IOP Conf. Series: Materials Science and Engineering **294** (2018) 012055 doi:10.1088/1757-899X/294/1/012055

Generate stepper motor linear speed profile in real time

M Y Stoychitch

University of Banja Luka, Faculty of Mechanical Engineering, Vojvode S. Stepanovica 71, 78000 Banja Luka, Republic of Srpska, Bosnia and Herzegovina

E-mail: mihajlo.stojcic@mf.unibl.org

Abstract. In this paper we consider the problem of realization of linear speed profile of stepper motors in real time. We considered the general case when changes of speed in the phases of acceleration and deceleration are different. The new and practical algorithm of the trajectory planning is given. The algorithms of the real time speed control which are suitable for realization to the microcontroller and FPGA circuits are proposed. The practical realization one of these algorithms, using Arduino platform, is given also.

1. 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 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 Figure 1 the relationships between acceleration a [rad/sec^2], deceleration d [rad/sec^2], speed v [rad/sec] and position s [rad] are shown. On this figure with t_a, t_v, t_d and T (all in [sec]) are labeled: the time of acceleration, time of motion of constant speed, time of deceleration and total time, respectively. Also, with N_a, N_v, N_d and N (all in [step]) are denoted number of steps on the mentioned time intervals, see Figure 1.

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

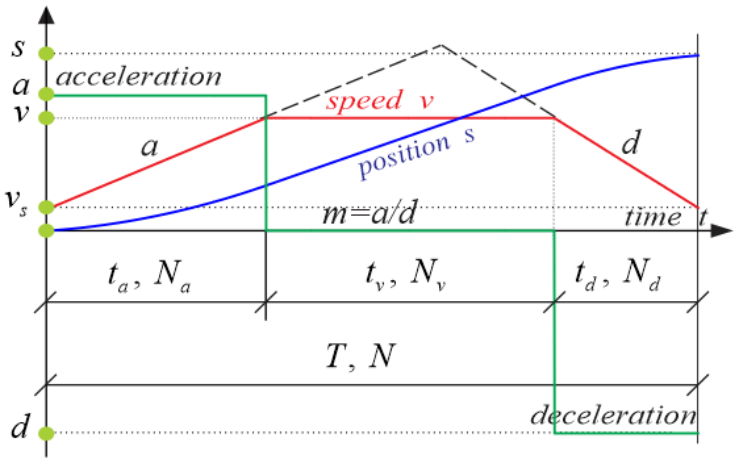

Figure 1. Change of speed, position and acceleration

(pulses). It means that the main problem of speed control is to determine instants of the time t_i (in [sec] when pulses (steps) are generated. If speed v [rad/sec] is constant $v = const.$ (independent of whether it is large or small), it is very easy to determine these instants (or equivalently, generate the pulses). In this case the time delays δt_i, between two arbitrary adjacent pulses, are the same and they are given as, $\delta t_i = \alpha / v = const.$, where α [rad] is the angle of the rotation motor shaft for every step. But, if the speed is variable, $v_i \neq const.$, it is more difficult to determine the instants when we need to generate pulses because the time delay between two successive pulses is changed, $\delta t_i = \alpha / v_i \neq const.$. In the case when acceleration/deceleration are constant, the speed is changed linearly, but the time interval δt_i between two adjacent pulses is not linear. So, we need to determine the time t_i, $i = 0,1,2,\ldots$, when pulses are generated and that ensure linear change of speed. It is shown for the acceleration phases in Figure 2.

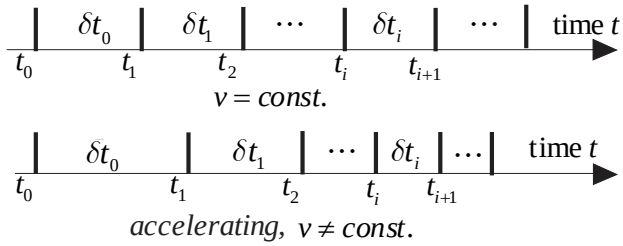

Figure 2. Constant speed and accelerating step sequences

Generally, the problem of generating appropriate speed profile to a stepper motor we can split in the two main tasks: 1) to determine a shape the speed change and 2) generate impulses in appropriate instant of time in order to achieve this speed.

And, at the end of this section, a few words about the structure of the paper. The second section is related to the trajectory planning (solving of the first task), where based on some input data about the motion and properties of the selected stepper motor we calculate characteristic points of a movement, i.e. determining a speed profile. In the third section are given two algorithms based on which to generate pulses (solving of the second task). One of these algorithms is appropriate for realization using a microcontroller and another using FPGA circuits. In the fourth section we will propose a way of realization one of these algorithms. And finally, in the fifth section, are given some conclusions, and in the appendix is given an Arduino program of practical realization one of algorithms which is proposed in the previous sections.

2. Trajectory planning

Since the change of speeds are linear, then trajectory planning means that need to determine the number of steps during which the following phases are realized: acceleration N_a, motion of constant speed N_v, deceleration N_d as well as the number of steps of the total movement N. In doing so we assume that the total movement s and the ratio $m = a/d$ between acceleration a and deceleration d are given. Usually, by the stepper motor manufacturer are specified the next parameters: number of steps per round K [step], starting speed (the stepper motor can go from 0 to this speed in no time) v_s [rad/sec], maximum speed v_M [rad/sec] and the maximum acceleration[1] a_M [rad/sec^2].

