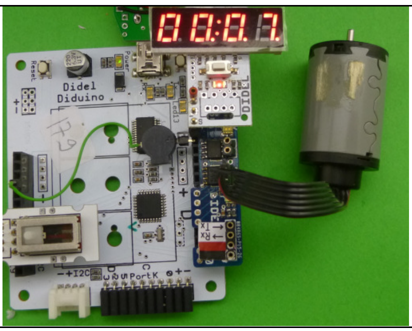
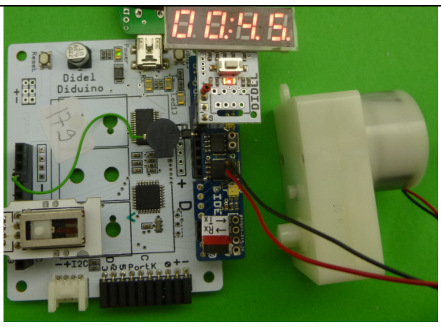
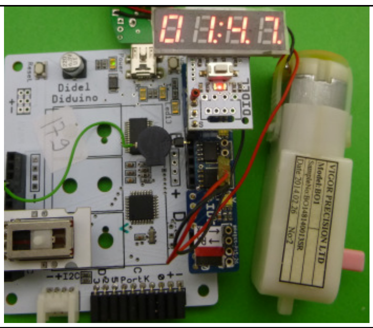
http://www.didel.com/ info@didel.com

http://www.didel.com/PFMversusPWMforRobots.pdf

# PFM vs PWM for Robots

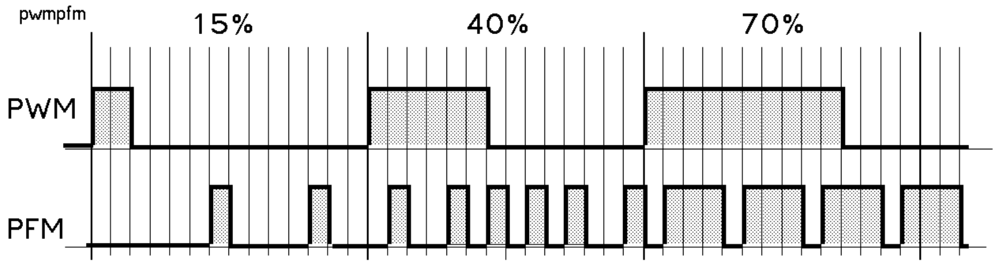
Let us compare ! What is the minimum PWM value to start a motor?

| | | |
|---|---|---|
|  |  |  |
| Maxon A-max 22mm needs 7/256 PWM (3%) (no gearbox) | Vigor BO-10 needs 45/256 PWM (18%) (1:192 gearbox) | Vigor B0-1 needs 147/256 PWM (57%) (1:120 gearbox) |

Motors are not perfect, they have friction on the brushes, bearings, gears. You can pay for the best construction, like NASA ordering Maxon motors, but for a few bucks, 20 to 50% of PWM will be required to bring enough energy to compensate friction, and the motor will start at a significant speed. If you need slow speed, PFM is the only way to go, see the Turtle moving https://youtu.be/ZIXC-oZ3zII
But notice that for LEDs (no friction, no inertia), PWM is better since PFM flickers at low PFM values.

## PWM and PFM

It is easy to understand the difference between PWM and PFM with the drawing below.
PWM is fix frequency, pulses widen to bring more power.
PFM send constant duration pulses that get closer to give more power.



As you see, below 50% of PFM, the motor gets more and more frequent pulses. Above 50%, 2ms no-pulses are missing more and more frequently and have desappeared at 100%. Motor gets an average higher frequency, this is good for DC-DC converters, but that's not the important point for a robot.

