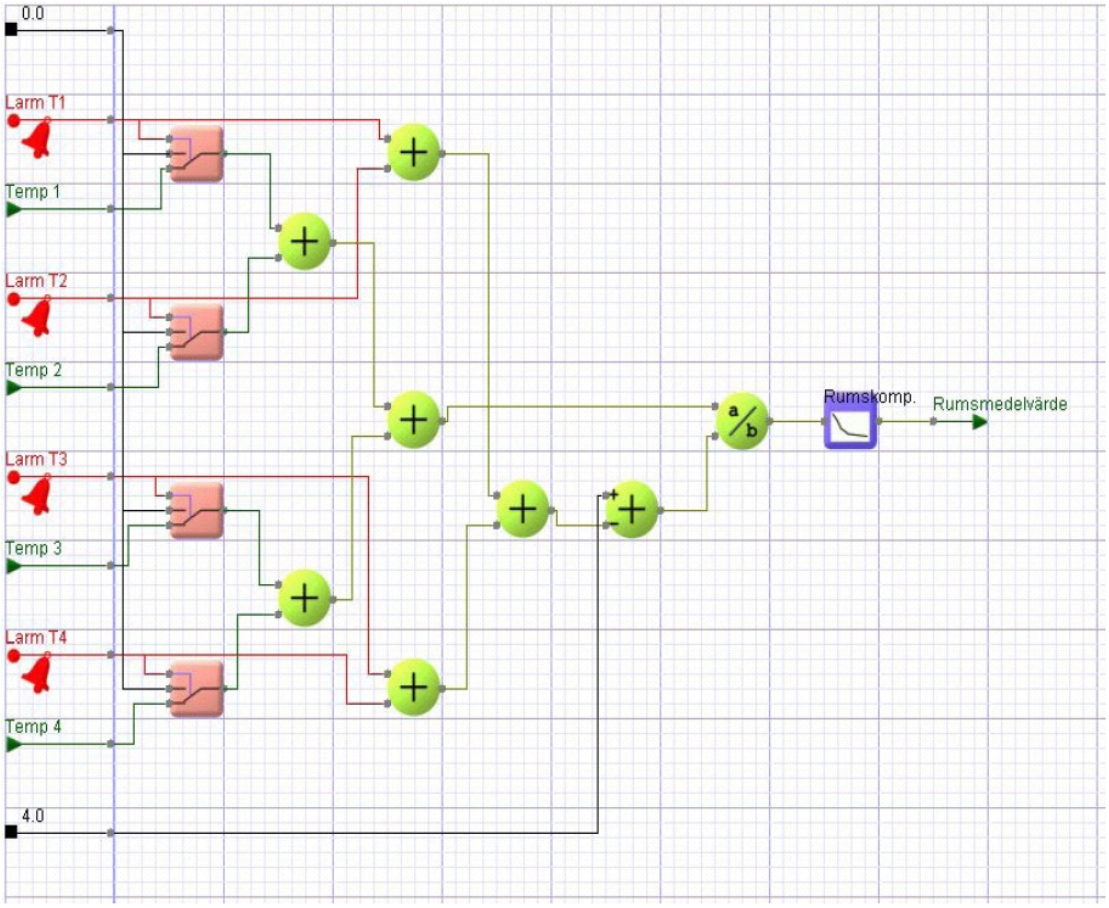
Abelko INNOVATION

# IMSE WebMaster Pro

# Applikationsexempel rumskompensering




| Document title | | |
|---|---|---|
| Applikationsexempel rumskompensering | | |
| Document Identity | | Date |
| 4655-010-01 | | 2005-04-22 |
| Valid for | Firmare version | Webpages version |
| IMSE WebMaster Pro | 1.07 | 1.07 |

# 1. Sammanfattning

För regulatorer där framledningstemperaruren styrs av utomhustemperaturen efter en reglerkurva kan man införa rumskompensering. Detta innebär att man har en eller flera givare som mäter inomhustemperatur, och justerar börvärdet för framledningstemperaturen efter dessa. Rumskompenseringen kan se till att man får ett bra inomhusklimat även om reglerkurvan är feljusterad, eller när väderförhållandena gör att reglerkurvan inte fungerar så bra. Kraftig uppvärmning av solinstrålning, eller nedkylning på grund av kraftig blåst är exempel på förhållanden som kan gör att en normalt välfungerande kurva inte fungerar så bra.

Detta applikationsexempel visar hur man kan göra rumskompensering med flera givare. Exempel finns både i grafisk programmering och i skript.