Now, our problem can be described as: we need to move a stepper motor for $N(=[s/\alpha])$ steps (here the symbol $[\circ]$ denotes the nearest integer), $\alpha = 2\pi/K$ [rad], so the start speed not higher then v_s, the acceleration a not higher then a_M, during the motion the speed v must never exceed the maximum speed v_M and the ratio acceleration/deceleration is m.

At the beginning of trajectory planning we assume that the total movement s can be achieved with a triangular speed profile, which include only two phases: acceleration and deceleration (we also assume that the acceleration a during the acceleration phase is maximum, $a = a_M$). In this case the movement s is (see Figure 3, left)

$$s = s_o + s_a + s_d = v_s T + \frac{1}{2} a t_a^{\,2} + a t_a t_d - \frac{1}{2} d t_d^{\,2} \tag{1}$$

where $s_o = v_s T$, $s_a = \frac{1}{2} a t_a^{\,2}$ and $s_d = a t_a t_d - \frac{1}{2} d t_d^{\,2}$ are the motions: due to the start speed v_s during the time T, due to acceleration a on the time t_a and due to deceleration d on the time t_d (we assume that the deceleration d not limited, but the time $t_d = T - t_a > 0$), respectively. Since the speed at the end of the acceleration phase is the same as the speed at the beginning of the deceleration phase, then the next equation

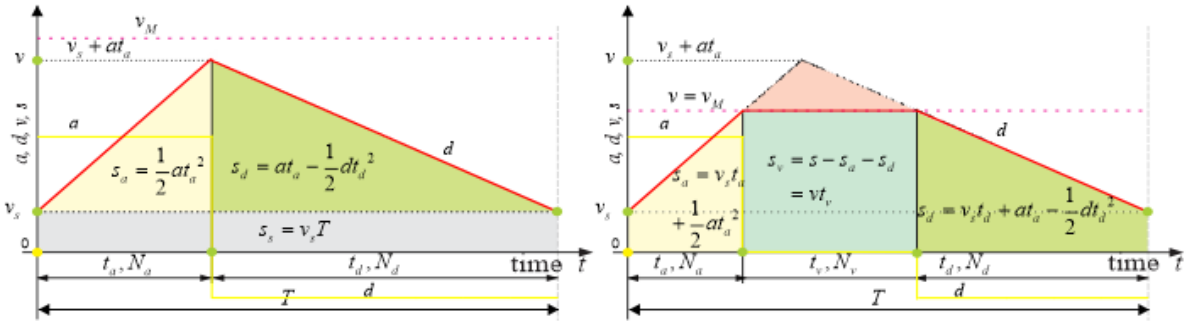

Figure 3. Triangular (left) and trapezoidal (right) speed profile

$$a t_a = d t_d = d\ (T - t_a) = \frac{a}{m}\ (T - t_a) \Rightarrow t_a = \frac{T}{1+m} \tag{2}$$

is valid. From the equations (1) and (2) it is obtained

$$2\ (s - v_s T) = \frac{a T^2}{1+m} \Rightarrow T^2 + \frac{2 v_s\ (1+m)}{a} T - \frac{2 s\ (1+m)}{a} = 0, \tag{3}$$

from where we get the solving of the above square equation per T, as

[1] If the manufacturer is specified the speed in [step/sec] and acceleration in [step/sec^2] then relation between them are: 1[step/sec] = $2\pi/K$ [rad/sec] and 1[step/sec^2] = $2\pi/K$ [rad/sec^2]

$$T = -\frac{v_s\ 1+m}{a} + \sqrt{\left(\frac{v_s\ 1+m}{a}\right)^2 + \frac{2s\ 1+m}{a}}. \tag{4}$$

Using this T we calculate the time of the acceleration phase $t_a = T/\ 1+m$ and checking that the speed $v = v_s + at_a$ at the end of the acceleration phase is greater than the maximum allowed speed v_M.

