S-curve Trajektoriengenerator

Generierung und Implementierung einer beschleunigungsbeschränkten Trajektorie für die Sollwertvorgabe

Author: Bernd Heufelder

May 28, 2021

Contents

1 Einführung/Motivation

Im Gegensatz zu einem Sollwertsprung kann durch Vorgabe einer Solltrajektorie der Verlauf zwischen Anfangssollwert und Endwert beliebig vorgegeben werden. Ziel ist es die Ordnung der überführenden Trajektorie so niedrig als möglich und so groß als nötig zu setzen.
Zur Sollwertüberführung der Heizertemperatur beim Projekt TGB2 hat sich gezeigt, dass eine lineare Überführung zwischen einer Start- und Endtemperatur für die meisten Anwendung nicht ausreichend ist. Es wird daher im Folgenden die Generierung einer Trajektorie 2. Ordnung behandelt.
Eine Trajektorie 2. Ordnung wird so verstanden, dass die 2. Ableitung des Sollwertverlaufs konstant ist und der Sollwertverlauf ein Polynom 2. Ordnung darstellt. Wir beschränken also die Beschleunigung des Verlaufs auf einen konstanten Wert a_{max}.

$$a = [0, \pm a_{max}]. \tag{1}$$

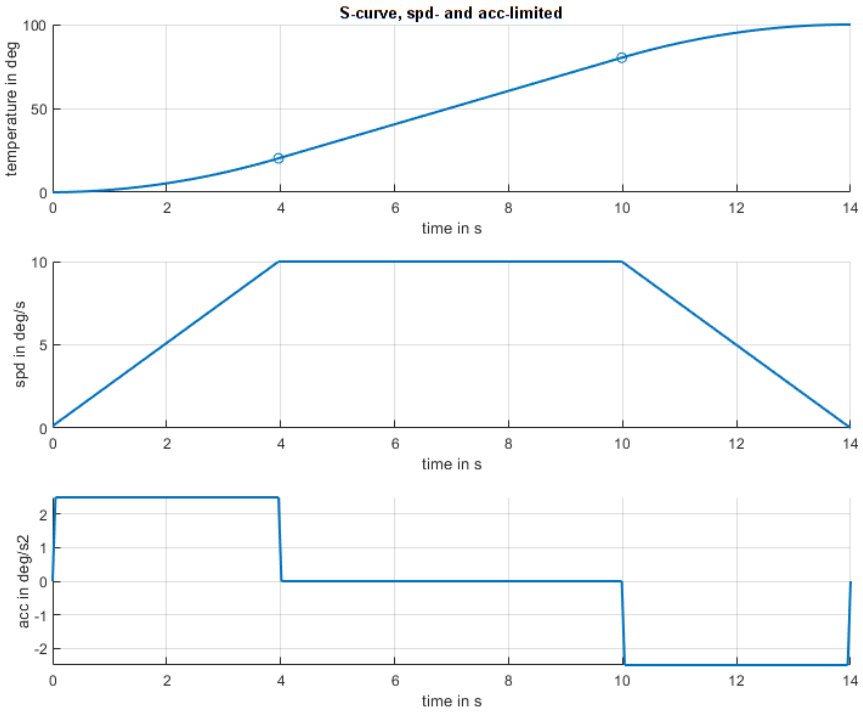

Figure 1: Beschleunigungsbeschränkte Trajektorie mit trapezförmigem Geschwindigkeitsverlauf

