[第4章]

フィード・バック制御により回転速度を一定に保つ

DCモータをPWMで駆動し速度制御を試みる

鶴見 惠一

DCモータをパルス幅変調(PWM)で駆動して速度制御を試みます．コントローラのデバイスはPIC16F84Aです．最初に製作するのは，ボリューム(*VR*)でスピード調整を行う装置です．*VR*の位置はPIC16F84Aのポートを1本だけ使って読み込み，その結果でPWMのパルス幅を変化させてスピードをコントロールします．次は回転センサを自作してフィード・バックを行い，モータを一定回転に保つ制御を試みます．フィード・バックにはPI制御を使いました．

ここで使用したモータは筆者のジャンク箱にあったDC24V，10W程度の小型DCモータで，ラベルが消えてしまって型番がわからないのですが，制約はありません．DC24Vもしくは以下の適当な小型DCモータを用意してください．

4-1　*VR*でDCモータのスピードをコントロール

DCモータのスピードを変化させるのは電圧を変化させるのが最も単純です．モータとシリーズにボリューム(可変抵抗器)をつなげばこれでスピードをコントロールできますが，ボリューム(可変抵抗器)でも相応の電力を消費するので，よほど大きなボリューム(可変抵抗器)でないと焦げてしまいます．ここではPWMの方法でコントロールします．

● PWMとは

Pulse Width Modulationの頭文字で，パルス幅変調の意味です．駆動対象(ここではDCモータ)が動きとしてまったく反応しない程度の早い周期でON/OFFを繰り返し，一周期の中でONの時間が占める割合でパワーを制御する方法です．

周期の選び方には少し微妙な問題もあって，あまり速すぎても電力のロスが増えます．ON/OFFのスイッチングに使うトランジスタがONからOFFに移行するほんの短い時間ですが，ONでもないOFFでもない中間の状態が発生します．周期が速すぎると全体に占める中間の状態が目立つようになり，トランジスタがその分電力を消費します．

PWMの様子をオシロスコープの波形で見てみましょう．**図4-1**は低速回転のとき，**図4-2**は高速回転にしたときです．どちらも周期は約8msで同じですが，低速回転では“H”の時間が短く，高速回転では長く

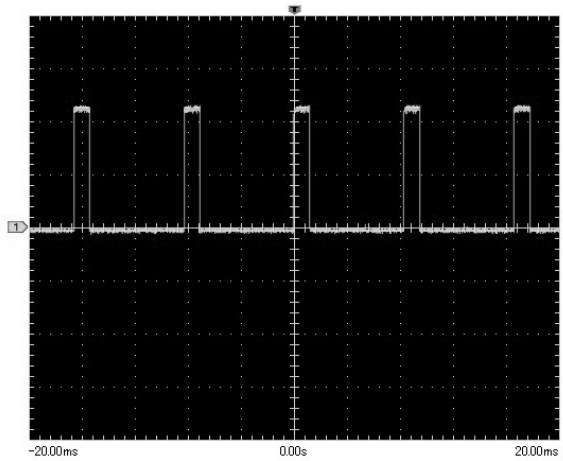

図4-1　低速回転のときのPWM出力波形
“H”になっている時間が短い（デューティが小さい）．
供給される電力は1周期中で“H”になっている時間の割合．
“H”の部分が10％ならフルに電流を流し続けた場合の10％しか電力が供給されない．

