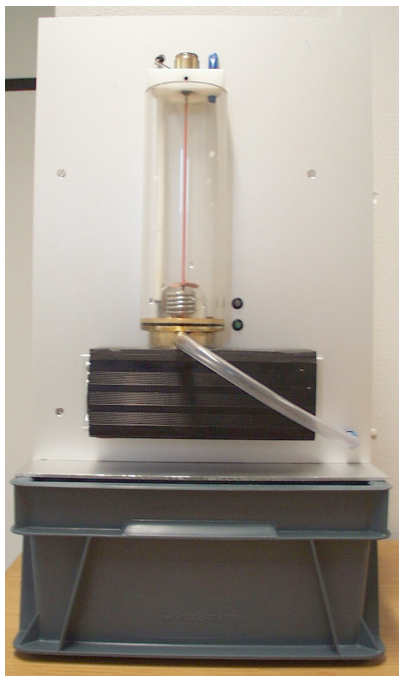
Projekt i FRTN25, Processreglering

Reglering av en CSTR-process

Institutionen för reglerteknik
Lunds tekniska högskola, Lunds universitet

Senast uppdaterad i maj 2013
av
Olof Garpinger och Ola Johnsson



1. Introduktion

Detta projekt behandlar styrning av en omrörd tank med kontinuerligt genomflöde (Continuous Stirred-Tank Reactor, CSTR). Projektet innefattar sekvensbaserad styrning, MIMO-system, modellering och regulatorinställning.

Processen är en tank med in- och utflöde där inflödet antas innehålla ett näringsmedium som ska värmas till 37°C inför nästa processteg, i labprocessen kommer dock vatten att användas. Denna temperatur uppnås med hjälp av en värmare och för att säkerställa en jämn uppvärmning av mediet finns även en omrörare i tanken. Utflödet ur tanken varierar beroende på efterfrågan i nästa processteg och regleringen måste kunna hantera dessa störningar så att såväl temperatur som vätskenivå i tanken påverkas så lite som möjligt.

Design av regulatorer för temperatur och vätskenivå ska göras med i processindustrin välkända inställningsmetoder för bestämning av regulatorparametrar, baserade på enkla processmodeller. Sekvensstyrning av processen används för att automatisera uppstart, kontinuerlig styrning av temperatur och vätskenivå, avslutning av processen samt rengöring av densamma.

Målet med projektet är att ni ska få erfarenhet av sekvensstyrning och MIMO-system samt möjlighet att lära er att använda enkla modelleringsverktyg och metoder för regulatordesign som är vanliga inom processindustrin.

Förberedelser

Innan arbetet med processen startas rekommenderas det att läsa igenom detta dokument. Det gäller speciellt de delar som beskriver hur JGrafchart fungerar, se appendix A.

1.1 Datorprogram och filer

Utvecklingen av styrsystemet för CSTR-processen ska göras i JGrafchart som är en grafisk Java-implementation av GRAFCET, utvecklad vid institutionen för reglerteknik i Lund. Arbetet kommer att utgå från filen start.xml, som innehåller grunden till det program som ska skapas under projektets gång. För loggning av data från processen används det separata programmet SockLog. Nedan beskrivs utförandet av olika procedurer i punktform.

För att starta första gången:

