Temperaturkontrol med PID og LabView

Nanoteket

April 2009

Beskrivelse

Formålet med denne øvelse er at opnå kendskab til et meget anvendt kontrolsystem til at holde en given variabel konstant i et system, hvor variablen er påvirket af omskiftelige omgivelser, nemlig PID- regulering. Denne form for regulering kan benyttes inden for mange forskellige områder, såsom regulering af temperaturen i et rum eller en bygning. Vi ønsker i denne øvelse at regulere temperaturen i et mindre system, nemlig en ganske almindelig elektrisk pære, der er omsluttet af en kop. I dette system kan vi variere spændingsfaldet over pæren og derved styre dennes opvarmning. Temperaturen af pæren måles med et termoelement, der udnytter Seebeck effekten til at generere en spændingsforskel , der er proportional med temperaturen af elementet (i forhold til en referencetemperatur).

Øvelsen fungerer også som en introduktion til programmering i LabView, da PID-reguleringen laves i dette program. LabView er et grafisk programmeringssprog, som gør det muligt med en PC at foretage dataopsamling og styre forskelligt apparatur. Programmerne du laver bliver gemt som *vi*-filer, som du gerne må gemme i kursusfolderen på skrivebordet.

> **Opgave 1**: Hvis følsomheden af termoelementetet er 41 μV/K med en forstærkning på 100 og LabView har en 12 bit opløsning på sin 0-10 V indgang, hvad er så den mindste ændring i temperaturen vi kan forvente at måle?

Praktiske informationer

Hvor og hvornår: Øvelsen foregår i rum 015, bygning 307. Vi mødes kl. 9 og slutter omkring kl. 16. Det er en rigtig god idé at have læst denne øvelsesvejledning i forvejen!

Flere opstillinger: Vi har til denne øvelse tre identiske opstillinger til rådighed, så I kan fordele jer på disse. Dette kan anbefales, selv om I ikke er så gode til LabView, da I lærer mest ved at gøre tingene selv.

Programmer: I skal som sagt bruge LabView, hvilket I bestemt ikke skal bekymre jer over - det er meget nemt at bruge. For at gøre det nemmere for alle at holde styr på deres programmer bedes I gemme disse i kursusfolderen på skrivebordet. Det er også en god ide at lave sikkerhedskopier af jeres programmer, når I laver meget om på dem. Skulle det nye så ikke virke efter hensigten, så har I altid en kopi der virker.

Journal: Udgøres af svar på de stillede spørgsmål i øvelsen og en kort beskrivelse/dokumentation af forløbet. I skal kun aflevere en journal per hold. Gør det kort - den bør ikke fylde mere end seks sider.

Miniprojekt: Der vil være mulighed for at lave miniprojekt om PID-regulering eller anden styring i LabView. Det bedste er, hvis I selv har nogle idéer, så må vi se på om det kan lade sig gøre.

Tænd/sluk regulering

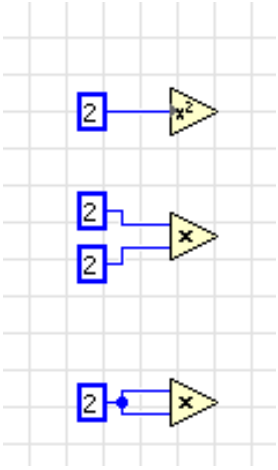
Lad os starte med at betragte en simpel tænd/sluk regulering, hvor den regulerbare ydelse, P, enten kan tændes ved maksimal ydelse, $P = P_{max}$, eller slukkes, $P = 0$. Reguleringen af ydelsen bruges så til at regulere en given variabel, U, i forhold til et forudbestemt sætpunkt, U_{set}. Ydelsen kunne f.eks. være varmen der kommer fra radiatoren i vores stue og variablen er så temperaturen i stuen, som vi så gerne vil regulere. Kommer vi f.eks. hjem fra ferie, så har sætpunktet formentligt været sat til 10 °C og det første vi gør er at sætte dette op til 20 °C. Temperaturen (10 °C) er nu under sætpunktet og radiatoren vil derfor tænde for varmen for at hæve temperaturen til den ønskede. Temperaturen vil nu stige til over 20 °C, hvorefter radiatoren vil slukke for varmen. Temperaturen vil så falde til under 20 °C, hvorefter radiatoren igen vil tænde for varmen og cyklusen kan forsætte.