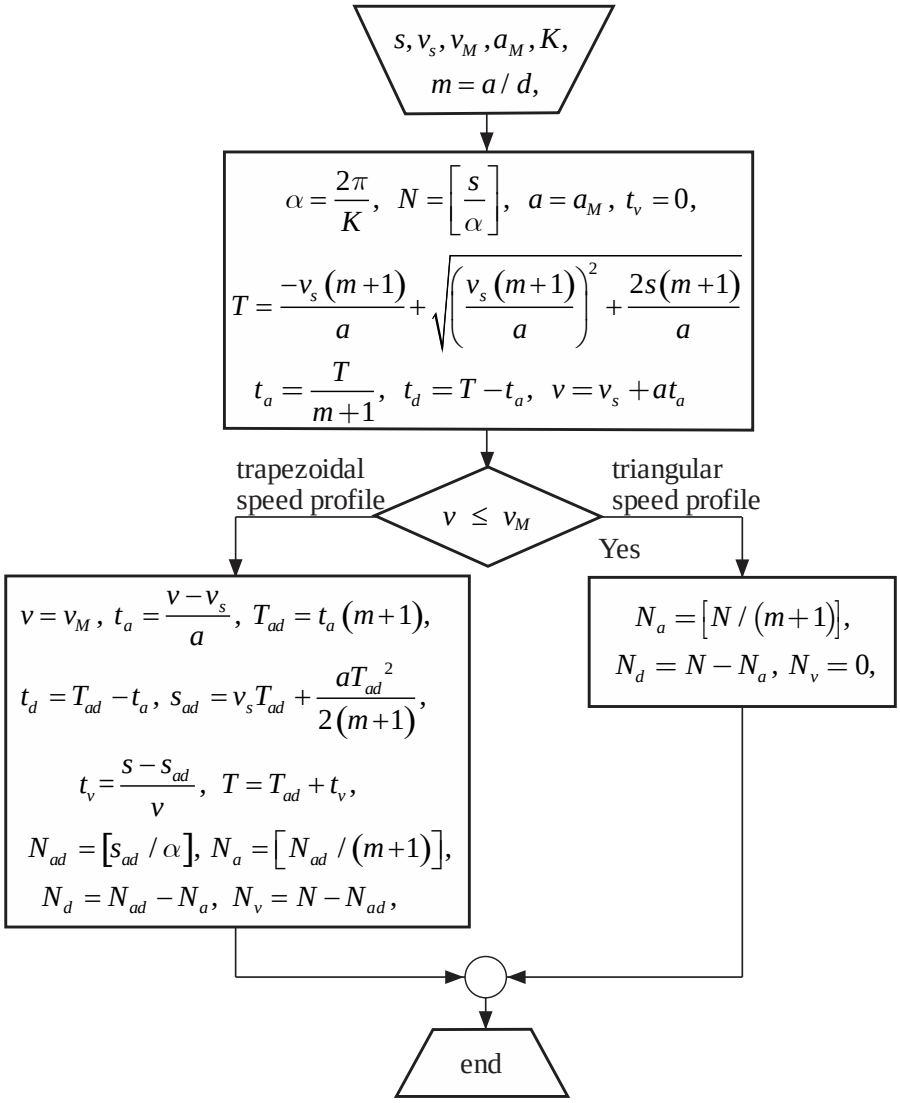

Figure 4. Flow chart of trajectory planning

If it is $v \le v_M$, then the initial assumptions is true, so that the total movement can be realized with a triangular speed profile. In this case the number of steps in the acceleration phase N_a and in the deceleration phase N_d are calculated as

$$N_a = [N/\ 1+m\ ],\ N_d = N - N_a. \tag{5}$$

The flow chart of this algorithm of the trajectory planning is shown on Figure 4. However, when $v \le v_M$ is not true, then is the total movement s can be realized with trapezoidal speed profile (see Figure 3, right). In this case, we additionally need to include a third phase, the phase of moving with constant and maximum speed, $v = v_M$. Now, based the speeds v_s and v and the accelerating a we calculate the

time of the acceleration phase $t_a = v - v_s / a$ and the total the time $T_{ad} = t_a\ 1+m$ of the both phases, of acceleration and deceleration, together. Using this T_{ad} we calculate the total movement s_{ad} in the phases acceleration and deceleration (see (3)) , as

$$s_{ad} = v_s T_{ad} + \frac{a T_{ad}^{\ 2}}{2\ 1+m}, \tag{6}$$

so that the movement s_v in the phase of constant speed is $s_v = s - s_{ad}$. The time t_v during the phase of constant speed is realized and the total time T of all phases are given as

$$t_v = \frac{s - s_{ad}}{v}, \quad T = T_{ad} + t_v . \tag{7}$$

Now, in the case of trapezoidal speed profile, using of the known movements s, s_{ad} and s_v and the step α , we can calculate the number of steps: N_a, N_d and N_v in the each phases of the motion, as:

$$N_{ad} = s_{ad} / \alpha\ ,\ N_a = \left[N_{ad} /\ 1+m\ \right],\ N_d = N_{ad} - N_a,\ N_v = N - N_{ad} . \tag{8}$$

The complete algorithm of the trajectory planning for the general case of the trapezoidal speed profile is shown on Figure 4.

3. Generate of the pulses - Algorithms

There are two methods – two group of algorithms for calculating instants of time when pulses must be generate. They are named as: *"time per step"* and *"steps per time",* that are described in [3-5], [8], [9] and partly in [6], [7], respectively. Operation mode of the first group of algorithms can be briefly described as: (*i*) calculate time period to next pulse, (*ii*) wait until that much time period elapses, (*iii*) generate next pulse and (*iv*) go back to step (*i*) and repeat until desired number of pulses is over. The another group of algorithms have an operation mode that is significantly differently, and briefly can be described as: (*i*) check the current time, (*ii*) multiply it by speed, to get expected current position, (*iii*) if difference from expected and actual current position greater than or equal one step, then to generate new pulse and (*iv*) repeat this until the final desired position is achieved.

Since in these algorithms all calculations are realized in real time (trajectory planning is not in real time), i.e. between two steps (pulses), then the basic goal is to find faster algorithms, because this allows the control of higher speed motors. Also, the difference in the speed generated through these algorithms and theoretical speeds for the given profile must be as small as possible.

The first group of algorithm is suitable for controllers that are implemented by the microcontroller, while the second group is suitable for the controllers that are implemented via FPGA circuits. In this paper are described the both groups of algorithms. From the first group is given the algorithm that is some modification of the algorithm that is proposed in [8], [9], while from the second group is proposed an algorithm that partly described in [6], [7].

