AN ALGORITHM OF LINEAR SPEED CONTROL OF A STEPPER MOTOR IN REAL TIME

ABSTRACT: In this paper we consider the problem of realization of linear speed profile of stepper motors in real time. The general case is considered when change of speed in the acceleration and deceleration phases is different. An algorithm of the real time speed control is proposed. Comparing this algorithm with the other ones, it was shown that the algorithm is better than the others with respect to the accuracy of speed, but at the same time it is slower. The practical realization of this algorithm, using Arduino platform, is also given.

KEYWORDS: speed control, control algorithm, stepper motors, Arduino

INTRODUCTION

Stepper motor is an electromechanical device that converts electrical digital pulses into mechanical shaft rotation. Many advantages are achieved using this kinds of motors, as: (i) precise positioning and repeatability of movement, (ii) the motor has full torque at standstill (if the windings are energized), (iii) very reliable and easy maintenance since there are no contact brushes, and (iv) a wide range of rotational speeds can be realized since the speed is proportional to the frequency of the input pulses. Some disadvantages of these motors are: (i) resonance can occur if not controlled properly, and (ii) not easy to operate at extremely high speeds, [1,2].

An important issue about stepper motors is that they are usually used in an open control loop. This means that the motor control system has no feedback information about the position, which eliminates expensive sensing and feedback devices.

Many systems with stepper motors need to control the speed using values of acceleration and deceleration defined in advance. Herein we will analyze the general case, when change of speed in the phase acceleration and deceleration is linear and different (ramp speed profile). In Fig. 1 the relation between acceleration $a$ [rad/sec$^2$], deceleration $d$ [rad/sec$^2$], speed $v$ [rad/s] and position $s$ [rad] is shown.

Since the stepper motor makes steps in discrete time (after each pulse) and the move of every step is constant, the change of speed is achieved by changing the time interval between successive steps (pulses). It means that the main problem of speed control is to determine instants of the time $t$, (in [sec]) when pulses (steps) are generated. If speed $v$ is constant $v = \text{const.}$ (independent of whether it is large or small), it is very easy to determine these instants (equivalently, generate the pulses). In this case the time delays $\delta t$, between two arbitrary successive pulses, are the same and they are given as, $\delta t = \alpha / v = \text{const.}$, where $\alpha = \text{[rad]}$ is the angle of the rotation motor shaft for every step. But, if the speed is variable, $v \neq \text{const.}$, it is more difficult to determine the instants when we need to generate pulses because the time delay between two successive pulses is changed, $\delta t = \alpha / v \neq \text{const.}$.

In the case when acceleration/deceleration is constant, the speed is changed linearly, but the time...
interval \( \delta t \), between two adjacent pulses is not linear. So, we need to calculate time \( \delta t \), which ensures linear change in speed. It is shown for the acceleration phases in Fig. 2.

Depending on the method used to calculate this time, generally, there are two groups of algorithms—two methods, named as: “time per step” and “steps per time”, that are described in [3,4,5] and [6,7], respectively. The first group of algorithms is mainly realized by using a microcontroller, while the second group is suitable for realization by using FPGA (field programmable gate array) circuits. Both of the above groups of algorithms generate pulses in real time.

However, there are alternate techniques where the pulses are generated based on the previously entered table of data, which include all parameters of motion. Using these techniques we can not calculate the time when pulses are generated in real time, because any change of some parameters of the motion imply a change above table of data. Today, these techniques lose their importance, because by increasing microcontroller’s speed it becomes possible to perform all the necessary computation within short time, which is less than \( \delta t \).

In the sequel of the paper we give a new real time algorithm that belongs to the first group of the above mentioned algorithms. The practical implementation of this algorithm that is based on the Arduino platform of the Atmel microcontroller ATMega 328 is also given.