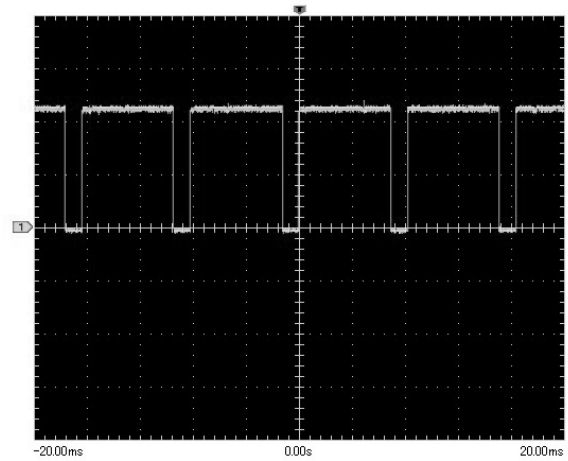

図4-2　高速回転のときのPWM出力波形
“H”になっている時間が長い（デューティが大きい）．
電力を供給する割合が大きい．連続して電流を流し続けた状態に近い．

なっています．

● コントローラの回路

このアイコンは，章末に用語解説があります

図4-3に回路図を示しました．*VR*でスピードをコントロールするだけでしたらRS-232Cインターフェース（U_2）はなくても実行できますが，フィード・バック速度制御では速度パラメータをコマンドとしてRS-232Cで与えるので必要です．

クロック発振には手持ちの8MHz水晶を使いましたが，安価なセラミック振動子で十分です．この周波数も8MHzにこだわる必要もないのですが，RS-232Cのボー・レートだけでなく，PWMのタイミングや*VR*位置の読み取り値にも係わるので変えないほうが無難です．

● モータ駆動回路

図4-4がモータのドライバ回路です．これだけですからコントローラとは別基板に作る必要はないのですが，筆者のコントローラはプログラムを入れ替えてほかの目的にも使う汎用PICコントローラなので，別にしたまでです．

トランジスタ2SD288は筆者のジャンク箱にあったもので大変古く，今では廃止品種のはずです．モータの電圧と電流に見合った適当なものを使ってください（選び方は第3章のAppendix1を参照）．ステッピング・モータのドライバの例（第3章）で使用したダーリントン・タイプのトランジスタ（2SD1590）も適当ですが，ダーリントン・タイプは電流増幅率が高いのでベースのシリーズ抵抗（R_1）を2.2～3.3kΩ程度に変えてください．

モータはインダクティブな負荷（コイル）なので，ONからOFFになった直後に高い電圧を発生します．この電圧でドライブのトランジスタを破損させないように，これをクランプするダイオード（D_1）を忘れないでください．