3.1. Calculate times when the pulses are generated using time per steps algorithm

The first pulse (step) controller generates at the start of motion, at the start of the phase of acceleration, i.e. at the time t_0, see Figure 5. After the first pulse is generated, the controller needs to calculate the time period δt_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 $v_o = v_s$, 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. Since after each pulse the motor makes one step for the angle α , so that is

$$\alpha = v_i \cdot \delta t_i \Rightarrow \delta t_i = \frac{\alpha}{v_i}, \tag{9}$$

where the v_i is speed at an arbitrary instant of time t_i and δt_i the time delay between two successive

instants of time, t_i and t_{i+1}. In the case when the controller is realized by using a microcontroller, the

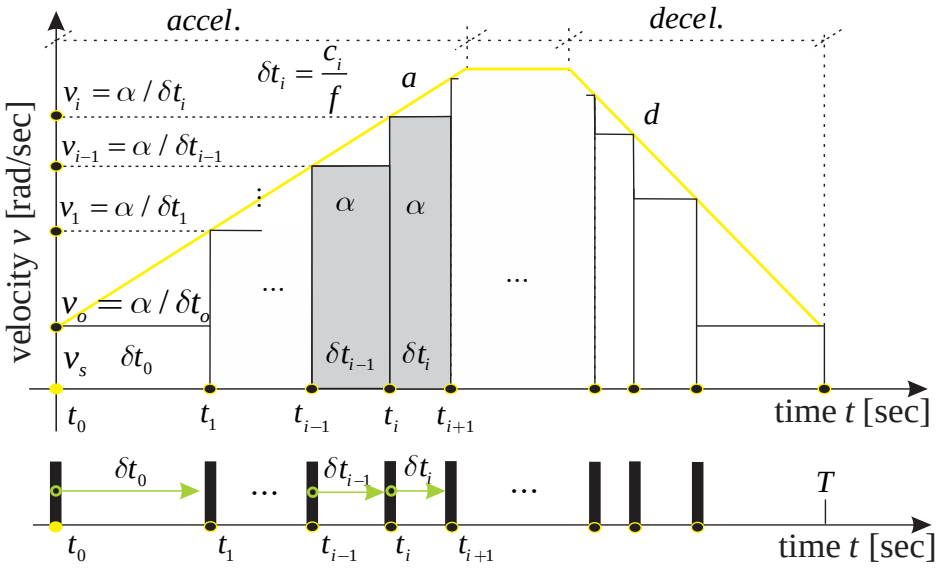

Figure 5. Calculate the time when pulses are generated

required time delay δt_i is implemented using counter c_i that counts impulses of known frequency f, so it is $\delta t_i = c_i / f$. 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 bright lines in Figure 5.). Thus, the speed v_i at the arbitrary instant of time t_i, and in the phase of acceleration, becomes

$$v_i = v_{i-1} + a \cdot \delta t_{i-1},\ i \geq 1. \tag{10}$$

Using above equations, the value of the counter c_i becomes

$$c_i = \frac{\alpha f}{v_i} = \frac{\alpha f}{v_{i-1} + a \cdot \delta t_{i-1}} = \frac{\alpha f}{\frac{\alpha f}{c_{i-1}} + a\frac{c_{i-1}}{f}} \Rightarrow c_i = \frac{c_{i-1}}{1 + R_a c_{i-1}^{\ 2}},\ R_a = \frac{a}{\alpha f^2}. \tag{11}$$

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

$$c_i = \frac{c_{i-1}}{1 + R_d c_{i-1}^{\ 2}},\ R_d = -\frac{d}{\alpha f^2}. \tag{12}$$

From (11) and (12) we can see that the time delay c_i (or equivalently δt_i) is calculated based on the previous time delay $c_{i-1}, i \geq 1$, which is already known and what is very suitable for realization. Further, it implies that it is necessary to determine the initial time delay c_o and the time delay during of the phase of constant speed (if exist) c_m. These time delays are obtained based on the start speed v_s and the maximum speed v_M, respectively, as

$$c_o = \frac{\alpha f}{v_s} \quad \text{and} \quad c_m = \frac{\alpha f}{v_M}. \tag{13}$$

After detailed analysis and simulation of the above proposed algorithm, in [9] was shown that the speed profile that generated by stepper motors more exact (closer to the theoretical) if in the first and last five steps are introduced some corrections. These corrections are realized by using the next equations (i – the current number of steps)

$$c_i = c_i\left(1+\frac{0.08}{i}\right),\ 1\le i\le 5 \quad \text{and} \quad c_i = c_i\left(1+\frac{0.08}{N-i}\right),\ 1\le N-i\le 5\,. \tag{14}$$

where are i the current number and N the total number of steps. For more details see paper [9]. On the Figure 6 is given flow chart of this algorithm (without the corrections that are given by (14)).