Des weiteren sei die 1. Ableitung (Geschwindigkeit) des Verlaufs auf einen vorzugebenden Wert v_{lim} wie folgt beschränkt

$$v = \{v \mid -v_{lim} < v < v_{lim}\}. \tag{2}$$

Mit den Bedingungen (1) und (2) ist der zeitlich kontinuierliche Verlauf des Sollwerts durch die Funktion

$$T(t) = T_0 + v \cdot t + \frac{1}{2} a \cdot t^2 \tag{3}$$

gegeben. Die Berechnung der S-curve erfolgt durch Integration des Geschwindigkeitsverlaufs. Um diesen Verlauf erstellen zu können, ist die Dauer der Beschleunigungsphasen nötig, welche den gewünschten S-Verlauf ergeben.
Als Ansatz für die Erzeugung des Geschwindigkeitsverlaufs, wird zwischen Beschleunigungs-, Konstantgeschwindigkeits- und Verzögerungsphase unterschieden. Bei geringen Überführungsdistanzen kann es abhängig von v_{lim} und a_{max} vorkommen, dass v_{lim} nicht erreicht wird und es somit keine Konstantgeschwindigkeitsphase gibt. Diese Variation wir anhand einer Fallunterscheidung abgefangen.

2 Fallunterscheidung

Es gibt den Fall, dass die maximale Geschwindigkeit erreicht wird (Fall 1) und es kann vorkommen, dass eine Konstantgeschwindigkeitsphase nicht auftritt (Fall 2).

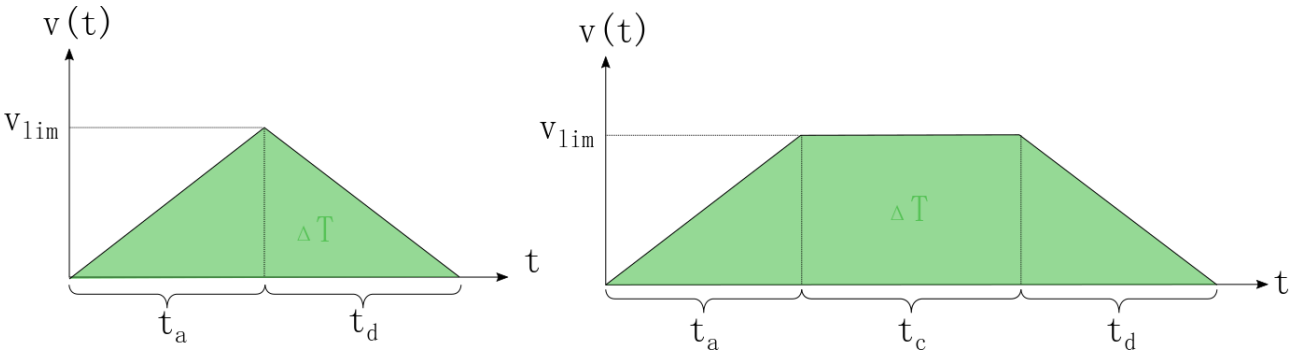

Figure 2: (links) Grenzfall, bei welchem noch keine Konstantgeschwindigkeitsphase auftritt

Falls der zurückgelegte Weg während der Beschleunigungs- und Verzögerungsphase kleiner als die gewünschte Überführungsdistanz ist, dann tritt Fall 1 ein. Wenn diese Bedingung nicht erfüllt ist, dann tritt Fall 2 ein.

Bedingung zur Fallunterscheidung

Fall 1 tritt ein, sobald die Fläche unter dem Geschwindigkeitsdreieck größer als ΔT ist. Mit $t_a \cdot v_{lim} = \Delta T$ und $t_a = \frac{v_{lim}}{a_{max}}$ als die Zeit bis zum Erreichen des Geschwindigkeitslimits, ergibt sich der Grenzfall, bei welchem noch keine Konstantgeschwindigkeitsphase auftritt. Es ergbit sich folgende Bedingung für Fall 1:

$$\Delta T > \frac{v_{lim}^2}{a_{max}} \tag{4}$$

mit

$$\Delta T = T_1 - T_0$$

T_0 ... Starttemperatur
T_1 ... Zieltemperatur

2.1 Fall 1

Für den Fall mit konstanter Geschwindigkeitsphase setzen wir die benötigte Gesamtüberführungszeit aus drei Teilen zusammen. Dabei wird die Verzögerung gleich der Beschleunigung und somit $t_a = t_d$ gesetzt.