# 2. Exempel med grafisk programmering

## *2.1. Förutsättningar*

I detta exempel antar vi att vi har en regulator som är konfigurerad så att den har börvärdesförskjutning från en kanal. Denna kanal har namnet Rumsmedelvärde.

Vi ska använda fyra rumsgivare till rumskompenseringen. Vi ska beräkna medelvärdet för dessa givare som utgångspunkt för förskjutningen. Vi använder en kurva för att ange hur stor förskjutningen ska vara vid olika temperaturer.

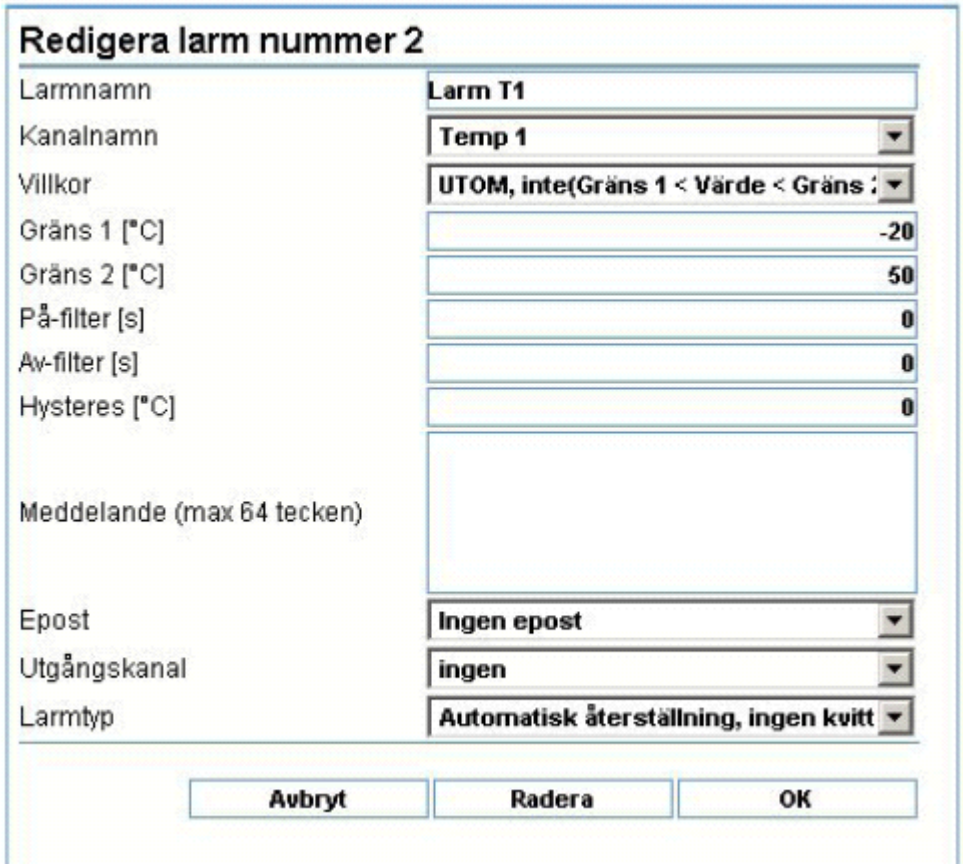
Ett problem är att givare både den och dess anslutning kan sluta fungera. En PT1000 givare med avbrott visar över 150 °C. Om det är kortslutning visar den ca –50 °C . Om ett sådant felaktigt värde skulle komma att ingå i medelvärdesbildningen så blir naturligtvis medelvärdet helt fel. Följaktligen blir även förskjutningen helt fel, vilket skulle leda till att det antingen blir väldigt varmt eller väldigt kallt inomhus. Vi ska därför se till att inte räkna med trasiga givare i medelvärdet.

## *2.2. Förskjutningskanalen Rumsmedelvärde*

Om man inte har gjort en kanal för börvärdesförskjutning redan tidigare så gör det nu. Döp helt enkelt om en ledig kanal i kanallistan till Rumsmedelvärde.