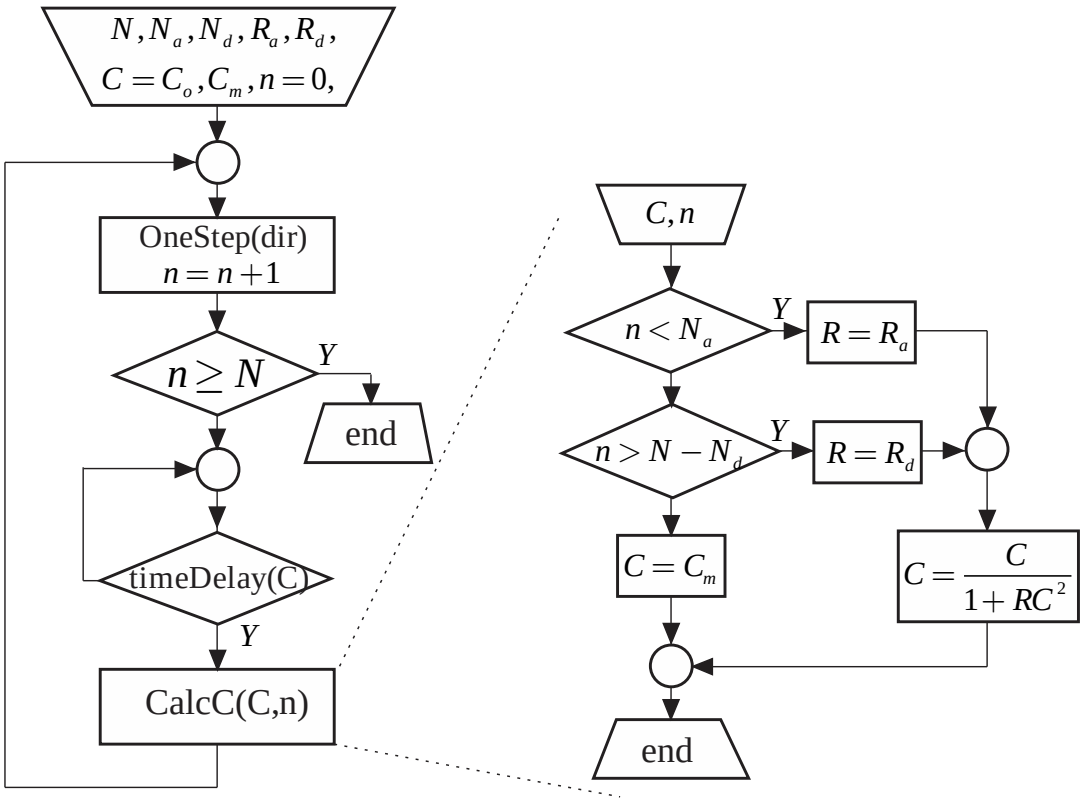

Figure 6. Flow chart of the algorithm from section 3.1

3.2. Calculate times when the pulses are generated using steps per time algorithm

Calculating the time delay between two pulses using algorithm that is given by equations (11) and (12) require two multiplication, one addition, and one division (all with float number of double precision) for each step. This algorithm it is not suitable for implementation by using FPGA circuits, because multiplication and division with them is difficult or very slow. Unlike them, the addition and subtraction with this circuits is realized very easily and very fast. Therefore, for FPGA-based controllers, it is necessary to find another algorithms, in which for all calculations in real time, are used only addition and subtraction operations.

It is known from theory that between the acceleration $a(t)$, velocity $v(t)$ and position $p(t)$, at some time interval $[0,T]$, the following relations apply:

$$v(t)=\int_0^t a(\tau)d\tau \quad \text{and} \quad p(t)=\int_0^t v(\tau)d\tau,\ \ t\in[0,T]\,. \tag{15}$$

Above equations are given in a continuous domain. In a discrete domain with a constant time period T_s, so it is $T = MT_s$, where T_s is very small real number and M is integer, these equations at discrete time interval $t_k = kT_s,\ k = 0,1,2,\ldots,m,\ m\le M$ become:

$$v(kT_s)=v(k)=\sum_{k=0}^{m}a(k)T_s \text{ and } p(kT_s)=p(k)=\sum_{k=0}^{m}v(k)T_s. \tag{16}$$

According equation (16), if it is dimension of the acceleration a in $\left[\text{steps}/T_s^2\right]$, then dimension of the velocity v is in $\left[\text{steps}/T_s\right]$ and dimension of the position p is in $\left[\text{steps}\right]$ (for example: if it is T_s in $\left[\text{ms}\right]$ - milliseconds, then a in $\left[\text{steps}/\text{ms}^2\right]$, v in $\left[\text{steps}/\text{ms}\right]$ and p in $\left[\text{steps}\right]$).

From (16) it can be seen that for calculating, for example $p(k)=\sum_{k=0}^{m}v(k)T_s$, necessary addition and multiplication with T_s. But if this equation it is solved at every T_s, then multiplication with T_s is not necessary, since at every instant of time T_s the new part of position $v(k)\bullet 1T_s=v(k)\bullet 1$ is added to the existing value of position.

