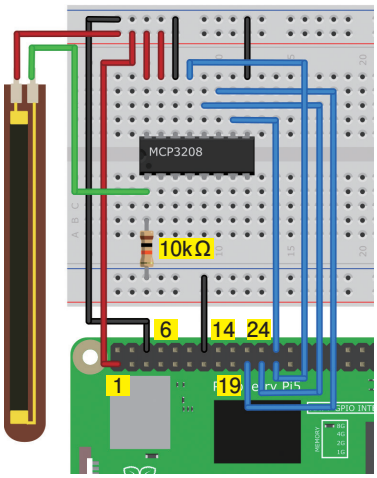
ここで紹介する曲げセンサーは、片面にアルファベットが記載されています（**下図**）。この面とは逆方向に曲げることで内部抵抗値が変化します。伸ばした状態では内部抵抗は小さく、曲げていくことで内部抵抗が徐々に増えていきます。

図　曲げセンサーの外観



曲げセンサーとラズパイは**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ADコンバーターを介して読み取ります（分圧回路については書籍の8-5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。

図　接続図



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。プログラムを実行し、曲げセンサーをまっすぐにした状態と180度曲げた状態における「Value」の値をメモしておきます。そして、BEND_MINにまっすぐの状態の値を、BEND_MAXには180度曲げた状態の値を設定してください。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）。

ソース　曲げた度合いを表示するプログラム

bend.py

```
import gpiozero
import time

BEND_MIN = 1000 ── 伸ばした状態の値
BEND_MAX = 50 ── 180度曲げた状態の値

MCP3208_CE = 0
BEND_CH = 0 ── 曲げセンサーを接続したADコンバーターのチャンネル番号

adc = gpiozero.MCP3208( device = MCP3208_CE, channel = BEND_CH )

while True:                    計測値を取得する
    value = adc.raw_value ──┘

    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( 0.5 )
```

43 センサー

UART

ドップラーセンサー

プログラム doppler.py

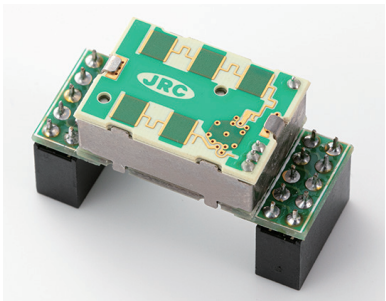
ライブラリ ー

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

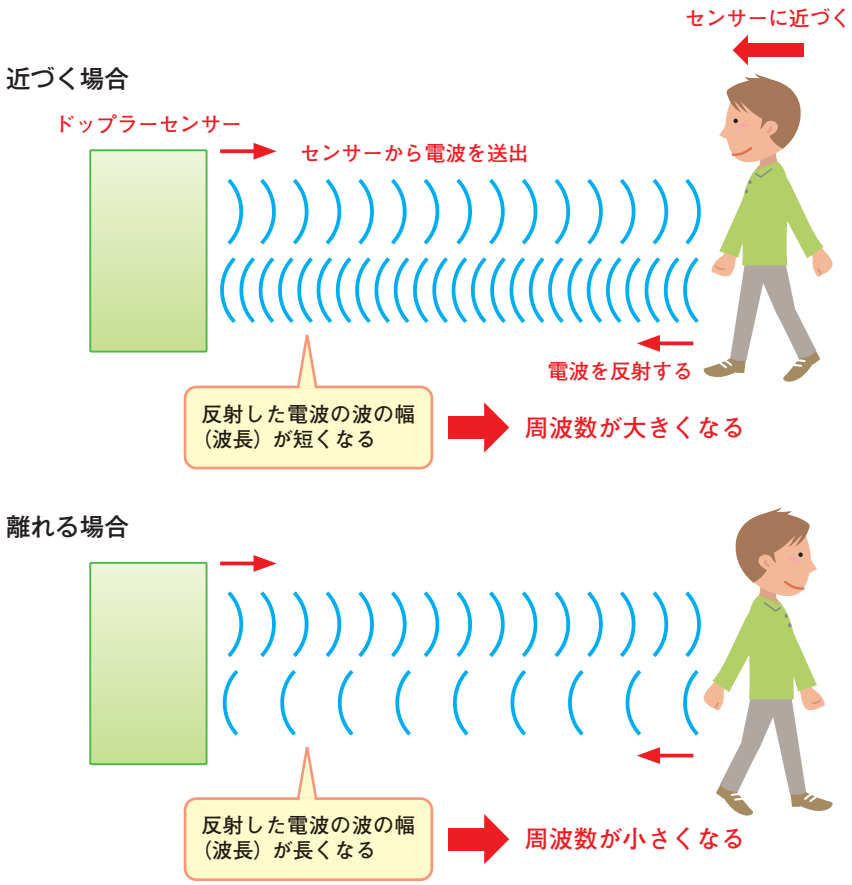


ドップラーセンサーは、電波を送出しています。1秒間に発生する電波の波の数を周波数と呼び、Hz（ヘルツ）という単位で表します。ここで紹介するドップラーセンサーは周波数24GHzの電波を発します。