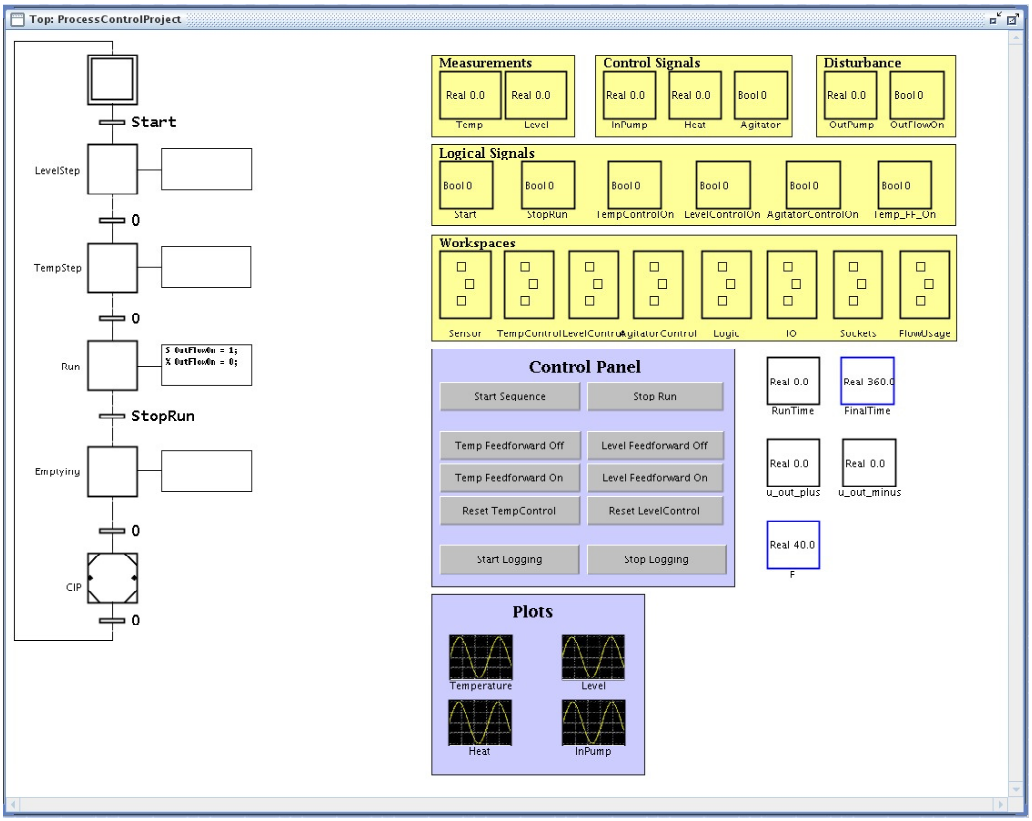
1. Logga in med ert studentkonto på en dator i laborationssalen.
2. Starta en webbrowser och hämta filerna start.xml och SockLog från kurshemsidan.
3. Öppna ett terminalfönster och starta JGrafchart med kommandot `JGrafchart`.
4. Öppna start.xml i JGrafchart.
5. Verifiera att gränssnittet ser ut som i figur 1.
6. Klicka på knappen *Properties* och verifiera att *Simulator Mode* är avmarkerat, se figur 2.
7. Klicka på knappen *OK* för att återgå till huvudfönstret.

För att exekvera programmet:

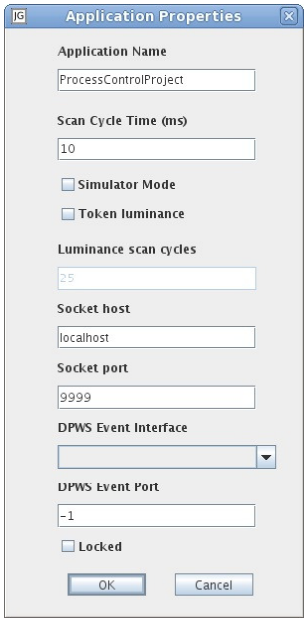
1. Se till att alla önskade ändringar i programmet gjorts.
2. Tryck på knappen *Compile* för att överföra ändringarna till programmet som ska exekveras, tillse att inga kompileringsfel uppstått.
3. Tryck på knappen *Execute* för att starta exekveringen.
4. Utför försök.
5. Tryck på knappen *Stop* för att stoppa exekveringen.

För att logga och analysera data:

1. Öppna ett nytt terminalfönster och gå till mappen där SockLog sparats.
2. Öppna SockLog med kommandot `sh SockLog`.
3. Starta exekvering av programmet enligt ovan.
4. Tryck på knappen *Start logging*.
5. Utför försök.



Figur 1 Huvudarbetsytan i start.xml.



Figur 2 Fönstret *Properties*.

6. Tryck på knappen *Stop logging*, namnge och spara filen.
7. Gör fler loggningar eller stoppa exekveringen enligt ovan.
8. Öppna och kör de sparade filerna i Matlab och analysera dem.

Observera att endast variabler som ändrats under den tid då loggningen utförts kom-

mer att sparas, detta gör att antalet variabler i den sparade filen kan variera.