**The PFM trick with sloppy motors is to program long enough pulses so a single pulse will make the motor start.**

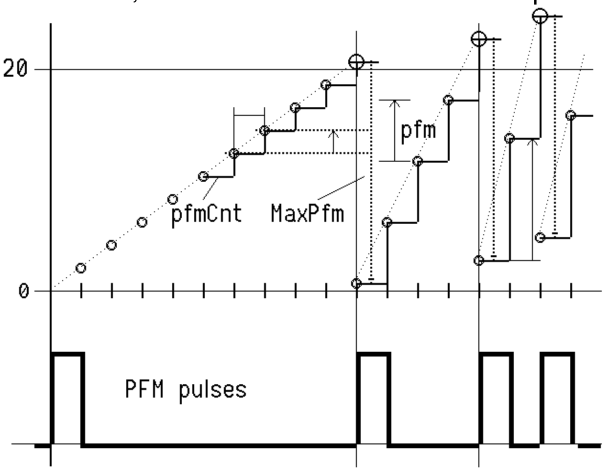
For small motors, a 2ms pulse will bring enough energy to win against frictions. It is easy to write a program to check, modify a blink program with 1-5ms On and 100 ms Off. Send one pulse every hour, the motor will spin, very slowly (if there is a gearbox, come back tomorrow to check).

PWM (Pulse Width Modulation), PFM (Pulse Frequency Modulation) and BCM (Binary Coded Modulation) are also explained in french on www.didel.com/kidules/PwmPfm.pdf . The PPM (Pulse Position Modulation) for servos has different objectives www.didel.com/kidules/Servos.pdf

## PFM routine

PFM is really simple to program. You can have as many levels you wish, but like for PWM, more than 50 is no-sense on hacker's designs. In the case of PFM, at low PFM value the period between pulses get longer. If you decide 256 steps, at 1/256 PFM, the 2ms pulses will be spaced by half a second and you will never use such a low speed.
Let us make our project with MaxPFM = 20.

The algorithm is simple. One defines `pfm` and `pfmCnt` variable. Every 2ms, `pfm` value is added to `pfmCnt`. If the result is greater than 20, 20 is subtracted and motor is powered. Otherwise, no power.



The corresponding C instructions are

```
pfmCnt+=pfm ;
if (pfmCnt>20){
  pfmCnt-=20; MotorOn;
} else {
  MotorOff;
}
```

Let us put these instructions in a loop with a delay and check different values of delay and pfm values.

```
#define MaxPFM 20
void DoPfm (pp) {
  pfmCnt+=pp ;
  if (pfmCnt>20){
    pfmCnt-=20; MotorOn;
  } else {
    MotorOff;
  }
}
// Test program
void loop () {
   pfm = 3;   // < MaxPFM
   DoPfm (pfm);
   delay (2);
}
```

That test program is blocking due to the `delay(2);` instruction. During 2ms you can do plenty of non blocking tasks and add a smaller delay to complete the 2ms cycle. For instance, if you need to blink a LED at 1 Hz, you call a DoBlink routine after the DoPfm. DoBlink counts the 2ms loops and toggle the LED every 250 counts. This will take 1 microseconds only, no need to modify the `delay(2);`.
Synchronous programming is the name of this programming style, quite applicable for simple robotic applications, and easy to program when state machines are well understood. Libraries working by interrupt can be used without interaction.

.

## PFM under interrupts

Install a timer interrupt and call every 2ms the `DoPfm()` routine and you can do your usual blocking activities in the main program, using `delay()` instructions.
The hart of the program using timer 2 is given below. You know how to do add definitions, complete the set-up and add the DoPfm routine.

```
void setup()  {
  TCCR2A = 0; //default
  TCCR2B = 0b00000010;
  TIMSK2 = 0b00000001;
  SetupMotor();
  sei();
}
ISR (TIMER2_OVF_vect) {
  TCNT2 = 50;  //  100 us
  if (cnt++ > 20)  {   // 2ms
    cnt = 0;
    DoPfm (pfm);
  }
}
```

```
void loop () {
   pfm = 3 ;  // slow
   delay (1000) ;
   pfm = 20; // fast
   delay (1000) ;
}
```

Notice that the timer does interrupt every 100 microsecond and the DoPfm routine is called only once every 20 interrupts. This is planned for adding tasks that must be served more frequently than every 2ms, not adding one timer for every task.

