Closed Loop Speed and Position Control of DC motors

Posted on April 15, 2008, by Ibrahim KAMAL, in Motor Control, tagged

Without getting too close to the mathematical nature of this subject, this tutorial aims to explain what is the meaning of closed loop control, and how to apply it in your projects. As you shall learn in this article, closed loop control offers new possibilities to a project designer, it increases accuracy, shorten response time and dramatically decreases error.

Closed loop vs Open loop control

In general open loop control means that you send electrical signals to an actuator to perform a certain action, like connecting a motor to a battery for example. In this scheme of control, there is no any mean for your controller to make sure the task was performed correctly, and it often need human intervention to obtain accurate results. A very simple example of open loop control, is the remote controller of an RC toy car; you – the human – have to constantly check the position and the velocity of the car to adapt to the situation and move the car to the desired place.

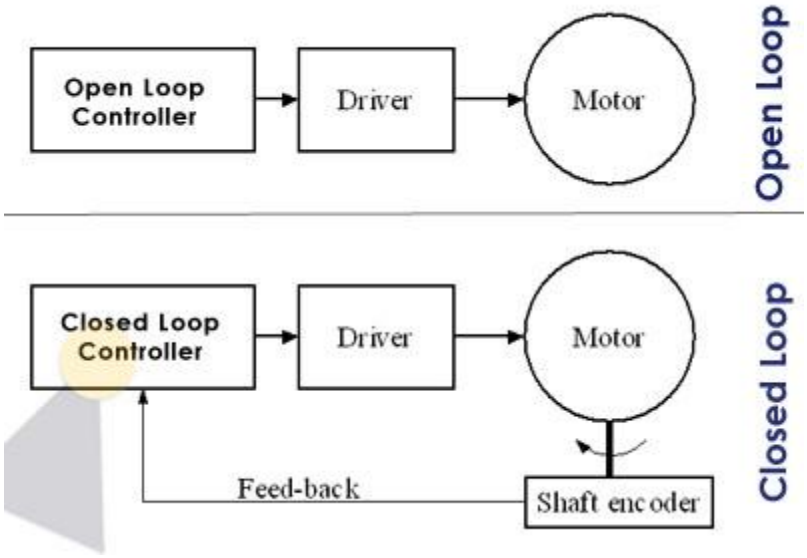

figure 1.A

But what if you could let the electronics handle a part, if not all of the tasks performed by a human in an open loop controller, while obtaining more accurate results with extremely short response time? This what is called closed loop control. In order to be able to build a closed loop controller, you need some mean of gaining information about the rotation of the shaft like the number of revolutions executed per second, or even the precise angle of the shaft. This source of information about the shaft of the motor is called “feed-back” because it sends back information from the controlled actuator to the controller.

Figure 1.A shows clearly the difference between the two control schemes. Both types have a controller that gives orders to a driver, which is a power circuit (usually an H-bridge) that drives the motor in the required direction. It is clear that the closed loop system is more complicated because it needs a 'shaft encoder' which is a devise that will translate the rotation of the shaft into electrical signals that can be communicated to the controller.

In other words, a closed loop controller will regulate the the power delivered to the motor to reach the required velocity. If the motor is to turn faster than the required velocity, the controller will deliver less power to the motor. Controlling the electrical power delivered to the motor, is usually done by Pulse Width Modulation.

Shaft encoders

figure 2.A

When working with DC motors, a shaft encoder is the most common and accurate way of providing feed-back to the controller. Shaft encoder come in many shapes and sizes, but they all rely of the same principle. ***Figure 2.A*** shows a classic encoder disk (which is one of the main parts of a shaft encoder mechanism) while ***figure 2.B*** shows how it is connected to the back-shaft of a gearhead DC motor.

The purpose of shaft encoders is shown in ***figure 2.C***, where a U-shaped photo-couple made of an Infra-Red sender and a matching receiver is positioned in a certain way so that the beam of infrared light passes through one of the small openings in the encoder disk.

figure 2.B

In reality, photo-couple come in many shapes and sizes, but most of them are more or less similar to the one shown in ***figure 2.D***. Any photo couple has four leads, two for the sender, which is usually an Infra-Red LED and, and the two others are for the receiver, which is usually a photo transistor. You can see the schematic representation of that photo couple at the lower corner of ***figure 2.D***, where is is clear that the photo-couple is composed of a LED and a photo transistor.

