マイコン実習⑤　Raspberry Pi Pico W　×　Python　デバイス制御 II

１．はじめに

スマートフォンや自動車、家電製品に搭載されている組み込み技術は日本の基盤産業に欠かせない技術である。IoT はソフトウェア技術、ハードウェア技術、ネットワーク技術が三位一体で進化してきた分野であり、AI の導入により「知能をもったデバイス」が生み出されている。センサ、モータ、ディスプレイの制御を通して、機械の知性とは何かを考えてもらいたい。

２．目的

Raspberry Pi Pico W と Circuit Python を利用して赤外線センサの反射を読み取り、アクチュエータや OLED を制御する。

３．基礎知識

①　SSD1331

半導体企業である SolomonSystech により開発されたディスプレイ制御 IC を搭載する OLED モジュール。フルカラー表示が可能であり、「SSD」型番は、同社の他のディスプレイ IC（例：SSD1306 など）にも共通して使われている。96×64 ドットのフルカラー表示が可能で、直線や四角形の描画、スクロールなどのグラフィックスアクセラレーション機能を持つ点が特徴である。SPI インタフェースでマイコンと接続され、赤・緑・青の RGB 各色でピクセルを制御することで、高品質なグラフィック表示を実現します。

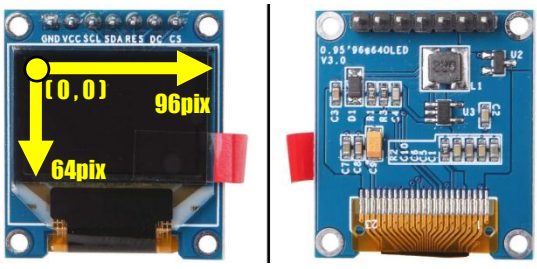


図１　SSD1331

②　RGB カラーコード

Red（赤）、Green（緑）、Blue（青）の光の三原色を混ぜ合わせることで色を表現するモデル。「加法混合」の原理に基づいており、光を重ねるほど明るくなるという性質を持つ。RGB の各成分は 0～255 の 10 進数で表され、これを 16 進数に変換した値（00～FF）が使われる。

・形式：#RRGGBB

RR	赤の強度（00～FF）
GG	緑の強度（00～FF）
BB	青の強度（00～FF）

・表示例

#FF0000	→	赤（255,　0,　0）
#00FF00	→	緑（　0,255,　0）
#0000FF	→	青（　0,　0,255）
#FFFFFF	→	白（255,255,255）
#000000	→	黒（　0,　0,　0）

③　表示枠と表示窓

OLED ディスプレイはピクセル単位で描画できるため、役割をレイヤに分けて設計することができる。

・表示枠（Frame Layer）
画面の境界や装飾、UI の区切りを示す静的な役割、背景として機能

・表示窓（Content Layer）
動的な情報やコンテンツを表示
頻繁に更新される部分

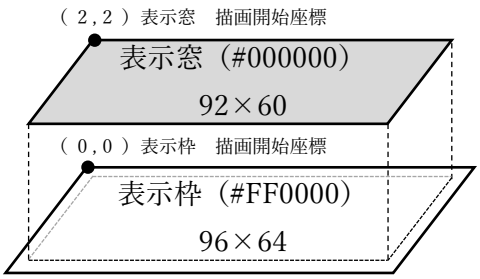


図２　表示枠と表示窓の関係

④　Pulse Width Modulation：(PWM)

パルス幅変調。ON／OFF を高速で繰り返す信号の ON 時間（パルス幅）を変化させることで、平均出力を制御する方式（図４）。モータの回転数が変化する。

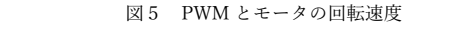


図５　PWM とモータの回転速度

④　PID 制御

モータ制御などで用いられるフィードバック制御のひとつで、入力値を目標値と比較して、出力値を制御する。PID はそれぞれ P 制御(Proportional)、I 制御(Integral)、D 制御(Differential)を組み合わせた制御の総称。

・P 制御

目標値との差に比例して出力値を制御する制御方法。モータの位置情報を入力値とし、目標値との差分から出力値を算出する。P 制御の比例ゲイン（Kp）が大きいほど出力値が目標値に近づく時間が早くなる。アクセルの踏み込みに近い。