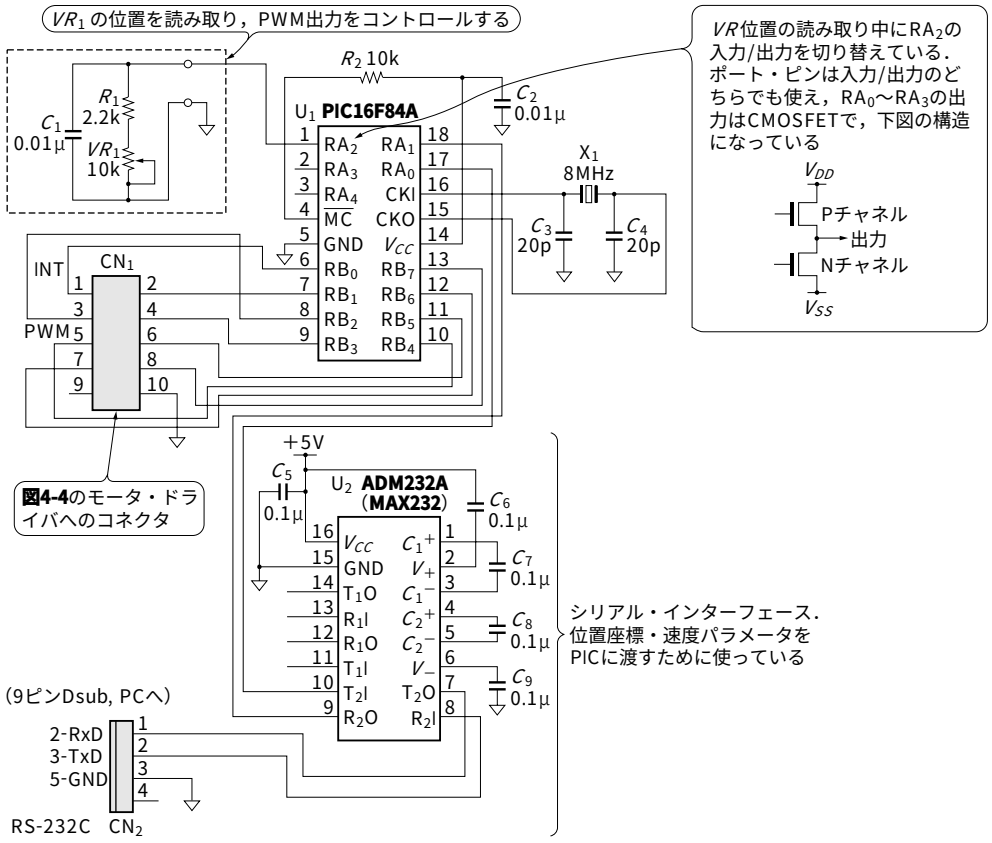

図4-3　コントローラの回路図
VR_1（10kΩ）はモータのスピード調整用ボリューム．ポート（RA_2）1本でVR位置を読み取る．
CN_1の5番（PWM）がパルス幅変調の出力信号．
X_1の水晶振動子の代わりに三端子のセラミック振動子を使う場合はC_3，C_4のコンデンサは不要．セラミック振動子の中間端子はグラウンドに接続．

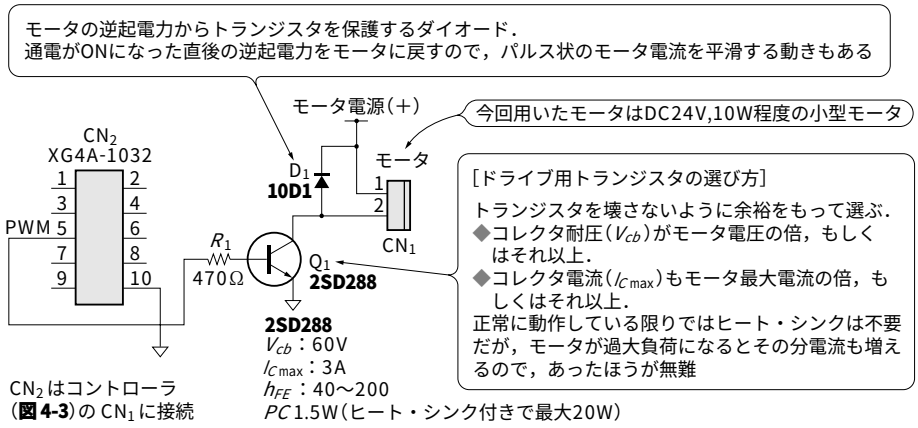

図4-4　モータ駆動回路
これだけの回路を別基板に組む必要はまったくない．図4-3のコントローラに組み込んだほうが簡単．コントローラを別目的で使うために小さな基板に組んでしまったのでこうなってしまった．

● *VR*位置読み取りのプログラム

*VR*の位置をポート1本で読み取る方法は，PIC16F84Aに内蔵されているタイマ・カウンタはPWMのクロック生成に使うので，ソフトウェアで*CR*の時定数をカウントする方法で組みました．

ポート(RA_2)を“H”出力にしておき，ポートの方向を入力に変えるとVR_1，R_1，C_1で決まる時定数で徐々にポートの入力電位が下がります．ポートが“L”を検出するまでの時間をカウントします．

リスト4-1　ポート1本で*VR*の位置を読み取るプログラム
変数tmvalueの値が*VR*の位置を反映する．

```
; VR位置読み取り
qspeed    bsf    STATUS,RP0    ; Bank1
          bsf    TRISA,2       ; RA2を入力
          bcf    STATUS,RP0    ; Bank0
          clrf   tmvalue
qspd2     btfss  PORTA,2       ; RA2が“L”になったら抜ける
          goto   qspend
          nop
          incf   tmvalue,F     ; RA2が“L”になるまでtvmalueを+1
          goto   qspd2
qspend    bsf    STATUS,RP0    ; Bank1
          bcf    TRISA,2       ; RA2を出力
          bcf    STATUS,RP0    ; Bank0
          bsf    PORTA,2       ; RA2を“H”レベル
          return
```