## *2.3. Givarlarm*

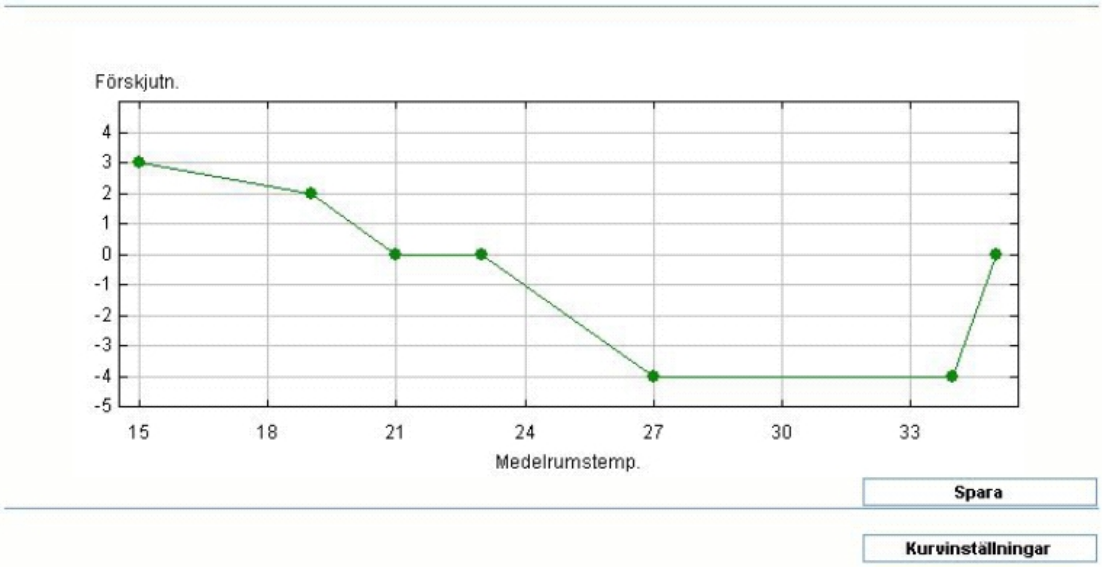
För vart och ett av de fyra rumsgivarna skapar vi larm. Välj ett ledigt larm i listan över larm i inställningar. Ge det ett namn, och välj en av rumsgivarna som kanal. Välj larmvillkoret UTOM. Gränserna sätter man antingen efter vad ingången kan mäta på den aktuella givartypen, eller efter vilka temperaturer man rimligen kan vänta sig att mäta inomhus. *Gräns 1* är den lägre gränsen och *Gräns 2* den övre.




Med larm för alla fyra givarna så kommer vi att få veta ifall någon av dem slutar fungera. Vi ska dessutom använda larmen för att se till att vi inte räknar med felaktiga givare i medelvärdet.

## *2.4. Förskjutningskurva*

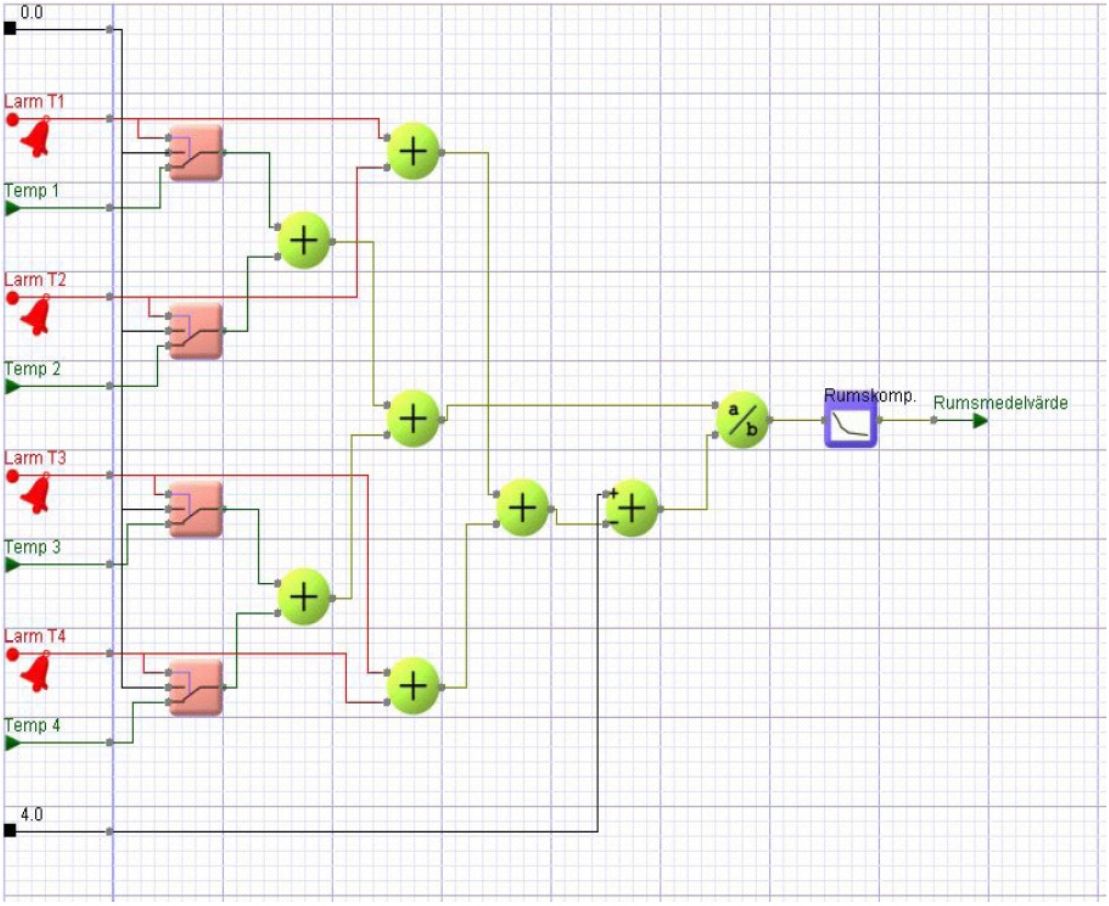
Steg tre är att skapa en förskjutningskurva. I denna bestämmer vi hur mycket framledningstemperaturens börvärde ska förskjutas vid olika inomhustemperaturer.