2. Projektets mål

Målet för projektet är att utifrån den givna mallen skapa ett program för styrning av en CSTR-process. Styrningen är uppdelad i flera steg:

1. Startsteg, processen förbereds för körning.
2. Huvudsteg, processen körs i CSTR-läge.
3. Tömningssteg, tanken töms.
4. Rengörningssteg, tanken rengörs för att kunna köras igen.

Kommandon och villkor i JGrafchart används för att styra såväl vilka funktioner som ska vara aktiva i de olika stegen som övergångarna mellan stegen. I vissa steg ska PI-reglering användas för att styra processen och ge bra hantering av laststörningar; detta kräver modellering av processen, inställning av regulatorer samt implementation av dessa i JGrafchart.

2.1 Projektrapport och muntlig presentation

När alla uppgifter i denna manual är utförda ska det färdiga programmet demonstreras för en projektassistent. Därefter ska en projektrapport, skriven enligt samma mall som används vid inlämningsuppgifterna, lämnas in till projektassistenterna. Numret på den process som ert program testats på ska finnas angivet i rapporten. En fil innehållande det färdiga programmet ska skickas in tillsammans med rapporten.

Projektet ska även presenteras i diskussionsgrupper ledda av projektassistenterna, där arbetet, resultaten och reflektioner kring dessa ska diskuteras. Se kurshemsidan för aktuellt datum.

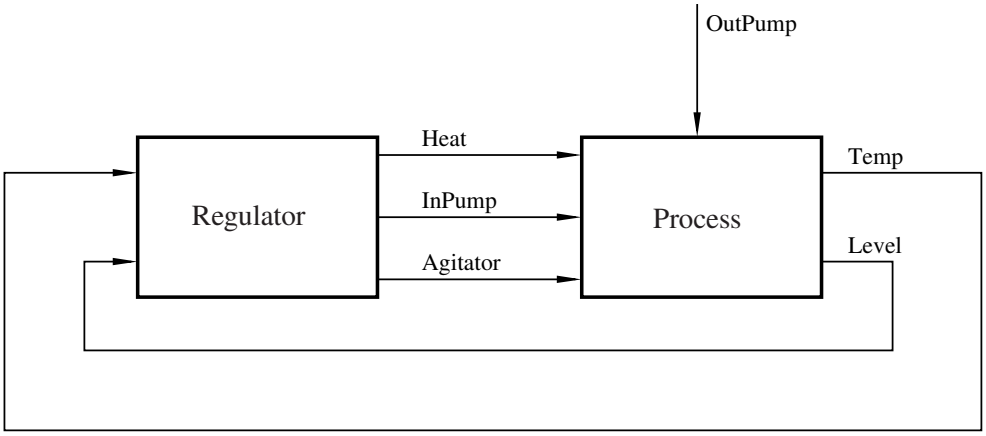
3. CSTR-processen

3.1 Praktisk information

Alla experiment kommer att utföras på institutionen för reglertekniks labprocesser. Det finns 6 processer och varje grupp rekommenderas att hålla sig till samma process under projektarbetets gång eftersom det kan skilja en aning i dynamik mellan processerna.

Innan processen kan användas måste den anslutas till dator och eluttag, därefter slås den på och av med strömbrytaren på dess sida. Vätskenivån i vattenkaret ska också kontrolleras; den ska täcka större delen av inpumpen. En LED-lampa på framsidan av processen lyser grönt när processen är redo och rött om ett fel har uppstått. De vanligaste felen är att de signaler som skickas till processen går emot dess interna säkerhetsregler:

- Vätskenivån måste ligga över värmeslingan när värmaren är påslagen, för att inte värmaren ska ta skada.



Figur 3 Blockdiagram över systemet och dess signaler. Signalernas benämningar är desamma som används i JGrafchart-mallen som ska användas.

- Om tanken nästan är full kan inte mer vätska tillsättas, detta för att förhindra att den svämmar över.

En reset-knapp finns på processens sida för att vid behov återställa den efter att ett fel uppstått; detta ska göras först efter att felet korrigerats.