**ALGORITHM**

As we emphasized earlier, we observe motion with the ramp speed profile, i.e. \( a = \text{const.}, \quad d = \text{const.}, \quad a, d > 0, \) and let \( a \neq d \). In this case, in order to determine control algorithm it means to solve two problems: to calculate the instants of time when pulses are generated and calculate the characteristic points of the motion in which the algorithm changes its behavior (trajectory planning), see Fig. 1. We will design a controller that solves both problems.

**Calculate time when the pulse is generated**

The first pulse (step) controller generates at the start of motion, or at the start of the state of acceleration, at the time \( t_0 \), see Fig. 3. After the first pulse is generated, the controller needs to calculate time period \( \delta t_0 \) until the next pulse, wait until this period has elapsed, and then generate the next pulse, at time \( t_1 \). This will go on until the desired position is achieved, or in other words, the desired number of pulses has been generated. At the start, the speed is \( v_0 \), and it retains its value until the moment \( t_1 \), when it becomes \( v_1 \), at the moment \( t_2 \) becomes \( v_2 \), and so on. As after each pulse the motor makes one step for the angle \( \alpha \), then the following is valid

\[
\alpha = v_i \cdot \delta t_i \Rightarrow \delta t_i = \frac{\alpha}{v_i},
\]

where the \( v_i \) is speed at an arbitrary instant of time \( t_i \) and \( \delta t_i \) the time delay between two successive instants of time \( t_{i-1} \) and \( t_i \). In the case when the controller is realized by using a microcontroller, the required time delay \( \delta t_i \) is implemented using counter \( c_i \) that counts impulses of known frequency \( f \), so \( \delta t_i = \frac{c_i}{f} \) is valid. Now (1) becomes

\[
\alpha = v_i \cdot \frac{c_i}{f} \Rightarrow v_i = \frac{\alpha f}{c_i}.
\]

Based on the above considerations, the speed is changed only at the discrete time \( t_i \). But, since inertia always exists, thus we can assume that the speed \( v_i \) between two arbitrary successive instants of time \( t_{i-1} \) and \( t_i \), \( i \geq 1 \), changes linearly (see dashed light lines in Fig. 3). Thus, the speed \( v_i \) at the arbitrary instant of time \( t_i \), and in the phase of acceleration, becomes

\[
v_i = v_{i-1} + a \cdot \delta t_{i-1}, \quad i \geq 1.
\]

Using (2) and (3), the value of the counter \( c_i \) becomes

\[
c_i = \frac{\alpha f}{v_i} = \frac{\alpha f}{v_{i-1} + a \cdot \delta t_{i-1}},
\]
which, after substituting \( v_{i-1} = \alpha / c_{i-1} \) and \( \delta t_{i-1} = c_{i-1} / f \), finally becomes

\[
\frac{c_i}{c_{i-1}} = \frac{\alpha f}{c_{i-1}} = \frac{c_{i-1} \cdot R_d}{1 + R_d c_{i-1}} \cdot R_d = \frac{a}{\alpha f^2} .
\]

(5)

In a similar way, in the phase of deceleration, we obtain

\[
\frac{c_i}{c_{i-1}} = \frac{c_{i-1} \cdot R_d}{1 + R_d c_{i-1}} \cdot R_d = -\frac{d}{\alpha f^2} .
\]

(6)

From (5) and (6) we calculate the time delay \( c_i \) (or equivalently \( \delta t_i \)) based on the previous time delay \( c_{i-1}, i \geq 1 \), which is already known. Further, it implies that it is necessary to determine the initial time delay \( c_0 \).