I kurvan ovan har vi ett dödområde mellan 21 och 23 grader där förskjutningen är noll. Är det kallare höjer vi börvärdet, och är det varmare sänker vi det. Det är viktigt att den sista punkten är noll, vilket förklaras senare.

## 2.5. Den grafiska programmeringen

Till sist så sätter vi ihop ett litet grafiskt program:




För varje givare har vi med givarens signal och dess larmvärde. Med en switch växlar vi in konstanten noll i stället för givarvärdet ifall larmet säger att givaren är sönder. Givarsignalerna ut från givarna summeras ihop till en signal. Tre andra summerare räknar ihop hur många larm, och därmed trasiga givare, som vi har. För att räkna ut hur många hela givare vi har tar vi fyra minus antalet larm.

Medelvärdet räknar vi ut som summan av alla givare delat med antalet givare. Detta skickas in i kurvan och sedan vidare ut i kompenseringskanalen.

Ifall alla givare skulle vara sönder så får vi division med noll. Skriptspråket hanterar detta genom att säga att resultatet inte är oändligt, men väl väldigt stort. Eftersom den sista punkten i kurvan är noll så kommer värdet ut från kurvan att vara noll. Går alla givare sönder så kör vi alltså utan rumskompensering.

## 2.6. Värt att notera

Rumskompenseringen är en regulator. Som den är implementerad i detta exempel så är det en P-regulator, där kurvans lutning bestämmer P-faktorn. I P-regulatorers natur ligger att de kan börja självsvänga ifall P-faktorn är för stor. Svängningarna kan vara väldigt lågfrekventa.

# 3. Exempel med skript

## *3.1. Förutsättningar*

Vi har, precis som i det första exemplet, fyra rumsgivare. Dom är placerade i olika rum, med olika karaktär. Därför vill vi vikta dom, så att större hänsyn kan tas till vissa givare, och mindre till andra.

Även i detta exempel används en kurva för att översätta en temperatur till en börvärdesförskjutning. Ett problem vi vill undvika i detta exempel är att börvärdet höjs så fort någon öppnar ett fönster i närheten av en givare. Därför ska alla temperaturer filtreras. Detta skulle man kunna göra i kanaler med den matematiska funktionen AR-filter, men eftersom vi jobbar med skript så behöver vi inte kanaler. Vi implementerar filterfunktionen i skriptet i stället.

Även här vill vi skydda oss mot trasiga givare, men vi använder inte larm för att göra detta, även om det hade gått lika bra. Vi använder i stället en gemensam inställning för alla givare av vad som ska betraktas som vettiga signaler.

## *3.2. Skriptet*

I aliassektionen kopplar vi namn till olika resurser som används. T1 till t4 är de fyra tempgivarna. W_t1 till w_t4 är viktfaktorerna för givarna. Ok_max_val och ok_min_val är parametrarna för högsta och lägsta värde för en korrekt givare. Ligger värdet utanför dessa så betraktas givaren som felaktig. FilterCoef är filterfaktorn, ett tal mellan noll och ett. Weigthet_temp är den viktade och filtrerade medeltemperaturen, och comp_temp är börvärdesförskjutningen. Comp_curve är kurvan för börvärdesförskjutning. Dessa kanaler, parametrar och kurvan skapar man lämpligen innan man skriver skriptet. Om man kopierar skriptet så byter man bara ut aliasdefinitionerna så att de pekar på rätt kanaler och parametrar.