> **Opgave 2**: Hvorfor vil variablen, i dette tilfælde temperaturen, komme til at svinge omkring sætpunktet som beskrevet ovenfor ved en tænd/sluk regulering?

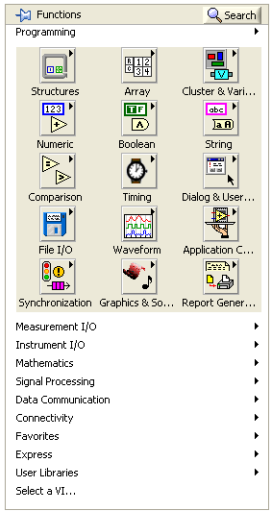
> **Øvelse 1**: Lav en tænd/sluk temperaturkontrol til pæren i LabView. Du kan frit lade dig inspirere af nedenstående vejledning.

Hvordan laves det i LabView?

LabView er som tidligere nævnt et grafisk programmeringssprog. Det betyder, man bygger sit program ved hjælp af en række delelementer, som man forbinder via ledninger. Vil vi for eksempel sætte et tal i anden, trækker vi tallet to gange hen til gange-funktion[1], eller alternativt finder vi en funktion, som i sig selv har evnen til at kvadrere et tal, som det er gjort i figur 1.



Figur 1: *Tre forskellige måder at finde værdien af* 2^2.



Figur 2: *Standardfunktionerne i LabView. Du kan vælge dine egne funktioner nederst i menuen.*

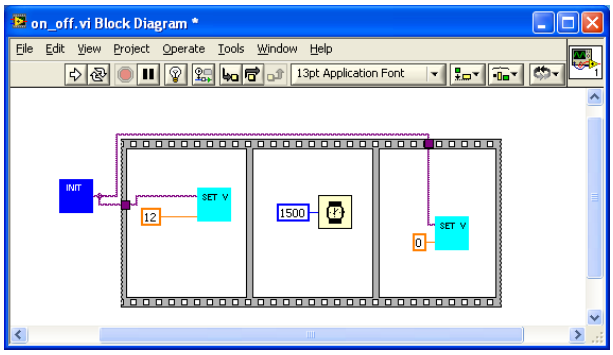
I LabView kaldes det vindue, hvor man laver selve programmet, for blokdiagrammet og det vindue, hvor man bruger programmet for frontpanelet. Du kan skifte mellem de to paneler med

[1]Slå eventuelt hjælpefunktion til med Ctrl+h for på den måde altid at få vist hvordan terminalerne skal forbindes til de forskellige enheder på blokdiagrammet.

tastaturet ved at trykke Ctrl+E. Det samlede program kaldes for et Virtuelt Instrument, eller blot en *vi*. Denne terminologi skyldes, at man ofte vil bygge LabView programmer op, så frontpanelet kommer til at minde om et rigtigt fysisk apparat med knapper, drejeskiver, viserinstrumenter, oscilloskopgrafer mv. - altså et virtuelt instrument.

1. Tænd og sluk for pæren

For at lave en temperaturregulering får vi brug for en funktion, som kan styre vores strømforsyning[2]. Denne funktion finder vi ved at højreklikke på blokdiagrammet og vælge "*Select a VI*". Vi skal i alt bruge to funktioner, en til at oprette kommunikationen med strømforsyningen og en til at sætte spændingen, henholdsvis "*Init-ISO-TECH.vi*" og "*Set-V-ISO-TECH.vi*" (De findes i kursusfolderen på skrivebordet). Træk en ledning fra Init til Set-V og forbind værdien 12 til input på Set-V; dette gør vi lettest ved at højreklikke på den tilhørende terminal og vælge "*Create*" - "*Constant*". For at undgå, at pæren bliver alt for varm, skal vi sørge for at slukke for den igen inden for en bestemt tid. Dette klarer vi ved at lave en såkaldt sekvens, som vi finder i menuen med komponenter under punktet "*Structures*" (vælg en "*Flat Sequence*", da denne er nemmest at overskue). Når vi har oprettet en sekvens, skal vi sørge for, at den har i alt tre sektioner. Der oprettes kun en fra starten, så vi skal altså lave to mere ved at højreklikke på kanten af strukturen og vælge "*Add Frame After*" eller "*Add Frame Before*" så det passer. Nu er vi klar til at oprette vores første fungerende program: Placer Init til venstre for sekvensen og placer en Set-V i både første og sidste vindue. I det første vindue forbinder vi værdien 12 og i det sidste vindue værdien 0. Da vores strømforsyning kun reagerer langsomt på kommandoer, skal der gå en vis tid fra den første kommando til den sidste, det klares ved at bruge en "*Wait*"-komponent (du finder den under menuen "*Timing*"), der placeres i det midterste felt og forbindes til værdien 1500. Det eneste, vi nu mangler, er at forbinde Init til begge de to Set-V komponenter og vi skulle nu stå med et skærmbillede svarende til figur 3. Tryk på den lille hvide pil i toppen af skærmen for at starte programmet. Vi skulle nu gerne se, at pæren tænder i 1,5 s og herefter slukker igen.



