**PWM (Pulse Width Modulation)**

PWM or pulse width modulation is used for various purposes in robotics like speed control of motor, running a RC servo etc.

Basic funda of PWM is to vary the on time of a square wave.

![Square Wave Diagram](image)

This is a simple square wave with symmetrical on-time and off-time i.e. it has 50% duty cycle

\[
\text{Duty Cycle} = \frac{t_{ON}}{t_{ON} + t_{OFF}}
\]

What happens if we change the on-time of the signal??
We are effectively changing the Pulse Width of the signal. This change of pulse width or changing of on-time or the change in duty cycle of a signal is called Pulse Width Modulation

![Square Wave Diagram](image)

Why to use PWM???
The most common use in robotics is to control the speed of a DC motor. How???
We know that the speed of a DC motor is proportional to the voltage applied to it. So a motor rated at 12V will run slower at 6V.
The average DC value of a square wave (50% duty cycle) is \(\frac{V_p}{2}\).
Eg: 12V for 1 min and 0V for next 1 min. Therefore average value for the 2 min time period is \( \frac{12}{2} = 6V \)

The formula for calculating the average value of voltage for a given time period is:

\[
V_{avg} = \frac{1}{T} \int_0^{t_{ON}} V_p \, dt
\]

So if we have a battery of 12V and want to slow down the robot we can apply a PWM signal to the motor driver which will reduce the average voltage applied across the motor. This will decrease the speed. At 100% duty cycle we will get 12V and it will decrease as we decrease the on time.

Another advantage is that using PWM we can make our robot take smooth turns. What we do in this case is that we don't reverse the motor direction, but slow down the speed of one of the motors.

Eg: if we need to turn right we decrease the speed of the right motor and let the left motor run as it is. As a result we don't lose the forward movement and we can complete the turn as well.

PWM can be generated using software alone, by hardware alone or by using both. Many micro controllers (ATMega series, some PICs) today have in built PWM channels which are fully software independent.

Using only software PWM can be generated as follows:

```c
while(1)
{
    SET a port pin;
    On-time delay;
    RESET port pin;
    Off-time delay;
}
```

This is the simplest way of generating a PWM waveform. The drawback obviously is that it takes up all the CPU time and you can't run any other process simultaneously.
As a combination of hardware and software u can use an internal timer overflow interrupt to generate a PWM wave. By changing the frequency of overflow of timer we can change the frequency of waveform. This is quite a simple and convenient method of generating PWM and can be used in all microcontrollers having a timer (I haven’t come across any muC not having a timer, so it is kind of a universal method to generate a PWM)

Here’s a sample code using 89V51RD2

```
#include “REG52MOD.h”

unsigned char ONTIME=50;
unsigned char DIRECTION=0x05;

void main(void)
{
    EA = 1;
    ET0 = 1;
    TMOD = (TMOD & 0xF0) | 0x01;   /* Set T/C0 Mode */
    TH0=0x00;
    TL0=0x00;
    TR0=1;
    for (;;);
}

void timer0 (void) interrupt 1
{
    static int a=0;
    a++;
    if (a == 3)
        a=1;
    switch (a)
    {
    case 1: TH0 = ONTIME;   //Change on time
            P2 = DIRECTION;
            break;
    case 2: TH0 = 255-ONTIME;
            P2 = 0x00;
    }
    TR0=1;
}

Download 89V51RD2 datasheet here.
```
And as a purely hardware type you can use micro controllers which have in built PWM channels in them. AVR is one of the most widely used microcontroller by hobbyists though some might argue that PIC rules.
The in built PWM channels use the timer register along with a compare register to change the state of a pin on a microcontroller.

Eg:
The following is a program to control the speed of a motor using PWM of Timer 0 in AVR ATmega16

```c
#include
#include"delay.h"
#include

#define CHECKBIT(x,b) x&b
#define SETBIT(x,b) x|=b;
#define CLEARBIT(x,b) x&=~b;
#define TOGGLEBIT(x,b) x^=b;

void timer0_init()
{
TCCR0=BIT(3)+BIT(2)+BIT(0)+BIT(6)+BIT(5); //Initialises Timer 0 for fast PWM
OCR0=20; //Changes the duty cycle
}

void main(void)
{
DDRB=0xFF; //Initialise PORTB as o/p
timer0_init();

while(1)
{
}
}
```