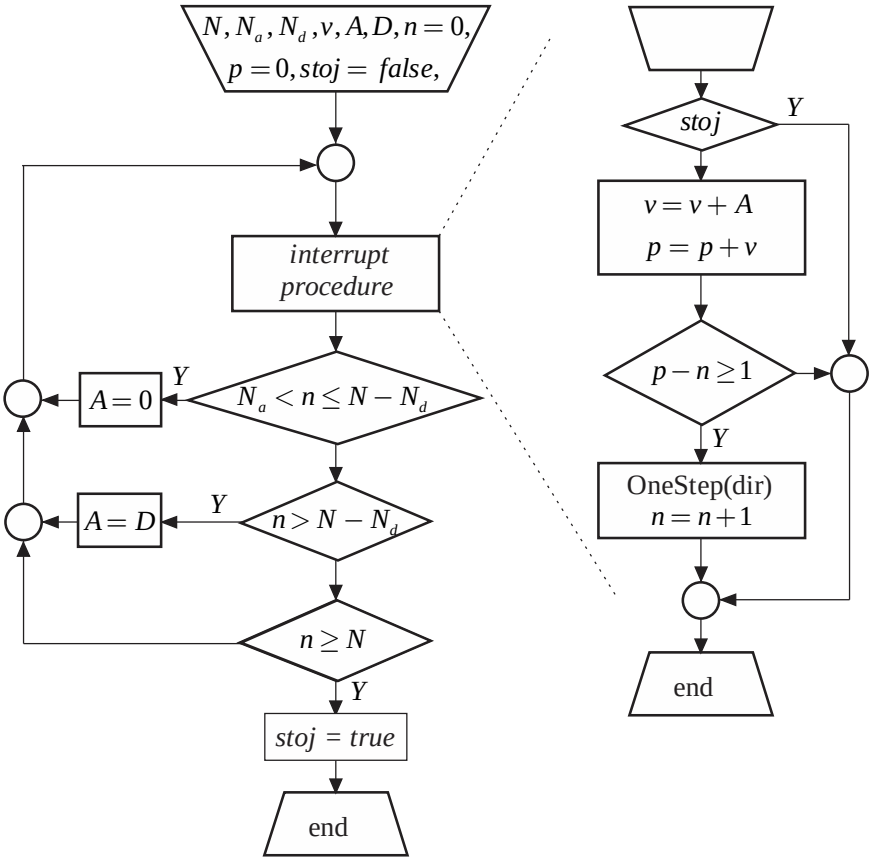

Figure 7. Flow chart of the algorithm from section 3.2

And finally, let the dimensions are: of acceleration a $\left[\text{steps}/T_s^2\right]$, of speed v $\left[\text{steps}/T_s\right]$ and of positions p $\left[\text{steps}\right]$, and let at the initial time $t_0=0=kT_s, k=0$, their values are: $acceleration=A$, $speed=V_s$, $position=0$ and number of steps $steps=0$, then by solving the next equations:

$$\begin{aligned} speed &= speed + acceleration \\ position &= position + speed \end{aligned} \tag{17}$$

at every instant of time T_s, it is equivalent to the solution of the equations (16). The value of *position* that is given from (17) it is the same to the expected current position, and value of the *steps* is the same

to the actual current position. The instant of time when a new pulse (step) is generated now is determined from the next requirement: if (*position-steps* >= 1) then the new pulse is generated and the new actual current position *steps* becomes *steps=steps+1*. Flow chart of this algorithm is shown in Figure 7. In this flow chart the next label: *p – position*, *n – steps*, *v- speed* and *A-acceleration* are used.

The implementation of the procedure (17) (i.e., the solution of the equation (16)) is easiest to accomplish using an interrupt, where interrupt occurs by every T_s instant of time. The value of the time period T_s depends of the motion parameter (K, a_M, v_M) and of type and speed of a processor, which is the problem of special analysis that is not the subject of this paper.

4. Implementation of the steps per time algorithm

Based on the consideration in the sections II and III, the algorithm that is proposed in section 3.2 (step per time algorithm) is implemented by using the microcontroller ATMega328. In this implementation as a hardware we use the 28BYJ-48 stepper motor (power supply 5V-DC, 4096 steps per round, because the motor has 64 steps and a gear unit with 64:1 ratio, so that 64*64=4096), the driver based on the ULN 2003 circuit and the Arduino UNO as a platform of the ATMega328 microcontroller, see Figure 8.

Figure 8. Devices for the experiment: Arduino, driver and stepper motor

Arduino program through which, based on the entered data, the planning of the trajectory and generation of the pulses using the steps per time algorithm (section 3.2) is given in the appendix. In this program we use the interrupt frequency F = 10 000 Hz, so that the time period $T_s = 1/F = 0.1$ ms. Since the inputs data for acceleration, speed and position the next units of measure: [rad/sec^2], [rad/sec] and [deg] are used, respectively, then it must be translated into new units: [steps/T_s^2], [steps/T_s] and [steps]. This conversion are achieved using the following relations: acceleration [rad/sec^2] $\times K / 2\pi / F^2 =$ acceleration[steps/T_s^2], speed[rad/sec] $\times K / 2\pi / F =$ speed[steps/T_s] and position [deg] $\times K / 360 =$ position [steps].

The given program enables the input and change of all parameters of movement, as well as their displaying. After the reset, the initial values are loaded, and the current position of the stepper motor shaft is considered as a zero position. After each next entry of the new position, the program determines the direction and value of the angle from the current to the new position and plans the trajectory between these two positions.