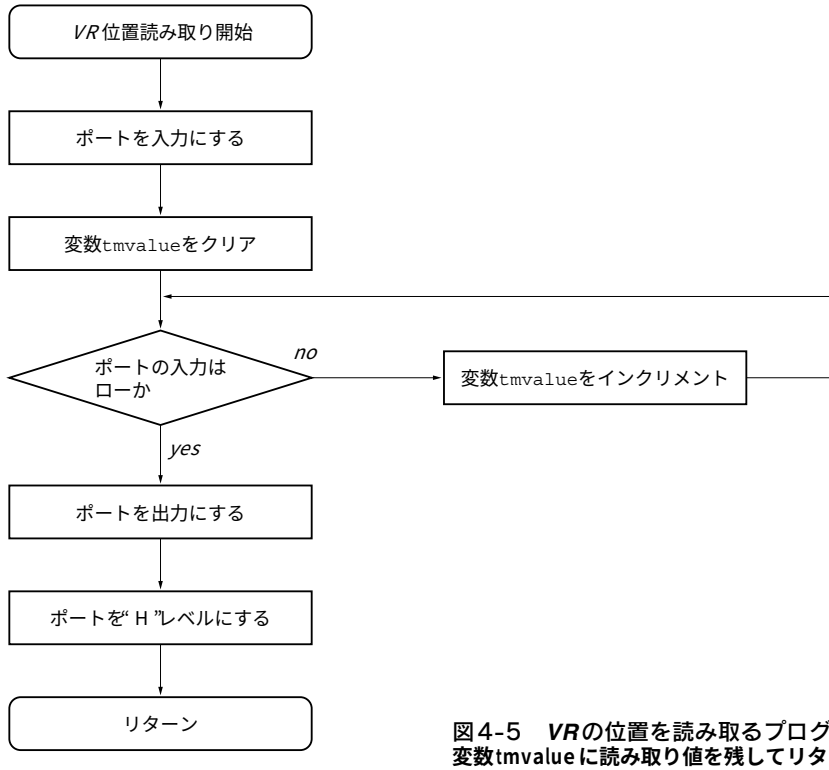

図4-5　*VR*の位置を読み取るプログラムのフロー
変数tmvalueに読み取り値を残してリターンする．

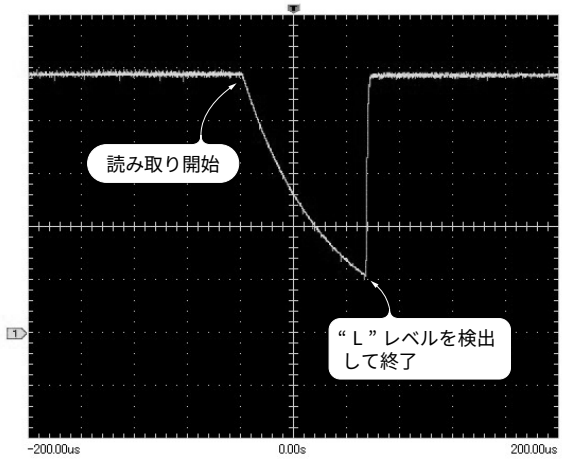

図4-6　*VR*位置読み取りを行ったときのポート(RA_2)の波形
読み取り開始から終了まで約100μsだが，*VR*の位置によって時間が変わる．読み取りプログラムの実行時間も同様に*VR*位置で変化する．読み取り精度よりも読み取り時間の短縮を優先するなら，コンデンサ(図4-3)のC_1を小さくする．読み取り値もそれに比例して小さくなる．