3.2 Teori

Processen består av en tank med en uppsättning sensorer och aktuatorer, illustrerade i figur 3. Mätsignalerna är:

1. Vätsketemperatur
2. Vätskenivå

Insignalerna är spänningarna som läggs över inpump, utpump, omrörare, värmare och kylare. Utflödet kan ses som en störning eftersom det kommer att variera oberoende av regulatorn och kylaren används inte under projektet. Detta gör att tre insignaler återstår för regulatorn att använda:

1. Spänning över inpump
2. Spänning över omrörare
3. Spänning över värmare

Omröraren kommer antingen att vara påslagen med en fördefinierad spänning eller vara avslagen.

I processindustrin mäts signalstyrka ofta inte i fysiska storheter utan i procent av spannet mellan signalens högsta och lägsta värde (0-100%). I fallet med CSTR-processen motsvarar en nivåsignal på 0% tom tank medan 100% motsvarar full tank. Temperatur mäts på motsvarande sätt men antas här ha en enkel fysikalisk tolkning; 0% motsvarar 0°C och 100% motsvarar 100°C. Eftersom värmaren har svag effekt anger temperaturmätaren egentligen högre värden än de faktiska; på så vis simuleras en högre värmareffekt än vad som egentligen är fallet.

Uppgift 1 Resonera er fram till hur styrsignalerna påverkar mätsignalerna. Diskutera och motivera vilken styrsignal som bör användas för att styra vätskenivån och vilken som bör styra temperaturen. ◇

4. Modellering av vätskenivådynamiken

Volymändringen dV/dt i tanken är lika med differensen mellan inflöde q_{in} och utflöde q_{ut}, vilket beskrivs av massbalansekvationen

$$\frac{dV}{dt} = q_{in} - q_{ut} \quad [\mathrm{m}^3/\mathrm{s}] \tag{1}$$

Antag att in- och utflöde är proportionella mot den spänning som skickas till pumparna, det vill säga $q_{in} = k_1 u_{in}$ och $q_{ut} = k_2 u_{ut}$.

Uppgift 2 Ta fram överföringsfunktionerna från både inpump och utpump till vätskenivån i tanken, h. ◇

Både in- och utpump har en tröskelspänning som måste nås innan de börjar pumpa igenom vätska. Denna nivå kan variera något och är i regel större innan vätskan väl har börjat pumpas igenom.

Uppgift 3 Beräkna både inpumpens och utpumpens tröskelspänningar, det vill säga de lägsta insignalvärden för vilka de pumpar igenom vätska. Det bör observeras att tröskelspänningen varierar beroende på om det redan flödar vätska genom pumpen eller inte; endast fallet då processen går från flöde till inget flöde behöver studeras. Det är därför lämpligt att ange ett högt värde på u_{in} respektive u_{ut} och minska det stegvis tills flödet upphör; vid behov kan detta upprepas med mindre steg för att få mer exakta värden. För att göra detta, exekvera programmet och ange önskade värden (0-100%) genom att ändra parametern `InPump` respektive `OutPump`.

I den färdiga styrsekvensen ska utpumpspänningen variera mellan att vara 5 och 10 procentenheter högre än dess tröskelspänning, dessa nivåer benämns u_{ut}^- respektive u_{ut}^+. Ange värden för dessa parametrar i mallens huvudfönster, där de benämns `u_out_minus` respektive `u_out_plus`. ◇

Pumparnas dynamik är olinjär, inte minst på grund av den tröskelspänning för genomströmning som redan diskuterats. Inom det intervall i vilket pumparna ska köras kan dock den linjära modell som tagits fram i uppgift 2 användas, det är då viktigt att modellparametrarna bestäms för den arbetspunkt som ska användas. Den enklaste metoden för detta är att mäta nivåförändringen över tid för två olika inställningar på pumpspänningen, u_1 respektive u_2, och beräkna förstärkningen κ genom

$$\kappa = \frac{\left(\frac{dh}{dt}\right)_1 - \left(\frac{dh}{dt}\right)_2}{u_1 - u_2} \tag{2}$$

Uppgift 4 Finn lämpliga värden på u_1 och u_2 för både in- och utpumpen, utför nödvändiga försök och bestäm förstärkningen för in- respektive utpump, κ_1 respektive κ_2. ◇

5. Vätskenivåreglering

Som kan ses i ekvation (1) kan nivådynamiken betraktas som integrerande. En metod för design av PI-regulatorer för integrerande processer är *arresttidstrimning*, som beskrivs i appendix B.