$$t_{end} = 2 \cdot t_a + t_c$$

Die Zeit bis zum Erreichen der Konstantgeschwindigkeit ist durch

$$t_a = \frac{v_{lim}}{a_{max}} \tag{5}$$

gegeben. Die Dauer der Phase mit konstanter Geschwindigkeit kann aus dem Ansatz $\Delta T = \frac{1}{2} v_{lim} \cdot t_a + v_{lim} \cdot t_c + \frac{1}{2} v_{lim} \cdot t_a$ hergeleitet werden:

$$t_c = \frac{\Delta T}{v_{lim}} - \frac{v_{lim}}{a_{max}}. \tag{6}$$

2.2 Fall 2

Im Fall wo keine Konstantgeschwindigkeit erreicht wird, berechnet sich die erreichte Maximalgeschwindigkeit zu

$$v_a = a_{max} \cdot t_a. \tag{7}$$

Die Temperaturdifferenz ergibt sich mit

$$\Delta T = v_a \cdot t_a$$

und durch einsetzen von (7) erhält man

$$\Delta T = a_{max} \cdot t_a^2.$$

Daraus kann die benötigte Überführungszeit berechnet werden:

$$t_a = \sqrt{\frac{\Delta T}{a_{max}}}$$

3 Generierung des diskreten Geschwindigkeitsverlaufs

Mit der Abtastzeit t_s und den Zeiten der einzelnen Phasen, t_a für Fall 2 und t_a, t_c für Fall 1, kann die Anzahl der Punkte pro Phase berechnet werden.

Fall 1:

$$n_a = round\left(\frac{t_a}{t_s}\right)$$
$$n_c = round\left(\frac{t_c}{t_s}\right)$$
$$n = 2 \cdot n_a + n_c$$

Fall 2:

$$n_a = round\left(\frac{t_a}{t_s}\right)$$
$$n = 2 \cdot n_a$$

Durch die Diskretisierung des Verlaufs ergibt sich zwangsweise eine Abweichung zum kontinuierlichen Verlauf. Dieser Fehler soll durch Anpassung von v_{lim} und a_{max} minimiert werden. Durch die Ganzzahlige Anzahl an Punkten pro Phase, ergeben sich minimal unterschiedliche Dauern der Phasen. Diese gerundeten Phasenzeiten werden im Folgenden durch eine Tilde $\tilde{t}$ gekennzeichnet.

Fall 1:

$$\tilde{t}_a = n_a t_s$$
$$\tilde{t}_c = n_c t_s$$
$$\tilde{t}_{end} = (n_a + n_c) t_s$$

Fall 2:

$$\tilde{t}_a = n_a t_s$$
$$\tilde{t}_{end} = 2 \cdot n_a t_s$$

Um eine exakte Überführungsdistanz ΔT zu garantieren, können nun die Parameter v_{lim} und a_{max} so angepasst werden, dass sich mit den neuen Werten $\tilde{t}_a, \tilde{t}_c, \tilde{v}_{lim}, \tilde{a}_{max}$ wieder exakt das vorgegebene ΔT ergibt.

Fall 1:

$$\tilde{a}_{max} = \frac{\Delta T}{\tilde{t}_a(\tilde{t}_a + \tilde{t}_c)}$$
$$\tilde{v}_{lim} = \tilde{a}_{max}\tilde{t}_a$$

Fall 2:

$$\tilde{a}_{max} = \frac{\Delta T}{\tilde{t}_a^2}$$
$$\tilde{v}_{lim} = \tilde{a}_{max}\tilde{t}_a$$