The encoder disk is firmly connected to the back-shaft of the motor, so that both the shaft and the encoder disk rotates at the same r.p.m. (the back-shaft is an extension of the output shaft of the motor at its back, usually present for the sole purpose of adding a shaft encoder). When this encoder disk is inserted in the configuration shown in ***figure 2.C***, the rotation of the motor causes the beam of light to be periodically intercepted by the solid parts of the encoder disk creating a sequence of pulses of light, that will be translated by the photo couple's receiver into pulses of electricity.

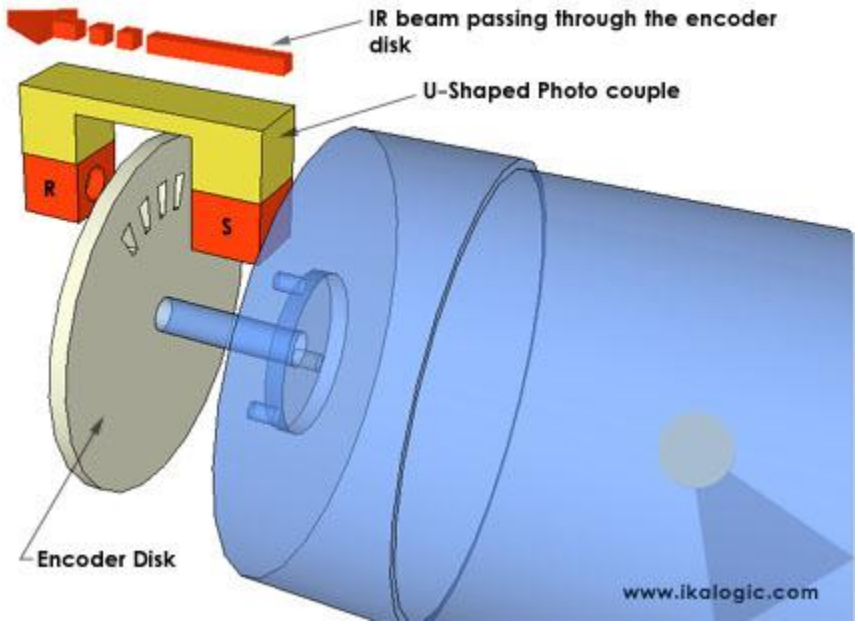

figure 2.C

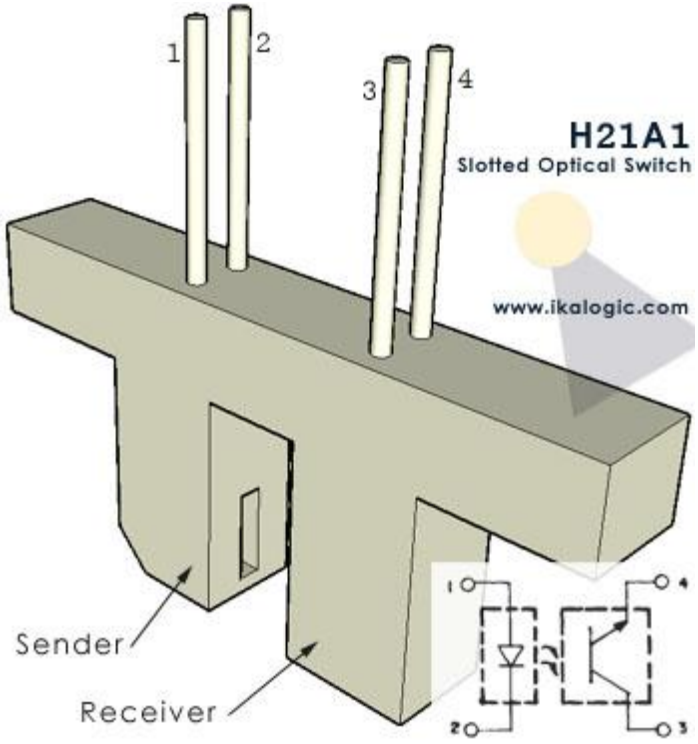

figure 2.D

Those pulses of electricity contain all the information we need to implement a closed loop control. The frequency of those pulses is directly proportional the the speed of rotation of the shaft (RPM) and the number of those pulses correspond to the angular displacement of the shaft.

The more the number of holes in an encoder disk, the higher will be the resolution (the slightest angular displacement that can be detected).