Reference

目標値、達成したい理想の状態。

Feedback

現在の状態（フィードバック値）

センサなどで取得した実際の値。

Reference – Feedback

誤差（Error)、目標と現在のズレ。

・I 制御

I は Integral（積分）を表す。偏差を時間で積分することで、P 制御の偏差を埋めることができ、オフセット（偏差＝目標光量との差）を時間で積分し、その積算量に応じて連続的に操作量を変化させる。

I の値が小さくなればなるほど、短時間で偏差を埋めようとするため、敏感に作動し、ハンチング（制御量が目標値をオーバーしたりアンダーシュートしたりする現象）が起きやすくなる。（図６）

・D 制御

D は Derivative（微分）を表す。I 動作はオフセットを無くすように作動するが、外乱など急激な光量変化に対応することができない。D 動作は急な光量増加や減少の変化に反応して、出力を修正する役割を持つ。PI 動作無しで単独で D 動作を行なうことはできないことから、あくまで補助的な動作である。

４．実習内容

□演習１

教材配布（S:ドライブ）から「thonny-4.1.6-windows-portable」個人ドライブ（U:ドライブ）から「code.py」をダウンロードしよう。

□演習２

Raspberry Pi Pico W と PC を接続し CIRCUITPY（E:ドライブ）に配布した「code.py」を格納しよう。

□課題１

OLED によりセンサ感度を確認しよう。

Display	voltageL	voltageR
[B][B]		
[B][W]		
[W][B]		
[W][W]		

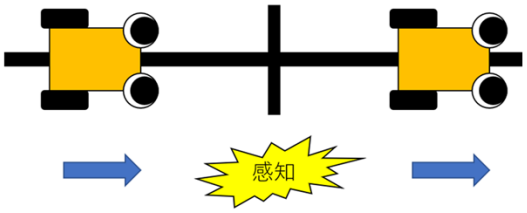
□課題２

　表示枠を緑色に、大きい文字を青色に、小さい文字を白色に変更しよう

□課題３

　L 黒 R 黒のとき減速しよう

　Duty 比 80%　50Hz＝周期（　　）ms



L黒R黒

□課題４

　サンプルプログラムを参照して PWM 制御用のプログラムに変更してみよう。

（L 黒 R 黒のとき Duty 比 30%で減速）

・手順１　import 文の追加

```
import pwmio
```

・手順２　モータ定義文の変更

変更前（削除）

```
motorL1 = digitalio.DigitalInOut(board.GP10)
motorL2 = digitalio.DigitalInOut(board.GP11)
motorR1 = digitalio.DigitalInOut(board.GP12)
motorR2 = digitalio.DigitalInOut(board.GP13)
motorL1.direction = digitalio.Direction.OUTPUT
motorL2.direction = digitalio.Direction.OUTPUT
motorR1.direction = digitalio.Direction.OUTPUT
motorR2.direction = digitalio.Direction.OUTPUT
```

変更後（追加）

```
L1 = pwmio.PWMOut(board.GP10,frequency=50)
L2 = pwmio.PWMOut(board.GP11,frequency=50)
R1 = pwmio.PWMOut(board.GP12,frequency=50)
R2 = pwmio.PWMOut(board.GP13,frequency=50)
motorL = motor.DCMotor(L1,L2)
motorR = motor.DCMotor(R1,R2)
```

・手順３　モータ制御文の変更

（L 白 R 白）の場合

変更前（削除）

```
if voltageL >= 0.8 and voltageR >= 0.8 :
        text="[W][W]"
        motorR1.value = 1
        motorR2.value = 0
        motorL1.value = 1
        motorL2.value = 0
```

変更後（追加：Duty 比 100%）

```
if voltageL >= 0.8 and voltageR >= 0.8 :
        text="[W][W]"
        motorL.throttle = 1.0
        motorR.throttle = 1.0
```

□課題５

　PID 制御プログラムを読み込み、動作を確認したい。CIRCUITPY(E:ドライブ)より program⑤PID.txt をダウンロードしてパラメータを設定しよう。コースを走らせ、その挙動を観察して記録し、なぜそのような動きとなったのか考察しよう。

課題 1	課題 2	課題 3	課題 4	課題 5
確認印	確認印	確認印	確認印	確認印

課題 5　検討・考察

① 条件 1

② 条件 2

③ 条件 3

④ 条件 4

⑤ 条件 5

⑥ 条件 6

PID 制御は私たちの生活の中で様々に利用されている。利用例を示し、PID それぞれの具体的な役割を説明しよう。

利用例

P 制御

I 制御

D 制御

2 年　　組　　番　　氏名