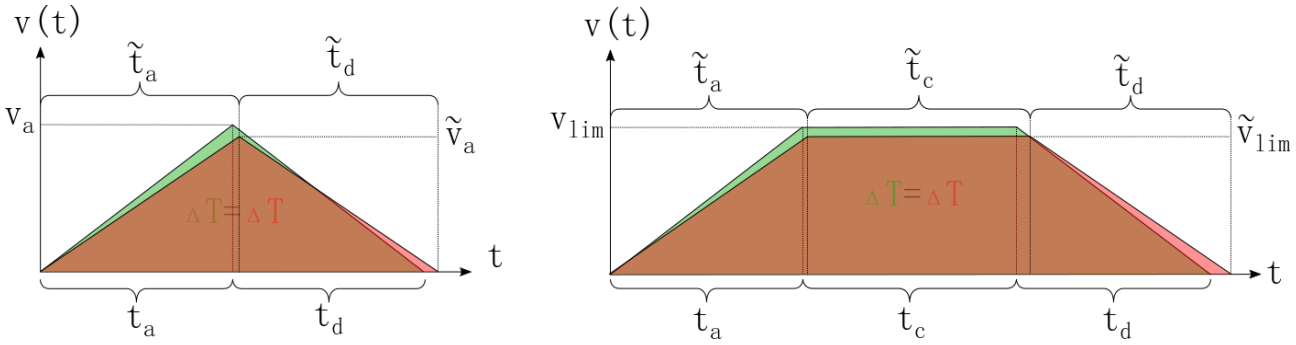

Figure 3: Visualisierung der Anpassung von v_{lim} und a_{max} auf $\tilde{v}_{lim}$ und $\tilde{a}_{max}$

Mit diesen angepassten Parametern $\tilde{v}_{lim}$ und $\tilde{a}_{max}$ kann nun der Geschwindigkeitsverlauf vorgegeben werden, mit welchem dann möglichst gut $T_1 = T_0 + \Delta T$ gilt. In Matlab oder Python kann dieser Geschwindigkeitsverlauf beispielsweise mit Hilfe der Funktion *linspace* bzw. *numpy.linspace* erzeugt werden. Durch einfache Integration, z.b. mittels der Trapezregel, erhält man dann die gewünschte S-curve zwischen T_0 und T_1.

$$T(k) = T(k-1) + t_s \frac{v(k) + v(k-1)}{2}$$

mit

$$T(0) = T_0.$$

4 Abschätzung des maximalen Fehlers zu v_{lim} und a_{max} (work in progress)

4.1 Fall 1

$$\Delta a_{max} = \Delta T \left(\frac{1}{t_a(t_a + t_c)} - \frac{1}{\tilde{t}_a(\tilde{t}_a + \tilde{t}_c)} \right)$$

Es ist ersichtlich, dass die Abweichung erst relevant wird, wenn t_a, t_c klein im Vergleich zum maximalen