One important factor that affect the performance of shaft encoder and thus the overall performance of a closed loop control system, is the position of the encoder disk. Most of the motors are used with a gearbox designed to reduce the r.p.m. while increasing the output torque (***figure 2.B*** shows a motor+gearbox assembly). Thus, the motor itself can be turning at 4400 r.p.m. for example, driving a 40:1 gear box, dividing the rpm by 40, giving a final output speed of 110 r.p.m. You can can take a great advantage of this to reach very high degrees of accuracy, by connecting the encoder disk at the back shaft of the motor (which is turning at 4400 rpm in our example). This way, each turn of the final output shaft from the gearbox will correspond to 40 turns of the shaft encoder, and if the encoder disk has 30 holes on its circumference, a single turn on the final output shaft will correspond to 1200 pulses, reaching a theoretical precision of 0.3 degrees (i.e. each pulse correspond to 0.3 degrees of rotation of the final output shaft). (Depending on the type of motor and gearbox, it may be difficult to reach exactly that theoretical precision).

The controller

A closed loop controller can be an analog circuit, a digital circuit made of logic gates, or a microcontroller. Generally, a microcontroller is the option that will provide more design flexibility. Recent microcontrollers running at very high clock rates can completely replace similar analog controllers, and can even be cheaper.

In a closed loop system, a microcontroller will have two main tasks:

- Constantly adjust the average power delivered to the motor to reach the required velocity.
- Precisely calculate the position/angle of the motor's output shaft.
  As you can see in ***figure 3.A***, the shaft encoder will provide the microcontroller's internal counter with a sequence of pulses that correspond to the rotation of the motor. A timer is set to execute two software routine every 1/10 th of a second (which is just an arbitrary value). One of those software routines is to recalculate the actual angle of the shaft or the total number of revolutions.

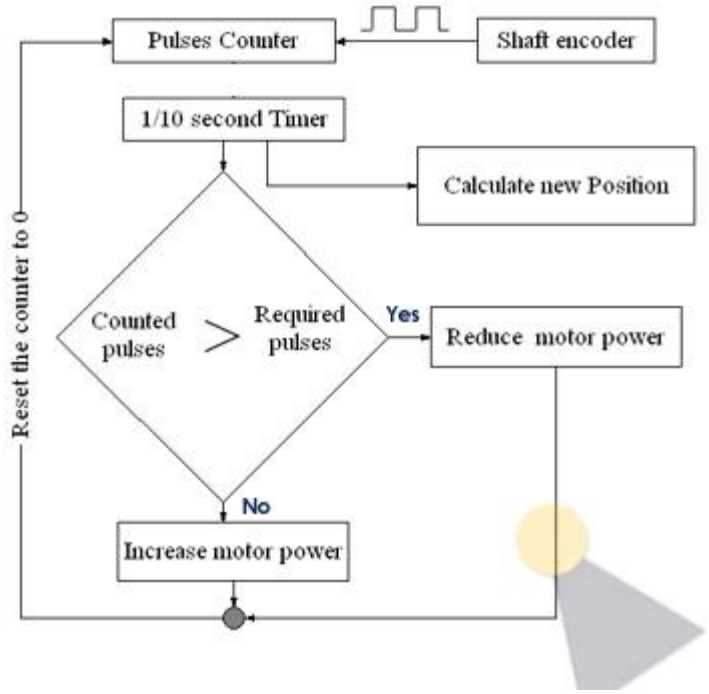

figure 3.A

Then, another software routine is executed to control the speed of the motor by comparing the number of counted pulses with a fixed number which is referred to as the “required pulses”. The “required pulses” corresponds to the desired speed, and the “counted pulses” corresponds to the actual speed of the motor.

Finally, as you can notice in the schematic, it’s all a matter of comparing those two values and constantly adjusting the power delivered to the motor. Note that choosing the right timing between each execution of this routine can dramatically improve overall system stability and performance, especially on low quality motors.

Controlling the power delivered to the motor to control its speed.

Recalling the ***figure 1.A***, a closed loop system contains a controller and a driver. the driver on its own – which is usually anH-bridge – cannot control the velocity of the motor. The most common technique to do so is to let the controller turn the driver ON and OFF at very high rates, changing the ratio between the ON and OFF time to control the speed of the motor. This is what is called PWM or Pulse Width Modulation. For more information about PWM, and to know how to implement it in a microcontroller program, you may read the “Electronics and Algorithms” part of this related article.