Figur 3: *Den første spæde start på en tænd/sluk-kontrol.*

[2]Da LabView ikke har en indbygget funktion til at styre lige præcis vores strømforsyning, har vi lavet en selv.

2. Måling af temperaturen

Næste opgave bliver at holde øje med temperaturen. For at gøre dette, skal vi have fat i en funktion til at kommunikere med eksternt måleudstyr (i vores tilfælde det termoelement som er monteret på pæren). Disse funktioner er placeret i menuen under punktet "*Measurement IO*", hvor vi skal finde "*DAQmx Start*" og placere den uden for sekvensen. For at fortælle LabView, hvilken kanal vi gerne vil måle på, skal vi højreklikke på terminalen "*Channel*" og vælge "*Create*" - "*Constant*" og her vælge det punkt, der hedder "*Temperature*". For at læse måleværdierne ind i programmet, skal vi have fat i funktionen "*DAQmx Read*" og forbinde "*Channel*"-terminalen med udgangsterminalen på start-funktionen. Nu mangler vi at fortælle programmet, at vi gerne vil måle temperaturen kontinuert og ikke kun en enkelt gang. Dette klarer vi ved at finde et "*While*"-loop under "*Structures*" og trække det rundt langs indersiden af det første panel i sekvensen. Et while-loop afvikler alle de funktioner, som står inden i det, i en evig løkke, indtil det bliver afbrudt, og vi skal derfor sørge for at lave os en sådan afbrydningsmekanisme. Dette klares ved at højreklikke på det røde stopskilt og vælge "*Create*" - "*Control*", hvorved der oprettes en stopknap på frontpanelet.