5. Conclusion

In this paper, problem of the control of a stepper motor with linear change of the speed is considered. This problem involves solving two tasks. The first task (that is solved in section 2) is the trajectory planning so that all requirements relating to the parameters of motor and motion are fulfilled, while the second task is the real-time control of the motor motion, according to the requirements from the first task (this task is solved in section 3). The general case is analyzed, when the absolute values of the acceleration in the phases of acceleration and deceleration are different. Two completely different algorithms related to real-time control of motors are proposed. One of these algorithms is adapted for controllers based on FPGA circuits, while both algorithms are suitable for controllers that are based on a microcontroller.

All the theoretical considerations in this paper were simulated by experiment. The results of the experiment confirm the proposed theoretical considerations and show that are all proposed algorithms very fast and simple.

Appendix

Arduino program

```
#define K   4096   //number of steps per one revolution
#define F   10000  //interrupt frequency  10kHz
//
float s = TWO_PI,vs = 0.5,vM = 2,aM = 2.0, m = 2,T,ta,tv,td,ss,aa = aM,vv = vM;//a[rad/sec^2],v[rad/sec],s[rad]
double alfa = TWO_PI/K, Af = aM/alfa/F/F, Vsf = vs/alfa/F, Df = -Af/m, //Af[steps/Ts^2],Vsf[steps/Ts]
    brzina = Vsf, pozicija = 0; //
//cN-current number of steps, dN-desired number of steps
long N, Na, Nv, Nd, broj, cN, np = -1;
byte cw[] = {0b0001,  //A,
             0b0011,  //A,B
             0b0010,  //B,
             0b0110,  //B,C
             0b0100,  //C,
             0b1100,  //C,D
             0b1000,  //D,
             0b1001}; //D,A
byte dcw = sizeof(cw), kk = 0, maska = 0xF0, ledPin = 13;
char ch, chh;
boolean smjer = true, stoj = true;
//
void setup() {
 Serial.begin(9600);
 Serial.print("CONTROL of STEPPER MOTOR\n");
 Serial.print("Commands: Pxxxx, desired position s[deg]X10 \n");
 Serial.print("        Bxxx, start speed vs [rad/sec]X10 \n");
 Serial.print("        Vxxx, max. speed  vM [rad/sec]X10 \n");
 Serial.print("        Axxx, max. acce  aM [rad/sec2]X10 \n");
 Serial.print("        Rxxx, the ratio m*10 of acce/dece \n");
 Serial.print("        S,-stop, M,- move, W-write paramet. \n");
 //
 DDRB  = 0b00001111;   //pins 8,9,10,11 are outputs
 PORTB = PORTB & 0xF0; //all outputs are zero
 pinMode(ledPin,OUTPUT); //led pin, this pin start and stop is labeled
 //settings interrupt, every 1/10[ms] the interrupt has occurred
```

```
  cli();                // disable global interrupts
  TCCR1A = 0;           // set entire TCCR1A register to 0
  TCCR1B = 0;           // same for TCCR1B
  TCNT1  = 0;           // initialize counter value to 0
  // dF-desired interrupt feq., pS- prescaler, cR-compare match register
  // cR=[16*10^6/(pS*dF)]-1
  // if dF=10^4, pS=1(no prescaler) => cR=[16*10^6/(1*10^4)]-1=1599
  // set compare match register to desired timer count:
  OCR1A = 1599;
  TCCR1B |= (1 << WGM12);  // turn on CTC mode:
  TCCR1B |= (1 << CS10);   // Set CS10 bit for no prescaler:
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt:
  sei();                // enable global interrupts
}
// interrupt procedure, every 1/10000 [sec]=1/10[ms] program calls this procedure
ISR(TIMER1_COMPA_vect){
  //
  if (stoj) return;
  brzina   = brzina + Af;
  pozicija = pozicija + brzina;
  if (pozicija - np >= 1) {
    oneStep(smjer);
    np++;
  }
}
//motor makes one step, if right = 1, then direction = CW
void oneStep(boolean right){
  if (right) { PORTB = (PORTB & 0xF0) | cw[kk++]     ; cN++; }
  else       { PORTB = (PORTB & 0xF0) | cw[dcw-1-kk++]; cN--; }
  if (kk == dcw) kk = 0;
}
//
void loop() {
  if (np >= Na && np < N-Nd) Af = 0;  //speed is constant
  if (np >= N-Nd)          Af = Df; //begin of deceleration
  if (np >= N) {
    stoj = true;
    Serial.print("..............................................\n");
    Serial.print("STEPS    = ");   Serial.print(cN); Serial.print(" [steps]\n");
    Serial.print("ANGLE    = ");   Serial.print(cN*360.0/K,2);Serial.print(" [deg]\n");
    Serial.print("POSITION = ");   Serial.print(float(cN*alfa),2);Serial.print(" [rad]\n");
    np = -1; pozicija = 0; brzina = Vsf;
    digitalWrite(ledPin,LOW);
  }
}
//this procedure enables the entry of new data
void serialEvent(){
  if (Serial.available() > 0){
    ch = toupper(Serial.read());
    if (ch == 'P' || ch == 'B' || ch == 'V' || ch == 'A'  || ch == 'W'
                || ch == 'R' || ch == 'S' || ch == 'M') chh = ch;
    if (ch >= '0' && ch <= '9') broj = broj*10 + int(ch - '0');
    //analiza komandi
```

```
   if (ch == ','){
     if (chh == 'P')  {
       s    = broj/10.0;      // s[deg]
       stoj = planTrajek(s);
     }
     if (chh == 'B')  vs = broj/10.0;
     if (chh == 'V')  vM = broj/10.0;
     if (chh == 'A')  aM = broj/10.0;
     if (chh == 'R')  m  = broj/10.0;
     if (chh == 'S')  stoj = true;
     if (chh == 'M')  stoj = false;
     if (chh == 'W')  pisiSve();
     broj = 0;
   }
  }
}
//this procedure displaying current data
void pisiSve(){
 Serial.print("\n.....\n");
 Serial.print("Displacement   s = ");Serial.print(s,2); Serial.print("[deg]\n");
 Serial.print("Start. speed  vs = ");Serial.print(vs,2);Serial.print("[rad/sec]\n");
 Serial.print("Max.  speed   vM = ");Serial.print(vM,2);Serial.print("[rad/sec]\n");
 Serial.print("Max. accler.  aM = ");Serial.print(aM,2);Serial.print("[rad/sec^2]\n");
 Serial.print("Ratio of a/d   m = ");Serial.print(m,2); Serial.print("\n");
 Serial.print("Stop = ");Serial.print((stoj)? 1:0);
 Serial.print(", Move = ");Serial.print((stoj)? 0:1);Serial.print("\n");
 Serial.print(".....\n");
}
//trajectory planning procedure
boolean planTrajek(float _s){
 Serial.print("\nDATA: newP = ");Serial.print(_s,2); Serial.print("[deg], ");
 Serial.print(", curP = "); Serial.print(float(cN*360.0/K)); Serial.print("[deg]");
 //
 N = int(_s*K/360.0) - cN ; //difference from desired and current position [steps]
 if (N < 0)  smjer = false;
 else       smjer = true;
 if (N == 0) {np = -1; return true;} //
 //
 N = abs(N); float s_ = N*alfa;
 T  = sqrt( pow( (vs*(1+m)/aa),2)+2*s_*(1+m)/aa  )-vs*(1+m)/aa; //N = round(s/alfa);
 ta = T/(1+m); vv = vs + aa*ta; //td = (T-ta); tv = 0;
 if (vv <= vM){//TRIANGULAR PROFILE
   Na = int(N/(1+m)); Nd = N - Na; Nv = 0;
 }else{       //TRAPEZOIDAL PROFILE
   vv = vM; ta = (vM-vs)/aa; T = (1+m)*ta; ss = vs*T+aa*pow(T,2)/(2*(1+m));
   //ss - is tha part of movement in the phases of acceleration and deceleration only
   //np - number of steps on the movement ss
   np = int(ss/alfa); T = (1+m)*ta+(s_-ss)/vv; //tv = (s_-ss)/vv; td = T - ta - tv;
   Na = int(np/(m+1)); Nd = np - Na; Nv = N - np; //Nv - number of steps in the phase v=const.
 }
 //Vsf [steps/0.1ms] - the start speed, Af[steps/(0.1ms)^2]-acceleration, Df-deceleration
 Vsf = vs/alfa/F; Af = aa/alfa/F/F; Df = -Af/m;
 brzina = Vsf; pozicija = 0;  np = 0;
```

```
 Serial.print(", dN = "); Serial.print(N); Serial.print(", Direction = ");
 Serial.print((smjer)?"CW.\n":"CCW.\n");
 digitalWrite(ledPin, HIGH);
 return false;
 //
}
```