Example C source code for a 89S52 microcontroller

This example shows how to implement the flow chart shown in ***figure 3.A*** into a microcontroller to control the speed of two motors simultaneously, which is usually the case in differential drive robots. The code is

taken from a robot's project, where two motors were controlled by the microcontroller through H-bridge modules identical to this one, which are controlled with only two wires to determine one of the four main orders than can be given to an H-Bridge module: turn clockwise, turn anticlockwise, break or provide high impendence output (free or not connected).

In this code, the internals timers of the 89S52 Timer0 and Timer1 are used as counters, to count the pulses coming from the right and left shaft encoders, while a software timer periodically executes the functions required to update the calculated position of the shaft, and rectify the duty cycle of the PWM sent to each motor to reach the required speed.

Note that this code was simplified from an originally more complicated code that included complex differential drive line following routines, so you may encounter some errors due to undeclared variables or other minor mistakes, but the logic and the main flow of the program are correct and proved to be working perfectly.

```
#include <REGX52.h>
#include <math.h>


unsigned char req_right_pulses,req_left_pulses,speed_chk_counter;
unsigned char right_pwm,left_pwm,max_pwm,pwm_counter,right_dir,left_dir;
unsigned char pwm_adjust_delay,req_delay,max_speed;
unsigned int position_counter;


setup_timers(){
EA = 1;
TMOD = 0X55;      // counters 1 and 2 in mode 1 (16 bit counter)
ET0 = 1;     //Enable the Timer/counter 0 interrupt
TR0 = 1;     //Enable Timer/counter 0 to count
ET1 = 1;     //Enable the Timer/counter 1 interrupt
TR1 = 1;     //Enable Timer/counter 1 to count


}
timer0_overflow() interrupt 1{}
```

```
timer1_overflow() interrupt 3{}


pwm_check_and_adjust(){
    speed_chk_counter++;                        //This is what makes the
between
    if (speed_chk_counter > pwm_adjust_delay){ //each two executions of this
    speed_chk_counter = 0;                      //function


        if (req_right_pulses >  TL1){      // Compare TL1 (which contains the
            if (right_pwm < max_pwm){  // value of counter 1) and either
                right_pwm++;       // increase of decrease the pwm
            }                          // of the right motor, while making
        }else{                                  // sure the value of the pwm
stays
            if (right_pwm > 0){        // between 0 and max_pwm.
                right_pwm--;
            }
        }
        if (req_left_pulses >  TL0){       // The same that applies
            if (left_pwm < max_pwm){   // to TL1 and right_pwm,
                left_pwm++;        // applies TL0 and
            }                          // left_pwm.
        }else{
            if (left_pwm > 0){
                left_pwm--;
            }
        }
    position_counter += TL1;  // Update the position of the shaft
    TL1 = 0;                 // Reset the counters to 0.
```

```
    TL0 = 0;
    }
}

pwm_generator(){

    pwm_counter++;                                  // This is Just a counter
    if (pwm_counter > max_pwm){ pwm_counter = 0; }    // From 0 to max_pwm

    if (right_pwm > pwm_counter){           // Right Pwm, ON period
        if (right_dir == 1){        // Depending on the value of
             P2_0 = 0;         // the variable right_dir
            P2_1 = 1;        // a corresponding order will be
        }else if(right_dir == 2){           // given on the pins P2.0 and
P2.1
            P2_0 = 1;         // that are connected to the H-Bridge
            P2_1 = 0;        // that drives the motor
        }else{
            P2_0 = 1;
            P2_1 = 1;
        }
    }else{
        P2_0 = 0;
        P2_1 = 0;
    }

    if (left_pwm > pwm_counter){        // Same applies for the left motor.
        //ON period
        if (left_dir == 1){
```

```
            P2_2 = 0;
            P2_3 = 1;
        }else if(left_dir == 2){
            P2_2 = 1;
            P2_3 = 0;
        }else{
            P2_2 = 1;
            P2_3 = 1;
        }
    }else{
        P2_2 = 0;
        P2_3 = 0;
    }

}

void main(){                // This is the main part of the program,
    pwm_adjust_delay = 12;        // where main variables are initialized
    max_pwm = 25;
    setup_timers();            // timers are also initialized here
    right_dir = 1;
    left_dir = 1;
    req_right_pulses = 15;
    req_left_pulses    = 15;

    while(1){            // This is the main loop
        pwm_check_and_adjust();    // where those twu finctions are
constantly called
        pwm_generator();
```

```
    }
}
```

Source: http://www.ikalogic.com/closed-loop-speed-and-position-control-of-dc-motors/