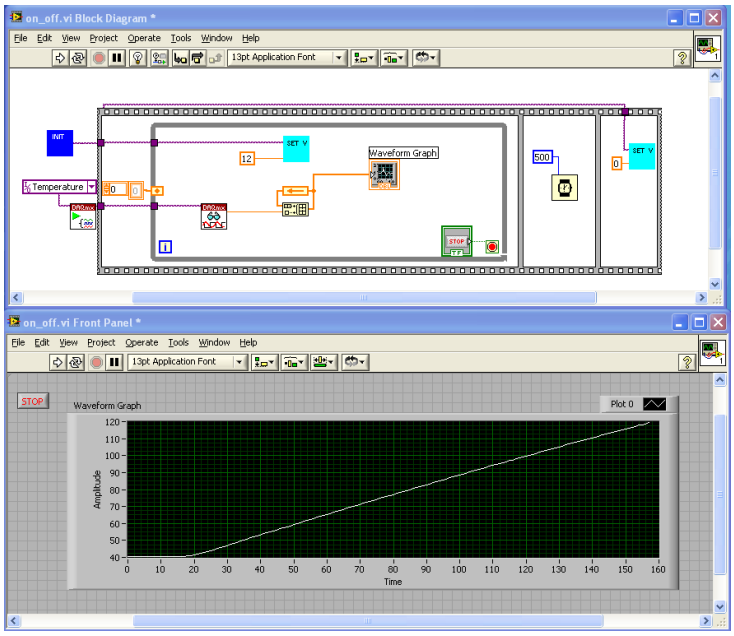
3. Visning af temperaturen

Næste opgave består i løbende at opsamle måleresultaterne og plotte dem på en graf. For at gøre dette skal vi oprette en datastruktur som kan gemme en række talværdier. Sådan en struktur kaldes et array. LabView har en lang række funktioner til at håndtere arrays, men vi har kun brug for en enkelt, nemlig "*Build Array*" der ligger under punktet "*Array*". Denne funktion tager et eksisterende array og tilføjer et ekstra element; tag fat i bunden af "*Build Array*"-kassen med musen og træk lidt ned, så der i alt kommer to terminaler på venstre side. Den nederste terminal skal nu forbindes med det element, som skal tilføjes til array'et; træk derfor en ledning fra "*data*" terminalen af "*DAQmx Read*"-funktionen til denne terminal. Den øverste terminal skal have et eksisterende array som input, hvilket let klares ved at trække en ledning fra udgangsterminalen på "*Build Array*" tilbage til øverste input. Der kommer nu en pil frem som repræsenterer en såkaldt "*Feedback node*", altså en metode til at gemme data fra den ene iteration til den næste i et loop. Mekanismen er altså nu, at vi for hvert gennemløb gemmer vores nye værdi plus alle tidligere værdier. Men hvordan kommer vi så i gang? Første gang loopet køres, er der jo ikke nogen tidligere værdier. Dette klares via den såkaldte initaliseringsterminal, som er kommet frem til venstre i while-loopet. Højreklik på denne terminal og vælg "*Create*" - "*Constant*", hvorved der oprettes et tomt array, som anvendes til at få loopet i gang. Ovenstående kan også klares ved at oprette et "*Shift register*", hvilket gøres ved at højreklikke på enten venstre eller højre side af while-loopet og vælge "*Add Shift register*". Nu kommer der et ikon med en pil i begge sider af while-loopet, hvor den venstre returnerer værdien fra forrige loop eller startværdien og den højre sender den aktuelle værdi videre til næste gennemløb. Begge metoder er ækvivalente, så det er smag og behag der afgører, hvilken der foretrækkes.

Nu mangler vi en graf til at vise temperaturen. Gå til frontpanelet, højreklik på baggrunden og find en "*Waveform Graph*". Tilbage på blokdiagrammet kan vi nu forbinde udgangsterminalen fra "*Build array*" til grafen som LabView har lagt et tilfældigt sted på blokdiagrammet. Du er nu klar til at afprøve dit program. Om alt går vel, tænder pæren og vi ser en graf som viser en jævnt voksende temperatur (jvf. figur 4). Stands programmet på stopknappen, hvorved pæren gerne skulle slukke.

4. Styring via sætpunkt

Nu kan vores program både tænde og slukke for pæren samt registrere temperaturudviklingen, men det kan stadig ikke styre temperaturen mod et bestemt sætpunkt. For at kunne dette skal vi først lave et felt på frontpanelet, hvor sætpunktet kan indtastes. Dette klares på samme måde som grafen, denne gang skal vi bare finde en "*Numeric Control*". Tilbage i blokdiagrammet skal



Figur 4: *Nu er vores tænd/sluk kontrol næsten færdig.*

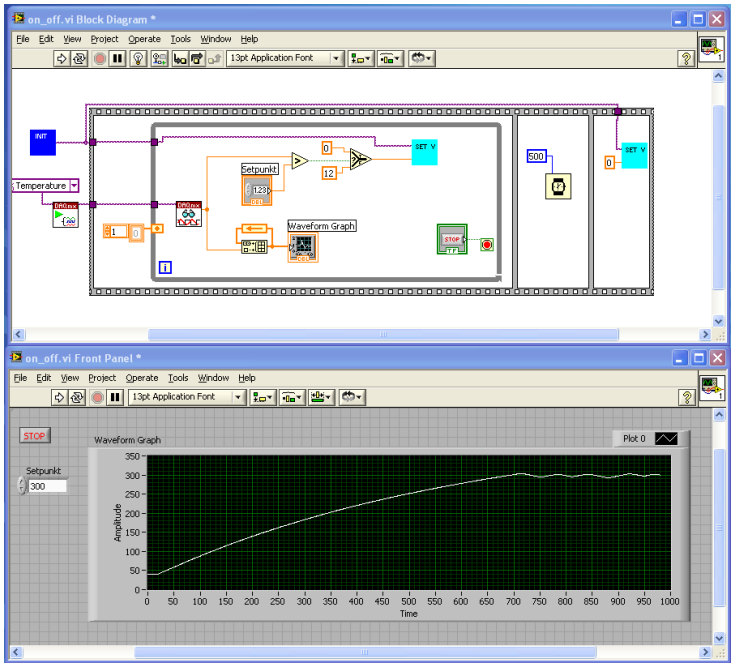
vi nu finde en sammenligningsfunktion, som kan afgøre, om den nuværende temperatur er større eller mindre end sætpunktet, sådan en funktion finder vi under "*Comparison*". Træk nu output fra henholdsvis kontrollen for sætpunktet og temperaturmålingen til sammenligningen. For at gøre kontrollen færdig mangler vi nu kun at sætte inputværdien til strømforsyningen afhængigt af, om resultatet af sammenligningen er sandt eller falsk. Funktionen til at gøre dette findes også under "*Comparisons*" og hedder "*Select*". En select skal have en sand/falsk værdi ind til den midterste terminal og to forskellige værdier til de to øvrige terminaler. Værdien af udgangen vil så være den ene eller den anden værdi afhængig af, om indgangen er sand eller falsk. Forbind værdierne 0 og 12 til indgangen og forbind udgangen til Set-V. Nu er din tænd/sluk-styring færdig og skal gerne se ud omtrent som figur 5.