Vidare finns det ett antal variabler. För varje givare finns två variabler som har med filtreringen att göra. För varje vikt finns det en temporärvariabel. Till sist så finns det en variabel med namnet first_run. Alla variabler har värdet noll första gången skriptet körs efter en omstart. Med first_run utnyttjar vi detta för att hålla reda på om det är första gången, så att vi kan göra en del initiering.

```
ROUTINE TEMP_WEIGHT
ALIAS
  t1 = CHANNEL[1];
  t2 = CHANNEL[2];
  t3 = CHANNEL[3];
  t4 = CHANNEL[4];
  w_t1 = PARAMETER[1];
  w_t2 = PARAMETER[2];
  w_t3 = PARAMETER[3];
  w_t4 = PARAMETER[4];
  ok_max_val = PARAMETER[5];
  ok_min_val = PARAMETER[6];
  filterCoef = PARAMETER[7];
  weighted_temp = CHANNEL[60];
  comp_temp = CHANNEL[61];
  comp_curve = CURVE[1];
VAR
  t1_filt; t1_filt_prev;
  t2_filt; t2_filt_prev;
  t3_filt; t3_filt_prev;
  t4_filt; t4_filt_prev;
  w_t1_tmp; w_t2_tmp; w_t3_tmp; w_t4_tmp;
  first_run;
```

```
BEGIN
  % kickstart the filter on first run
  IF firstRun = 0 THEN
    t1_filt := t1;
    t2_filt := t2;
    t3_filt := t3;
    t4_filt := t4;
  ELSE
  % filer the signals
    t1_filt := filter_coef * t1_filt_prev + (1-filter_coef) * t1;
    t2_filt := filter_coef * t2_filt_prev + (1-filter_coef) * t1;
    t3_filt := filter_coef * t3_filt_prev + (1-filter_coef) * t3;
    t4_filt := filter_coef * t4_filt_prev + (1-filter_coef) * t4;
  ENDIF;
  t1_filt_prev := t1_filt;
  t2_filt_prev := t2_filt;
  t3_filt_prev := t3_filt;
  t4_filt_prev := t4_filt;
  % Filter end

  % check values min, max limit
  IF (t1 > ok_max_val) OR (t1 < ok_min_val) THEN
    w_t1_tmp := 0;
  ELSE
    w_t1_tmp := w_t1;
  ENDIF;
  IF (t2 > ok_max_val) OR (t2 < ok_min_val) THEN
    w_t2_tmp := 0;
  ELSE
    w_t2_tmp := w_t2;
  ENDIF;
  IF (t3 > ok_max_val) OR (t3 < ok_min_val) THEN
    w_t3_tmp := 0;
  ELSE
    w_t1_tmp := w_t1;
  ENDIF;
  IF (t4 > ok_max_val) OR (t4 < ok_min_val) THEN
    w_t4_tmp := 0;
  ELSE
    w_t4_tmp := w_t4;
  ENDIF;

  % make sure the wights calculation do not result in a NaN
  IF (w_t1_tmp + w_t2_tmp + w_t3_tmp + w_t4_tmp) <> 0 THEN
    weighted_temp <- (t1_filt*w_t1_tmp + t2_filt*w_t2_tmp +
    t3_filt*w_t3_tmp + t4_filt*w_t4_tmp)/
    (w_t1 + w_t2 + w_t3 + w_t4);
    comp_temp <- comp_curve(weighted_temp);
  ELSE
    weighted_temp <- 0;
    comp_temp <- 0;
  ENDIF;
  firstRun := 1;
END;
```

Första delen av skriptet hanterar filtreringen av givarna. Funktionen är ett ett autoregresivt filter, precis samma som den matematiska funktionen AR-filter för kanaler. Variablerna tx_filt innehåller efter det ett filtrerat temperaturvärde.

Nästa del av skriptet kollar om givaren är hel. Är den det så tilldelas givarens temporära viktvariabel samma värde som motsvarande parameter, men är givaren trasig så får den värdet noll.

Sist så görs den viktade medelvärdesberäkningen. Om alla givare är trasiga så sätts både medelvärdet och förskjutningen till noll. Annars beräknas medelvärdet som summan av alla givare multiplicerad med sin vikt, delad med summan av alla vikter. Detta sätt att räkna innebär att det inte spelar någon roll hur man anger viktningen, det är bara deras inbördes storlek som spelar roll.