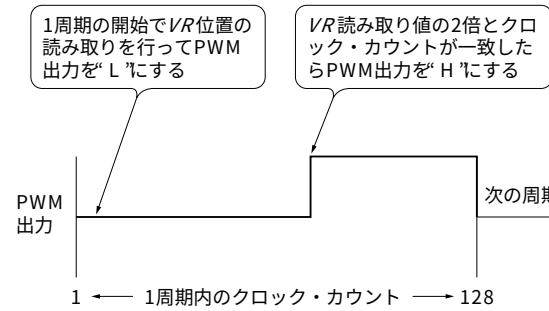

図4-7　PWMの1周期で行うプログラム処理
***VR*の位置に対応してPWM出力“H”のデューティが変化する．1周期は約8ms．**

リスト4-1が*VR*位置読み取りのソース・プログラムです．簡単なプログラムですが，フローチャートを**図4-5**に示しておきました．**図4-6**は*VR*位置読み取りを実行したときの波形を観測したオシロスコープの画面です．

ポートはRA_2を使いましたが，ほかのポートを使う場合は注意が必要です．RA_4はオープン・コレクタ(ドレイン)出力で，“H”レベルの出力ができないので使えません．ポートBでも使えますが，この場合は内部プルアップなしに設定してください．

● PWMのプログラム

16ビット幅のタイマ・カウンタを内蔵した上位のデバイスであれば，1周期の分割数を大きくして分解能の高いパルス幅変調ができるのですが，PIC16F84A内蔵の8ビット・カウンタとソフトウェアの構成では高い分解能を望むと，1周期の時間がどうしても長くなってしまいます．そこで，今回は1周期を128分割で妥協しました．高精度のサーボ・モータを実現させようとするならば不足ですが，今回の目的は十分満たすと思います．

PWM 1周期の処理を**図4-7**に示しました．*VR*の読み取り値を2倍にしていることに深い意味はありません．*VR*位置読み取りの結果をタイニ・デバッグ・モニタ(第5章)で確認すると，最小が10(16進0A)，最大が56(16進38)だったので，1周期の幅(128)に近付けようとして2倍しました．PIC16F84Aや*CR*の誤差でこの数値にはばらつきがでます．あまりかけ離れているようでしたら**図4-3**のC_1(0.01μF)を増減してください．

タイニ・デバッグ・モニタを使って*VR*位置の読み取りを行う様子は，**図4-8**のターミナル画面を参照してください．

1周期内のクロックはTMR0で作ります．TMR0のカウンタがオーバフローまでの時間は50μsに設定しましたが，割り込み処理の実行時間が加わって，クロック周期は約62μsになり，PWMの1周期が128倍の

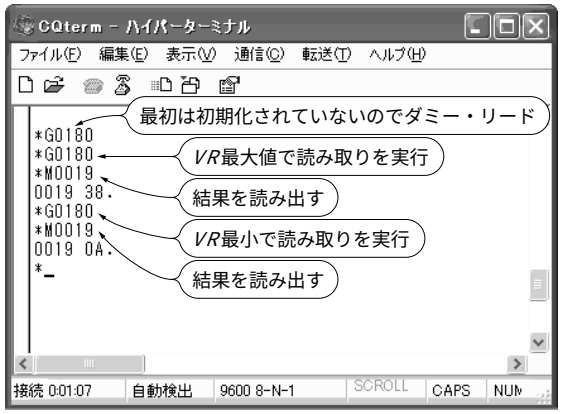

図4-8　タイニ・デバッグ・モニタを使って*VR*位置の読み取りを行ったターミナル・ソフトの画面
0180は*VR*位置読み取り(qspeed)サブルーチンのアドレス．
0019は読み取り値を格納した変数(tmvalue)のアドレス．