PID-regulering

Vi så i forrige afsnit, at en tænd/sluk regulering vil resultere i en svingende variabel, som funktion af tiden. Dette er måske tilfredsstillende med temperaturen i ens stue, men i mange henseender vil disse udsving være alt for store. I disse tilfælde kan vi benytte en PID regulering, der er en algoritme bestående af tre funktioner med meget forskellige virkemåder: Proportional-, Integral- og Differential-regulering. Vi vil i det følgende gennemgå disse virkemåder i detaljer, så I kan få en fornemmelse for deres betydning i reguleringen.

Proportional-regulering

Lad os tage udgangspunkt i en ganske almindelig fartpilot i en bil, der prøver at holde en konstant fart. I dette tilfælde vil ydelsen være motorkraften og variablen være farten. Prøvede vi at lave en fartpilot med tænd/sluk reguleringen fra tidligere, ville ingen køre i bilen, da bilen i cyklus hele tiden ville accelerere (fuld motorkraft) og efterfølgende bremse (ingen motorkraft). Det er dog ikke svært at forbedre dette, hvis vi bare justerer motorkraften gennem dens fulde ydelsesområde, altså kan variere P kontinuert mellem 0 og P_{max}. Inden vi implementerer dette, indfører vi dog



Figur 5: *Den færdige tænd/sluk regulering i LabView.*

først et offset på ydelsen, P_{set}, hvilket er den ydelse der skal bruges for at holde variablen på sætpunktet, U_{set}. I eksemplet med fartpiloten svarer dette til, at fartpiloten når en ønsket fart, aflæser den aktuelle motorkraft og holder denne konstant. Ofte er vi interesseret i at variere sætpunktet som funktion af tiden, dvs. offsettet også vil variere med tiden. Sammenhængen mellem disse kunne benyttes til at bestemme offsettet som funktion af sætpunktet, men i praksis er dette besværligt og ikke nødvendigt. Derfor sættes offsettet normalt bare til middelydelsen, P_{mid}, hvilket vi også vil gøre her. Kigger vi nu tilbage på at variere motorkraften gennem hele ydelsesområdet, så gør vi nu følgende: Skulle farten (variablen) nu falde sættes fartpiloten til at øge motorkraften (ydelsen) proportionalt med faldet eller omvendt hvis den skulle stige. Rent matematisk kan vi skrive

$$P(t) = P_{mid} + G \cdot E(t), \tag{1}$$

hvor $E(t) = U_{set} - U(t)$ er forskellen mellem sætpunktet og den aktuelle værdi og G er forstærkningen. Sidstnævnte bestemmer, hvordan reguleringen virker, og denne parameter kan justeres efter ønske.

Opgave 3: Hvor stort er det temperaturinterval (variabelinterval) hvori spændingen (ydelsen) over pæren varieres mellem 0 og 12 V, når G sættes til 2?

Opgave 4: Hvor er intervallet fra forrige opgave placeret i forhold til sætpunktet og varierer placeringen med valget af offsettet på spændingen? Hvilken betydning har dette for vores P-regulering?