**Calculate the initial time delay**

In order to determine the initial time delay \( c_0 \), we observe motion during the first step. The first pulse (that is generated at the time \( t_1 \)) initiates the first step for the angle \( \alpha \), where \( \alpha \) is given as \( \alpha = 1/2 \cdot \frac{\text{const}}{a \cdot \delta t_i} \) (because the initial speed \( v_{oi} \) is zero, \( v_{oi} = 0 \)). It implies that the speed at the end of the first step is \( v_1 = a \delta t_1 = \sqrt{2 a a} \). Let speed \( v_1 \) be the mean value of the initial speed \( v_{oi} \) and the final speed value in the first step, so the following \( v_o = 0.5(v_{oi} + v_1) = 0.5 \sqrt{2 a a} \) is valid. This, together with (4), gives

\[
c_o = \frac{2 \alpha f}{\sqrt{2 a a}} = f \frac{2\alpha}{\sqrt{a}} .
\]

(7)

In the case when speed is constant, \( v = \text{const.} \), the time delay \( c_i \) is determined (using (4)), as \( c_i = \alpha / v \). For the maximum speed \( v = v_M \), the time delay \( c_i = c_m \) is minimum, and it is given as,

\[
c_m = \frac{\alpha f}{v_M} .
\]

(8)

**Simulation and correction of the algorithm**

In Fig. 4. and Fig. 5., the results of simulations of the algorithm that is proposed by Eqs. (5), (6), (7) and (8) are given (in Fig. 4 whole profile and in Fig. 5 only a part of the profile in the acceleration phase). These results are given for a stepper motor with 64 steps per round, and with the following parameters of motion: \( v_M = 3 \), \( a = 4 \), \( d = 2 \).

**Fig. 4. Results of simulation for different types of algorithms**

**Fig. 5. Results of simulation -the part of phase acceleration**

We have also compared different types of algorithms, as: this algorithm (speed \( v \) - dashed lines) and algorithm that is widely used, and which is proposed in [4] (speed \( v_u \) - light solid line). It is easy to see that they are nearly the same except in the initial steps (one or two steps). In both cases, the step speed is greater than the linear speed (see Fig. 5). In order to decrease the difference from step speed and linear speed, we need to introduce some corrections. If in the acceleration phase, after the time delay \( c_i \) is calculated by (6), we introduce correction in the form

\[
c_i = c_i \left(1 + \frac{0.08}{i}\right), \quad 1 \leq i \leq 5 ,
\]

(9)

we obtain the new step speed \( v_k \) - dark solid line in Fig. 5. We can see that the correction step speed \( v_k \) is much better than the previous uncorrected step speed \( v \) or \( v_u \) (i.e. difference between the step speed \( v_k \) and the linear speed \( v_i \) at the time \( i_i \) is significantly smaller). In the same way, correction
in the last five steps of the deceleration phase is done, so that the time delay in the last steps of this phase is given as

$$c_i = c_0 \left(1 + \frac{0.08}{n-i}\right), \quad 1 \leq n - i \leq 5,$$

where the total number of steps during the desired motion is denoted by $n$, see Fig. 7.

The speed error (it is defined as the difference between analog and step speed) for the cases: this algorithm (uncorrected and corrected) and the algorithm in [4] is shown in Fig. 6. (error in the first step is omitted).

**TRAJECTORY PLANNING**

We need to determine characteristics points of the ramp speed profile where the algorithm changes. Continuous motion and in the case when acceleration and deceleration are the same (for various ways of their change) is described in [8] and [9]. Herein we consider discrete motion for the linear ramp speed profile, see Fig. 7.

For each stepper motor, next parameters must be known: maximum speed, maximum acceleration, maximum start speed and number of steps per round\(^1\) (spr). In addition, at the beginning of each motion the following parameters of motion are also known: desired position $s_{\text{Max}}$ [rad] or the total number of steps $n$, desired speed $v_d$, acceleration $a$ and deceleration $d$. The number of the steps $n$ (see labels in Fig. 7.) is calculated as $n = s_{\text{Max}} / \alpha$, where $\alpha = 2\pi / \text{spr}$ [rad] is an angle for one step. In this paper we assume that all phases of motion are finished during $n$ steps exactly. In order to calculate remaining parameters of the speed profile, we use two different approaches which are dependent on the following: (i) desired speed is reached before the start of deceleration (trapezoidal speed profile, Fig. 7, solid line, $v_d < v_r$) and (ii) deceleration starts before desired speed has been reached (triangular speed profile, Fig. 7, dashed line, $v_d \geq v_r$). As to which of these speed profiles should be used, it is dependent on values $n_a$ and $n_{ad}$, where $n_a$ is the number of steps that are needed to reach desired speed $v_d$ during acceleration and $n_{ad}$ is the first step when the deceleration phase starts. From the reached speed $v_r$ and during next $(n - n_{ad})$ steps, the speed decreases from $v_r$ to the zero value. Now, the cases (i) and (ii) expressed with number of the steps $n_a$ and $n_{ad}$ become: $n_{ad} > n_a$ and $n_a \geq n_{ad}$, respectively. In both cases the speed at the end of acceleration must be the same as the speed at the start of deceleration (due to the continuity of motion), which implies