ドップラーセンサーから発した電波は、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**）。

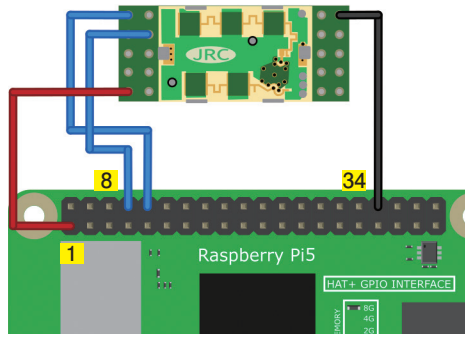
一方、センサーから離れる方向に動くと、1秒当たりの波の数が少なくなり、周波数が低くなります。この周波数の変化を見ることで、近づいているか離れているかを判断できます。これがいわゆる「ドップラー効果」で、救急車が近づいてくる場合にサイレンが高い音（周波数が高い）で聞こえ、遠ざかると低い音（周波数が低い）で聞こえるのと同じ原理です。

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ラズパイは**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-メ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。

図　接続図



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「@C」の場合は物体がセンサーに近づいており、「@L」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

```
import serial

serial_tty = "/dev/ttyAMA0"
serial_speed = 9600

ser = serial.Serial( serial_tty, serial_speed )

while(True):
    sensor_line = ''
    while ( True ):
        buf = ser.read()
        if ( buf == b'\r' ):
            break
        elif ( buf != b'\n' ):
            sensor_line = sensor_line + buf.decode()
    sensor_line = sensor_line.replace('\n','')
    sensor_line = sensor_line.replace('\r','')

    if ( sensor_line == '@C' ):
        print( "Approach." )
    elif ( sensor_line == '@L' ):
        print( "Move away." )
```

値が@C(近づいている)の場合に表示する

値が@L(遠ざかっている)の場合に表示する

44 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） ●ADコンバーター ×1（秋:100238）
●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

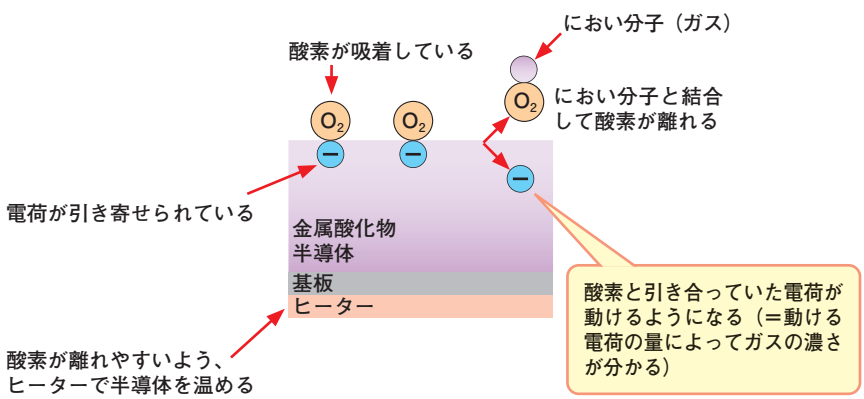
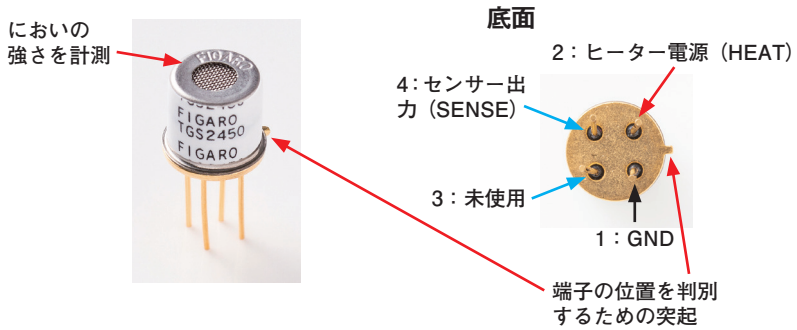
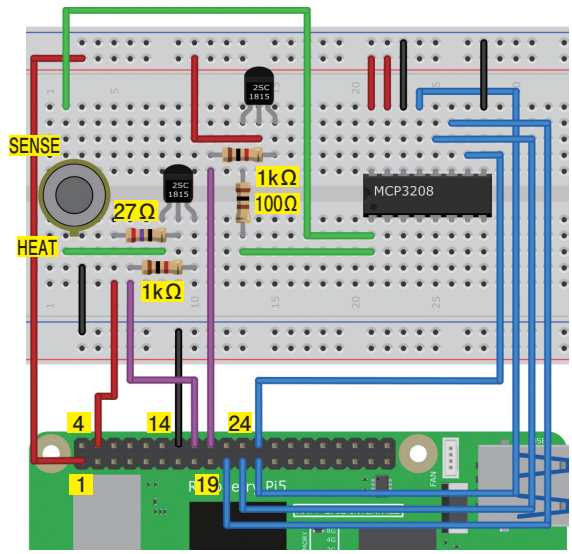