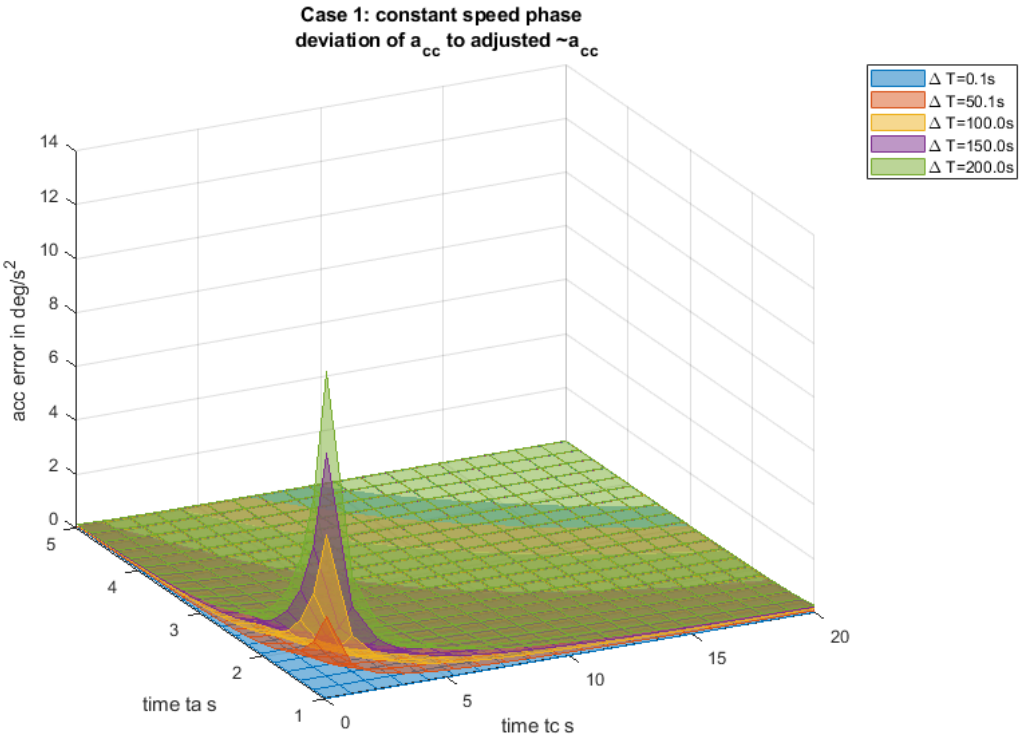

Figure 4: Abweichung der angepassten Beschleunigung $\tilde{a}_{max}$ zur Vorgegebenen a_{max} in Abhängigkeit der Beschleunigungs- und Konstantgeschwindigkeitszeit t_a, t_c und ΔT

Rundungsfehler $\Delta t_{a,max} = \Delta t_{c,max} = 0.4\dot{9}t_s$ werden. Dieser Fall tritt auf, wenn entweder die Beschleunigung a_{max} und die Geschwindigkeitsbeschränkung v_{lim} groß werden oder ΔT sehr klein.

4.2 Fall 2

$$\Delta a_{max} = \Delta T \left(\frac{1}{t_a^2} - \frac{1}{\tilde{t}_a^2} \right)$$

Für den Fall 2 gilt: Eine große Abweichung der angepassten Beschleunigung zur Vorgegebenen tritt auf, wenn das ΔT groß und t_a klein, also a_{max} groß, wird. Da keine großen Beschleunigungen vorkommen, ist auch dieser Fall zu vernachlässigen.