References

[1] *** Industrial Circuits Application Note, *Stepper Motor Basis*, http://solarbotics.net/library/pdflib/pdf/motorbas.pdf

[2] Condit R, Douglas W Jones, *Stepping Motors Fundamentals*, Microchip AN907, http://homepage.cs.uio- wa.edu/~jones /step/ an907a.pdf

[3] Austin D 2005 *Generate stepper motor speed profiles in real time,* Embedded Systems Programming, www.embedd ed.com/56800129

[4] *** Atmel Corporation, *AVR446: Linear speed control of stepper motor*, Application note, http://fab.cba.mit.edu/ classes/ MIT/961.09/prjects/i0/doc8017.pdf

[5] Eiderman A, *Real Time Stepper Motor Linear Ramping Just by Addition and Multiplication,* http://hwml.com/ LeibRamp.pdf

[6] Ranade P 2009 *Linear Motor Control Witdout the Math,* SPJ Embedded Technologies, http://www.eetimes.com/document.asp?doc_id=1276928

[7] *** *Stepping Motion Profiles in Realtime*, http://picprog.strongedge.net/step_prof/step-profile.html

[8] Stoychitch M Y 2012 *Linear Speed Control of Stepper Motor in Real Time,* XI International SAUM Conference on Systems Automatic Control and Measurements, Niš, Serbia, November 14th-16th, pp 406-409, http://www.ni.ac.rs/images/stories/events/SAUM_2012_abstracts.pdf

[9] Stoychitch M Y 2013 An Algorithm of Linear Speed Control of a Stepper Motor in Real Time, *Annals of Faculty Engineering Hunedora, International Journal Engineering* **XI** (3) 51-56, http://annals.fih.upt.ro/pdf-full/2013/ANNALS-2013-3-06.pdf