においセンサー
酸素が吸着している
においセンサー
O2
O2
O2
においセンサー
においセンサー
においセンサー

図　TGS2450 の外観と各端子の役割



TGS2450 とラズパイは**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



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。

ソース　においを計測するプログラム

smell.py

```
import gpiozero
import time

TH = 2.0 ── においが強いと判断するしきい値

HEATER_PIN = 23
SENSOR_PIN = 24 ── センサー制御用のGPIO番号

MCP3208_CE = 0
SENSOR_CH = 0 ── においセンサーを接続したADコンバーターのチャンネル番号
VREF = 3.3 ── 電源電圧

adc = gpiozero.MCP3208( device = MCP3208_CE, channel = SENSOR_CH )

heater = gpiozero.DigitalOutputDevice( HEATER_PIN )
sensor = gpiozero.DigitalOutputDevice( SENSOR_PIN )

heater.off()
sensor.off()

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

    value = adc.raw_value
    volt = value / 4096.0 * VREF ── センサーの計測値を取得する
    if( volt > TH ):
        print ( "OK. Volt: {:.2f} V".format( volt ) )
    else:
        print ( "BAD. Volt: {:.2f} V".format( volt ) ) ── においが強いかを判別する

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

    time.sleep( 1 )
```

45 センサー

UART

CO2センサー

プログラム co2.py

ライブラリ ー

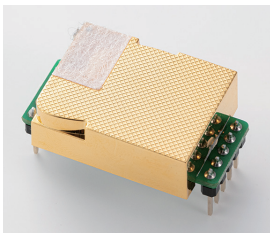
利用パーツ

●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。

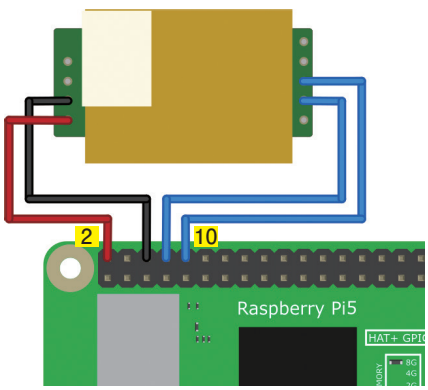
ここで紹介するCO2センサー「MH-Z19C」は、400～5000ppmのCO2濃度を計測可能です。ラズパイで計測値を読み取るにはUARTを使います。

図　CO2 センサーの外観



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**のようにメス-メス型ジャンパー線を使ってラズパイと接続します。

図　接続図



CO2濃度の計測プログラム「co2.py」を以下に示します。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　二酸化炭素の濃度を計測するプログラム

co2.py

```
import serial
import time

AUTO_CALIB = True

serial_tty = "/dev/ttyAMA0"
serial_speed = 9600

ser = serial.Serial( serial_tty, serial_speed )

if ( AUTO_CALIB == True ):
    data = [ 0xff, 0x01, 0x79, 0xa0, 0x00, 0x00, 0x00, 0x00, 0xe6 ]
    senddata = bytes( data )
    ser.write( senddata )
else:
    ser.write( ''.encode('ascii') )
    data = [ 0xff, 0x01, 0x79, 0x00, 0x00, 0x00, 0x00, 0x00, 0x86 ]
    senddata = bytes( data )
    ser.write( senddata )

time.sleep(1)

while( True ):
    data = [ 0xff, 0x01, 0x86, 0x00, 0x00, 0x00, 0x00, 0x00, 0x79 ]
    senddata = bytes( data )
    ser.write( senddata )
    s = ser.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