Øvelse 2: Overfør ovenstående principper til systemet med pæren og lav en P-regulering i LabView. *Hint: Vælg et højt setpunkt for temperaturen, f.eks. 200 °C.*

Øvelse 3: Find forstærkningen G, så temperaturen kommer tættest på sætpunktet uden at den begynder at svinge omkring dette (± 2 °C).

Øvelse 4: Beskriv hvordan P-reguleringen virker for forskellige værdier af G i forhold til resultaterne i opgave 3 og øvelse 3. Prøv for lave værdier og høje værdier af G (f.eks. $\frac{1}{2}$ og 20). *Hint: For hvilke værdier af G ligner P-reguleringen en tænd/sluk regulering?*

Integral-regulering

Du så formentligt i øvelse 3, at P-reguleringen ikke indstiller temperaturen ved sætpunktet (±0.5 °C). Samme fænomen ses i situationen, hvor bilen begynder at køre op ad bakke. Farten vil nu begynde at falde, men den øgede motorkraft er muligvis ikke tilstrækkelig til at hæve farten nok, da bilen kører op ad bakke! Til at afhjælpe denne situation ønsker vi en funktion der øger motorkraften gradvist, så længe farten er under setpunktet. Denne funktion kan repræsenteres vha. integralet over fejlen, da dette vil stige hvis fejlen ikke mindskes numerisk. Tilføjes dette til ligning (1) får vi

$$P(t) = P_{mid} + G\left(E(t) + \frac{1}{T_i}\int_0^t E(t')dt'\right), \tag{2}$$

hvor T_i er integral tidskonstanten, som i lighed med forstærkningen kan justeres efter ønske. Integral-reguleringen er formentligt det vigtigste led for at holde variablen ved sætpunktet - den ændrer ydelsen langsomt men præcist.

Opgave 5: Beregn ud fra værdierne i tabel 1 ydelsen fra en PI-regulering (2) for forskellige værdier af T_i. Hvad er betydningen af T_i? *Hint: Kig på forholdet mellem proportional- og integral-delen uden offsettet.*

Tid (s)	0.0	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0	2.2	2.4	2.6	2.8	3.0
E (°C)	0.0	0.3	0.6	0.9	1.2	1.5	1.8	2.1	2.4	2.7	3.0	3.3	3.6	3.9	4.2	4.5

Tabel 1: *Fiktive værdier for udviklingen af fejlen som følge af en pertubation til tiden lig* 0.

Opgave 6: Hvordan kan integral-reguleringen nemt implementeres i LabView og hvordan styres den tidslige del af integralet? *Hint: Er det muligt at repræsentere integralet som en sum?*

Øvelse 5: Lav en PI-regulering i LabView baseret på svaret i forrige opgave. Plot og kommenter på I-reguleringens virkemåde samt kommenter på forbedringerne i forhold til P-reguleringen.

Du har sikkert set, at temperaturen skyder godt over sætpunktet ved start. Dette er helt normalt og skyldes hukommelsen som integral-reguleringen besidder - værdien af integralet stiger så længe temperaturen er under sætpunktet og begynder først at falde når temperaturen kommer over sætpunktet. Det er desværre ikke muligt at undgå dette ved at stille på de normale parametre (G og T_i), uden at den normale drift forringes. Vi vil senere kigge på metoder til at undgå dette, så for nu må vi bare acceptere dette.

Differential-regulering

Ovenstående ser jo meget godt ud, men hvad sker der når bilen kører over toppen af bakken? Farten vil begynde at stige yderligere og den vil formentligt også stige hurtigere end vores hidtidige fartpilot kan sænke motorkraften tilstrækkeligt til at holde stigningen nede - bilen kører jo ned ad bakke. For at undgå dette har vi brug for en funktion, der ændrer motorkraften alt

afhængig af hastigheden hvormed fejlen ændrer sig. Dette kan vi repræsentere vha. differentialkvotienten af fejlen, da dette netop er hastigheden hvormed fejlen ændrer sig! Tilføjes dette til ligning (2) får vi

$$P(t) = P_{mid} + G\left(E(t) + \frac{1}{T_i}\int E(t)dt + T_d\frac{d}{dt}E(t)\right), \tag{3}$$

hvor T_d er differential tidskonstanten, som også i lighed med forstærkningen kan justeres efter ønske. Det skal her understreges at differential-reguleringen ikke skal benyttes til at undgå overskydningen ved start, da dette vil forringe hele algoritmens effekt kraftigt under normal drift. Den skal derimod bruges som et effektivt middel mod uventede pludselige ændringer i omgivelserne.