$$a n_{ad} = d (n - n_{ad}) \Rightarrow n_{ad} = \frac{d}{a + d} n. \quad (11)$$

On the other hand, to achieve the desired speed $v_d = at$, $t \in [0, T_a]$ in the acceleration phase, movement $s_a = n_a \alpha$ is given as

$$n_a \alpha = \frac{1}{2} a t^2 = \frac{1}{2a} \left(\frac{1}{2} (at)^2 \Rightarrow n_a = \frac{v_r^2}{2 a \alpha}. \quad (12)$$

The number of steps $n_d$ when deceleration starts in case (i) is calculated from $a n_a = d(n - n_d)$ which implies $n_d = n - n_a \cdot d / a$, and in the case (ii) it is the same $n_a$, i.e. $n_d = n_a = n_{ad}$ (see more details in Arduino program).

\(^1\) these parameters will be specified by a motor manufacturer
IMPLEMENTATION OF THE ALGORITHM

Based on the consideration in the Sections II and III, the proposed algorithm is implemented by using the microcontroller ATMega328.

As the hardware components we use: Arduino UNO unit, 28BYJ-48 stepper motor (power supply 5V-DC, 4096 spr, because the motor has 64 steps and a gear unit with 64:1 ratio, 64x64=4096) and driver based on the ULN 2003 circuit. A program that realized the above algorithm is given in the appendix of the paper and it is adjusted for Arduino platform of the Atmel microcontroller. The desired data such as angle of positioning, speed, acceleration and deceleration are obtained from a program console using serial communication. The equipment and devices for the experiment are shown in Fig. 8.

CONCLUSIONS

According to the results of the experiment and simulation, we can conclude that the algorithm which is proposed by equations (5)-(10) satisfies the expected results completely. The difference between linear speed and step speed, in comparison with the algorithm in [4], is significantly smaller, so from this point of view the proposed algorithm is better, but it is weaker in relation to the time of computing. Generally, we can conclude that the proposed algorithm is better than the one in [4] for lower speed of stepper motors. Also, it is clear that this speed limit depends on the applied hardware and software. In our case, for the Arduino platform and devices that are shown in Fig.8., this speed limit is close to 5000 steps per second (5kHz).