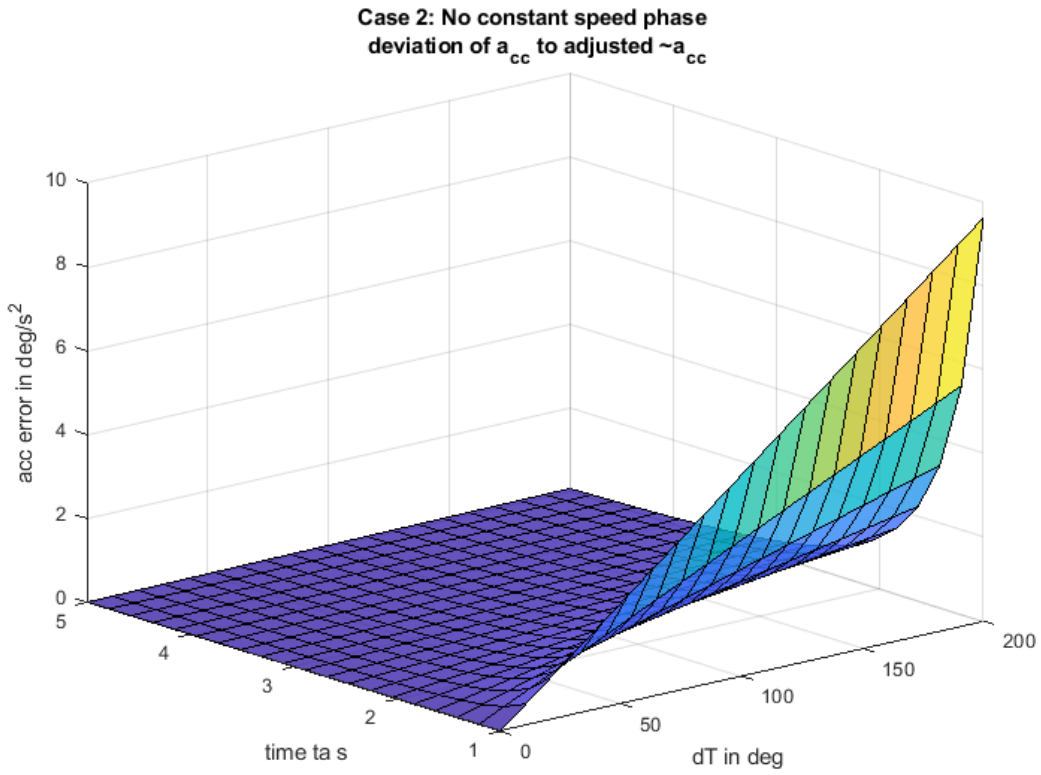

Figure 5: Abweichung der angepassten Beschleunigung $\tilde{a}_{max}$ zur Vorgegebenen a_{max} in Abhängigkeit der Beschleunigungszeit von t_a und ΔT

5 Code Implementierung

5.1 Matlab

```
function [t, y, dy, ddy, ta, tc] = fcn_s_curve(T0, T1, spd, acc, ts)
    dT = T1-T0;
    tc = [];

    if spd <= 0 || acc <= 0
       disp('error: negative speed, acc settings are not valid. Use positive values for all trajectories')
       return
    end

    if dT == 0
        disp('target-temp has to be different than the start temperature')
        return
    end

    if abs(dT) > spd^2/acc % constant speed trajectory

        % calculate phase times
        ta = spd/acc;
        tc = abs(dT)/spd - spd/acc;

        % calculate number of points per phase
        na = round(ta/ts);
        nc = round(tc/ts);
        n = 2*na+nc;

        % calculate rounded phase times
        tcr = nc*ts;
        tar = na*ts;
        tendr = n*ts;

        % adapt acc and speed at end of acc-phase to achieve dT exactly
        accr = abs(dT)/(tar*(tar+tcr));
```

```
        spdr = abs(dT)/(tar+tcr);

        % continuous acc, spd
        acc_cont = abs(dT)/(ta*(ta+tc));
        spd_cont = abs(dT)/(ta+tc);

        % create speed curve
        t = linspace(0,tendr,n);
        dy0 = linspace(0,spdr,na);
        dy1 = linspace(spdr,spdr,nc);
        dy2 = linspace(spdr,0,na);
        dy = [dy0,dy1,dy2];

    else % trajectory without constant speed

        % calculate phase time
        ta = sqrt(abs(dT)/acc);

        % calculate number of points per phase
        na = round(ta/ts);
        n = 2*na;

        % calculate rounded phase times
        tar = na*ts;  % time of acc after rounding
        tendr = 2*tar;

        % adapt acc and speed at end of acc-phase to achieve dT exactly
        accr = abs(dT)/tar^2;
        spdr = abs(dT)/tar;

        % continuous acc, spd
        acc_cont = abs(dT)/ta^2;
        spd_cont = abs(dT)/ta;

        % create speed curve
        t = linspace(0,tendr,n);
        dy0 = linspace(0,spdr,na);
        dy2 = linspace(spdr,0,na);
        dy = [dy0,dy2];
    end

    % change trajectory direction if start-temp is higher than target-temp
    dy = sign(dT)*dy;

    % check deviation from set values
    err_acc = abs( (acc_cont-accr)/acc_cont );
    err_spd = abs( (spd_cont-spdr)/spd_cont );
    fprintf('\n----------\nacc(max)=%.4f, acc(cont)=%.4f, acc(adjusted)=%.4f, err=%.2f%%, \nspd(max)=%.4f, spd(co

    if err_spd > 0.2 || err_acc >0.2
       disp('warning: trajectory deviates more than 20% from set values')
    end

    % init speed vector
    y = zeros(1,n);
    y(1) = T0;

    % init acc vector (optional)
    ddy = zeros(1,n);

    for l=2:n
        y(l) = y(l-1) + ts*(dy(l)+dy(l-1))/2;    % Integration by trapezoidal rule
        ddy(l) = (dy(l)-dy(l-1))/ts;             % Forward difference quotients
    end

end
```