Opgave 7: Beregn ud fra værdierne i tabel 1 ydelsen fra en PID-regulering (3) for forskellige værdier T_d. Hvad er betydningen af T_d? *Hint: Kig på forholdet mellem proportional- og differential-delen uden offsettet og integral-delen.*

Opgave 8: Hvordan kan differential-reguleringen nemt implementeres i LabView? *Hint: Betragt det som hældningen mellem to på hinanden følgende fejlværdier.*

Øvelse 6: Lav en PID-regulering i LabView baseret på svaret i forrige opgave og programmet fra øvelse 5. Plot og kommenter på D-reguleringens virkemåde samt kommenter på forbedringerne i forhold til PI-reguleringen. *Hint: Kig på perturbationer i systemet, prøv f.eks. at puste til pæren.*

Tuning

I det forrige afsnit kiggede vi på PID reguleringens forskellige virkemåder og indførte parametrene G, T_i og T_d til at justere disse. Til trods for denne gennemgang, må vi dog stadig stille os selv spørgsmålet: Hvad skal vi sætte disse parametre til for at optimere vores PID-regulering til et givet system? I har sikkert fundet nogle værdier gennem arbejdet med reguleringen, men disse værdier er ikke globale, da de er stærkt afhængige af systemet. Vi har derfor brug for en mere generel og systematisk måde til at bestemme parametrene på. Til dette formål fandt J.G. Ziegler og N.B. Nicholls i 1942 en metode der hedder "*Closed-Loop Cycling method*". Metoden er ret simpel og går ud på følgende:

1. Slå integral- og differential-reguleringen fra.
2. Hæv forstærkningen, G, så temperaturen begynder at svinge kraftigt (± 10 °C) og find perioden, T_u, af svingningen.
3. Sænk forstærkningen til punktet, hvor variablen svinger en anelse (± 2 °C). Dette kaldes den ultimative forstærkning, G_u.
4. De optimerede værdierne af parameterne er nu givet som

$$G = 0.6 \cdot G_u, \quad T_i = 0.5 \cdot T_u \quad \text{og} \quad T_d = 0.12 \cdot T_u.$$

Dette giver desværre ikke altid de bedst mulige værdier for parametrene, så det er en god ide at finjustere dem efterfølgende.

Øvelse 7: Find de optimale værdier for dit system vha. den ovenfor beskevne metode. *Husk: Resultatet vil variere mellem opstillingerne.*

Øvelse 8: Undersøg PID-reguleringens reaktion på ændringer i omgivelser og hvor godt den tilpasser sig disse. Plot og kommenter de relevante resultater. *Hint: Prøv med pludselige korte ændringer i omgivelserne (pust til pæren) eller længerevarende ændringer (tænd en blæser).*

Ved at lade sætpunktet følge en sinuskurve er det muligt at undersøge reguleringens evne til at følge ændringer i omgivelserne nærmere.

Øvelse 9: Find den højeste frekvens af ændringer, som PID-reguleringen kan følge med rimelighed? Hvad er de begrænsende faktorer?

Forbedringer af overskydning

Vi snakkede tidligere om reguleringens tydelige svaghed - den får temperaturen til at skyde godt over sætpunktet ved start. En metode hvorpå dette kan undgås, er ved at ændre T_i, så størrelsen af integralleddet begrænses.

Opgave 9: Hvad skal man huske at undersøge, hvis metoden med begrænsning af integralleddet skal benyttes? *Hint: Tænk på integral-funktionens virkemåde og formål.*

En anden metode er ved at lade sætpunktet rampe op fra den aktuelle temperatur til det ønskede sætpunkt med f.eks. 10 °C per sekund. På denne måde vil fejlen hele tiden holdes nede og dermed også integralleddet.

Øvelse 10: Prøv at implementere en eller begge af ovenstående metoder og vurder deres effektivitet. Plot og kommenter de relevante resultater.

(Sidst revideret af Simon Brodersen, April 2009)