リスト4-2　TMR0の割り込み処理ルーチン

```
IntSvc     ;割り込み処理
           movwf  wstac          ;wレジスタの待避.
           swapf  STATUS,W       ;(zフラグが影響しない命令を使用)
           movwf  sstac          ;statusレジスタの待避.
           btfsc  INTCON,T0IF
           goto   tm0sv          ;TMR0割り込み

Intie      swapf  sstac,W        ;(zフラグが影響しない命令を使用)
           movwf  STATUS         ;statusレジスタを戻す.
           swapf  wstac,F        ;(zフラグが影響しない命令を使用)
           swapf  wstac,W        ;wレジスタをもとに戻す
           retfie                ;割り込みからリターン
;
tm0sv      ;TMR0割り込み処理
           bsf    PORTB,3        ;Debug,タイミング・テスト
           incf   TickTM0        ;1周期のクロック・カウントをインクリメント
           movlw  .128
           subwf  TickTM0,W      ;クロック・カウントを128と比較
           btfsc  STATUS,Z       ;一致でなければスキップ
           call   SetLpParam     ;1周期開始の処理
           movf   TickTM0,W
           subwf  tmvalue,W      ;カウントをVR読み出し値×2と比較
           btfsc  STATUS,Z       ;一致しなければスキップ
           bsf    PORTB,PWMout   ;PWM出力を"H"
           bcf    PORTB,3        ;Debug,タイミング・テスト
           movf   STm0Rt,W
           movwf  TMR0           ;タイマ・カウンタを初期値に戻す
           bcf    INTCON,T0IF
           goto   Intie
;
SetLpParam                              ;1周期開始の処理
           call   qspeed         ;VR位置読み取り
           bcf    STATUS,C
           rlf    tmvalue,F      ;VR読み出し値を2倍
           clrf   TickTM0        ;1周期のクロック・カウントをクリア
           bcf    PORTB,PWMout   ;PWM出力を"L"
           return
```

約8msになったわけです．

図4-7の動作を行うプログラムが**リスト4-2**です．これは割り込み処理ルーチンで，TMR0のカウンタがオーバフローするたびに実行されます．実行のたびに1周期のカウンタ（変数TickTM0）をインクリメントし，その値に応じて**図4-7**で示す処理を行います．

割り込み処理の最初の部分でポートのRB_3出力を“H”にし，途中で“L”に戻しているのは，割り込み処理実行の様子をオシロスコープで観測してタイミングを確かめるためのテスト端子としてRB_3を利用しているからです．コメントとして「Debug，タイミング・テスト」と書かれている箇所です．

● コントローラのプログラムをアセンブル，書き込み

ソース・プログラムは“DCcont1.asm”です（リストは稿末に掲載）．インクルードしているタイニ・デバッグ・モニタ“PicMonV2.asm”と“MonMacro.inc”も同じフォルダに置いてアセンブルします．“PicMonV2.asm”ではレジスタ・ファイルなどの定義ファイル“p16F84A.inc”もインクルードしますが，アセンブラのMPASMWINがインストールされた際に定義ファイルも一緒に用意され，パスも有効になっているはずなので何もする必要はないはずです．

“PicMonV2.asm”は一部変更が必要です．TMR0の割り込みを使うので，**リスト4-3**で示したように割り込みベクタを追加します．

ファイルの用意ができましたらマイクロチップ・テクノロジー社版のMPASMWINでアセンブルします．

● PWMを実行

アセンブル結果のファイル“DCcont1.HEX”をデバイスに書き込んで実行してみましょう．RS-232Cでパソコンと接続し，パソコンにはターミナル・ソフトを用意します．

電源を投入するとタイニ・デバッグ・モニタのコマンド待ちの状態になります．デバッグ・モニタの詳細ついては第5章を参照してください．

プロンプトの‘*’がターミナル・ソフトの画面に現れるはずですが，出なかったら[Enter]だけを打ってみてください．

モニタのコマンドで初期化のサブルーチン（ラベル名はInit）を実行すると割り込みが有効になり，*VR*の位置に対応したPWMが行われます．紹介したままのソース・プログラムであれば，Initのアドレスは0152になり，‘G0152’に続けて[Enter]をタイプすると実行します．