## Bidirectional control

As usually, negative values of speed are encoded in 2-s complement. If MaxPfm is lower than 128, pfm variable is of type `char`. The algorithm for negative values has little differences; understand with a drawing as above.

```
pfmCnt+=pfm ; // instructions when pfm is negative
if (pfmCnt<0){
   pfmCnt+=20; MotOn;
} else {
   MotOff;
}
```

We can write now the function for PFM values between –MaxPFM and +MaxPFM, with stauration, ready to be called every 2ms by interrupt or in a synchronous loop:

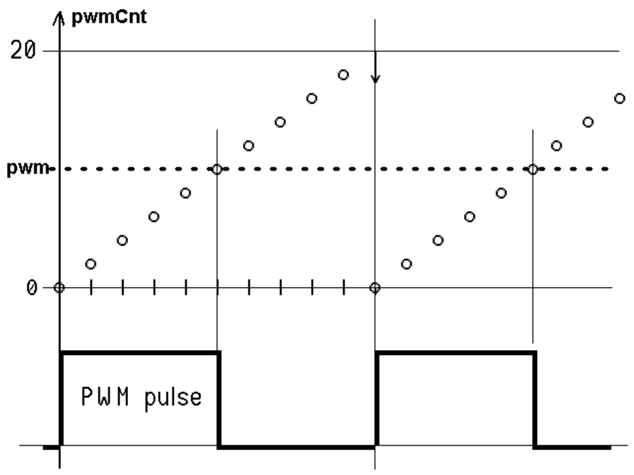
```
//PFM function to be called every 2ms (according motor)
#define MaxPfm 64 // any value lower than 128
volatile char pfm, pfmCnt;
void DoPfm () {
  if (pfm > MaxPfm) pfm=MaxPfm; // saturate
  if (pfm < -MaxPfm) pfm= -MaxPfm;
  if (pfm >=0) {
     if ((pfmCnt += pfm) > MaxPfm) {
        pfmCnt -= MaxPfm;   Forw;
     } else {
        Stop;
     }
  } else {
     if ((pfmCnt += pfm) < 0) {
        pfmCnt += MaxPfm;   Back;
     } else {
       Stop;
     }
  }
}
```

## Add motors

It was not mentionned as a clear advantage, PFM can be used on any pin, and by adding DoPfmxx routines and MotorxxOn definitions, you can add as many mono- or bidirectionnal motors as you wish. The cost will be few bytes and 1 or 2 microsecond every 2ms per motor – nothing!

## PWM

You  liked the simplicity of PFM programs? You can do the same with PWM and add as many PWM outputs as you have free pins.
The algorithm is based on next picture.



If we take again 20 steps, a counter by 20, `pwmCnt`, is compared with the `pwm` value. Motor is set On if `pwmCnt=0` and Off if `pwm<pwmCnt`.

```
#define MaxPWM 20
void DoPwm (pp) {
  pfmCnt++ ;
  if (pfmCnt>20){
    pfmCnt=0; if (pwm){MotorOn;}
  if (pfmCnt<pp){
    MotorOff;
  }
}
```

The DoPwm routine must be called with a maximum delay that depends on the flickering of LEDs. For 20 levels with a refresh rate of 20 ms, the delay is 1ms. Less is better. Doing it 100 microsecond is compatible with the interrupt routine shown earlier.

```
 ISR (TIMER2_OVF_vect) {
  TCNT2 = 50;  //  100 us
  DoPwm (pwmLed1);
  DoPwm (pwmLed2);
  if (cnt++ > 20)  {   // 2ms
    cnt = 0;
    DoPfm (pfmMot1);
    DoPfm (pfmMot2);
  }
}
```

All this will take 4-5 microsecond every 100 us (5%) and use a ridiculous amount of bytes (compared to C++ libraries).

As a last point, synchronous programming using a single timer interrupt is very flexible and efficient.. Most sensor tasks can be programmed as a state machine called every 100 microsecond. The LibX library (www.didel.com/xbot/LibX.pdf and GitHub - LibX) is based on this paradigm.

jdn 161212