Appendix - Arduino program

```c
/* for simulation we use stepper motor, model 28BYJ-48, that has 64 steps per round. It includes a gear unit with 64:1 ratio, so that for 1 rotation of the output shaft we need to generate 64x64=4096 steps */
const long f = 1000000; //frequency of the counter;
int Km = 64, reduktor = 64, K = Km*reduktor, kasni=66;
// decape [rad/s]-desired speed, acce [rad/s^2]-acceleration
// dece[rad/s^2]-deceleration, ugao[deg]-angle
float dspeed = 3, acce = 0.5, dece = 1.5, ugao = 0;
double alfa = TWO_PI/K, Co, Cm, Ci,Ra,Rd;
long imaK = 0, N = 0, nad = 0, na = 0, nd = 0, np = 0, broj = 0;
boolean smjer = true, acc = true, stoj = true;
//uni-polar motor - we use microstepping manner of stepping
byte cw[] = {0x01,0x03,0x02,0x06,0x04,0x0C,0x08,0x09};
byte duzcw = sizeof(cw), kk = 0, maska = 0xF0;

void setup(){
  Serial.begin(9600);
  Serial.println("Enter some of the following commands:");
  Serial.print("Uxxxx, - ");
  Serial.println("desired angle[deg]x10");
  Serial.print("Bxx, - ");
  Serial.println("desired speed [rad/s]x10");
  Serial.print("Axx, -");
  Serial.println("acceleration[rad/s^2]x10");
  Serial.print("Dxx, -");
  Serial.println("deceleration[rad/s^2]x10");
  Serial.print("S, - ");
  Serial.println("STOP");
  Serial.print("M,- ");
  Serial.println("MOVE");
  DDRE = 0x0F;//pins 8,9,10,11 of the Arduino are outputs
  PORTB &= 0xF0;//four low bits of PORTB are zero
}

void loop(){
  if (!stoj){   oneStep(smjer);
    delayMicroseconds(Ci - ((Ci == Cm) ? 4 : kasni) );
    Ci = solveC();
    if (++np >= N) {  np = 0; kk = 0; Ci = Co; stoj = true;
    Serial.println("Steps = ");Serial.println(imaK);
    Serial.println("angle = ");Serial.println(imaK*360/K);
    }
  }
}

//one step in the desired direction, where 'desno' is direction
void oneStep(boolean desno){
  if (desno) {PORTB=(PORTB& maska) | cw[kk++]; imaK++;
  }
  else {PORTB=(PORTB & maska) | cw[duzcw-1-kk++];imaK--;
  }
  }
```

Fig. 8. The experimental devices: Arduino, driver and stepper motor
```c
//solve the time delay
double solveC(){
    if (na == 0) return (Cm);
    if ( (np<=na)||(np>=nd) ){ //acceleration and deceleration phase
        double q = 1+( (np<=na)?Ra:Rd)*Ci*Ci; Ci = Ci/q; kasni=66;
        if ( ( (np>= 1 && (np<= 5)) || ((N-np <= 5) && ( N-np >= 1)))
            Ci = Ci*(1+0.08/np) ; kasni=120; //correction
        if (Ci < Cm) Ci = Cm; //using variable 'kasni' we take into
                                //account all delays occurred during
                                //the call and execute of program
        else Ci = Cm; return Ci; //this procedure calculate the parameters of the motion
    }
    else {Ci = Cm; return Ci; //this function is called for every cycle in the loop procedure
        //trajectory planning
        if (acce != 0){// acceleration exists
            nad = (long)(N*dece)/(acce+dece);
            na  = (long)(dspeed*dspeed)/(2*alfa*acce);
            if (nad > na)  { nd = N - na * acce/dece; }//case (i)
            else    { na = nad; nd = na; } //case (ii)
            Co = f * sqrt(2*alfa/acce); Ci = Co;//Co is initial time delay
        } else {//without acceleration
            na =  0;     nd =  N;     Ci =  Cm;
        }
        if (nd < na) { // this is might be due to rounding
            long np=na; na=nd;nd=np;//exchange the values of 'na' and 'nd'
            np = 0;
        }
        //this function is called for every cycle in the loop procedure
    void serialEvent(){ //command of the stepper motor
        if (Serial.available() > 0 ){
            char ch = toupper(Serial.read());
            if ((ch == 'S') || (ch == 'M')) chh = ch;
            if ((ch == 'U') || (ch == 'B') || (ch == 'A') || (ch == 'D')){
                chh = ch;   broj = 0;
            } //if ch is alphanumeric than calculate of the parameter value
            if (chh == 'U') ugao = broj/10.0; init(ugao);
            if (chh == 'B') dspeed = broj/10.0;
            if (chh == 'A') acce = broj/10.0;
            if (chh == 'D') dece = broj/10.0;
            if (chh == 'S') stoj = true;  //STOP moving
                                //continumue the moving
            broj = 0;
        }
    }

    REFERENCES
    [7.] http://picprog.strongedge.net/step_prof/step_prof.html
```