*VR*の位置に対応してモータのスピードが変化すれば成功です．

リスト4-3　PicMonV2.asm にTMR0の割り込みベクタを追加
グレーの部分1行．

```
; Program
            org     0x00
            goto    start
            org     0x04
            goto    IntSvc          ;割り込みサービス
            org     0x05
start
            bsf     STATUS,RP0      ; banksel Bank1
            movlw   b'11110'        ; RA0を出力ポートへ
                                    ; RA1を入力ポートへ
```

● モータ駆動を実行するとデバッグ・モニタが使えなくなる

Initは初期化のサブルーチンですから，デバッグ・モニタのGコマンドで実行してもすぐにリターンして次のコマンド待ちになっているはずですが，Initを実行してみるとコマンドが利かなくなってしまいます．この原因はRS-232Cのシリアル-パラレル変換(UART機能)をソフトウェアで行っていることにあります．RS-232Cの受信中あるいは送信中でも約62μsごとに実行される割り込み処理に時間が取られ，ボー・レートを保つタイミングが崩れてしまって正常な送受信ができなくなっているのです．

UARTを内蔵した上位のデバイスを使えばこの問題は生じないのですが，PWMプログラムの実行はできたので，ここでは目をつむって次の「一定速度にモータを制御」で対策を試みます．

4-2　PI制御で一定速度にモータを制御

PWMでDCモータのスピード・コントロールができたので，フィード・バックをかけて一定速度に制御することを試みます．

この制御にはモータの速度，つまり回転数の検出が必要になります．DCモータの回転数を検出する方法は「モータ電流のリプル」や「逆起電圧」を利用する方法もありますが，アナログ回路が少々ややこしくなるのでここでは敬遠しました．ディジタル的な方法ではフォト・インタラプタを使うのが確実ですが，ここは遊び心で，マグネットとピックアップ・コイルで自作しました．

フィード・バックにはPI制御を使ってみます．

● PI制御とは

PID(ピー・アイ・ディ)制御からDを省略した制御の方法です．ではそのPIDとは，Proportional(比例)，Integral(積分)，Derivative(微分)の頭文字で，この三つの要素を組み合わせて各要素の割合(ゲイン)を調整できるようにした制御です．

P(比例)は一般にいわれる単純なフィード・バックです．毎回測定するたびの結果が目標よりも大きすぎたらそれに比例して減らす，小さすぎたら増やす動作です．ここではPWM駆動ですから，減らすとはパルス幅を小さく(デューティを小さく)，増やすとはパルス幅を大きく(デューティを大きく)することです．フィード・バックのゲインが大きすぎると不安定になるのでむやみに大きくはできません，そのため比例だけの制御では目標値との差(定常偏差)が残る場合があります(**図4-9**参照)．

このような時に役に立つのがI(積分)を加えることです．**図4-10**に注目してください．目標との隔たりを記録に残した部分，**図4-10**の「面積」と表示した部分が積分の結果です．目標値との差が小さくても，この状態が長く続けば面積はどんどん大きくなります．その結果，Iゲインが比較的小さくても，限りなく目標値に近づこうとするフィード・バックが働き，偏差をゼロにすることが期待できます．

D(微分)は変化率を表します．**図4-10**のグラフでいえば，「変化の傾斜」と示した直線の角度の大きさ，つまり現在での測定値曲線の接線です．この量は偏差の変化量を意味するので，急激に変化が生じた場合は敏速にそれを押さえ込む制御が期待できます．

言葉を変えて**図4-10**の意味を表せば，Pは現状に反応，Iは経験に反応，Dは予測に反応と解釈しても概ね間違いではなかろうと思います．

今回はDを省略しました．モータの回転中はその慣性で回転数は急激には変わらないこと，それを無理やりに変えようとすれば相応のエネルギが必要なことなどがその理由です．