Uppgift 5 Använd arresttidstrimning för att ta fram lämpliga regulatorparametrar för nivåstyrningen. Använd en arresttid $T_a = 6$ s. ◇

Uppgift 6 Finns det några begränsningar för hur lågt man kan sätta arresttiden T_a? Resonera kring detta. ◇

För att implementera regulatorn räcker inte endast regulatorparametrarna. För att undvika att aktuatorerna skadas krävs att utsignalen mättas av regulatorn, därtill måste integratoruppvridning motverkas i de fall då en I-del används. En av de enklaste metoderna för att undvika integratoruppvridning är *integrator-clamping*, som verkar enligt principen att om styrsignalen är mättad ska I-delen inte uppdateras utan behålla sitt tidigare värde.

I start.xml är P-reglering för nivåreglering implementerad i den arbetsyta som benämns `LevelControl`. I macrosteget `PI` kan dess struktur betraktas; i ett inledande steg beräknas den önskade styrsignalen ($v1$) och om den överskrider respektive underskrider sitt tillåtna intervall begränsas den till sitt maximala respektive minimala värde.

Uppgift 7 Implementera PI-reglering för vätskenivån i JGrafchart, använd ett referensvärde på 40%. Styrsignalen ska begränsas så att den ligger mellan inpumpens tröskelspänning och 100% och *integrator-clamping* ska användas för att motverka integratoruppvridning. Uttrycket för att uppdatera I-delen blir vid användning av framåt Euler: `Ipart = Ipart + K * h / Ti * (Lref-Level)`.

För att implementera nivåstyrningen:

1. Öppna `LevelControl`.
2. Ange önskade värden för K, T_i och referensvärdet L_{ref}.
3. Öppna modulen `PI` som finns i `LevelControl`.
4. Modifiera innehållet i `PI` för att ge PI-reglering.

När nivåstyrningen är implementerad och ska aktiveras sätts värdet av den booleska parametern `LevelControlOn` till 1, för att stänga av styrningen sätts parametern till 0. Justera om nödvändigt arresttiden och därmed även regulatorparametrarna till det att en stegändring i utpumpen från u_{ut}^- till u_{ut}^+ inte ger en nivåavvikelse på mer än 2% av tankens fulla höjd. Hur väl stämmer teori med verklighet beträffande storleken på T_a? ◇

6. Modellering av temperaturdynamiken

Användandet av återkoppling vid processreglering möjliggör användandet av enkla processmodeller eftersom regleringen kan kompensera för fel i modellen. En modell-

typ som trots sin enkelhet kan användas för att beskriva många processer i processindustrin är en första ordningens modell med tidsfördröjning:

$$P(s) = \frac{K_p}{sT+1} e^{-sL} \tag{3}$$

För att bestämma en modell av typen (3) för en process med okänd dynamik kan ett stegsvarsexperiment utföras. Detta görs genom att lägga processens utsignal nära den arbetspunkt den ska köras vid, eftersom en olinjär process kan ha varierande beteende vid olika arbetspunkter, och sedan göra ett steg i insignalen. När detta görs bör processen vara i jämvikt så att stegsvaret inte störs av andra effekter. I figur 4 ges ett exempel på ett sådant försök, där den önskade arbetspunkten ligger vid 10%. I figuren har ett steg av magnitud 1% gjorts i styrsignalen, vilket ger upphov till en förändring av processens utsignal.

För att hitta de tre processparametrarna i (3) utifrån ett stegsvarsexperiment kan följande procedur tillämpas:

1. Bestäm förstärkningen K_p genom att dela förändringen i utsignal med förändringen i utsignal efter det transienta förloppet, $K_p = \frac{\Delta y}{\Delta u}$.
2. Dra en tangent till utsignalen där dess lutning är som störst, motsvarande den rödstreckade linjen i figur 4.
3. Bestäm dödtiden L som tiden från det att stegändringen gjordes till den punkt där tangenten från föregående punkt skär utsignalens tidigare jämviktsnivå, 10% i figur 4.
4. Beräkna den tid $t_{63\%}$ för vilken förändringen i y har nått 63% av Δy.
5. Bestäm tidskonstanten T genom att subtrahera dödtiden från den tid som uppmättes i föregående punkt, $T = t_{63\%} - L$.

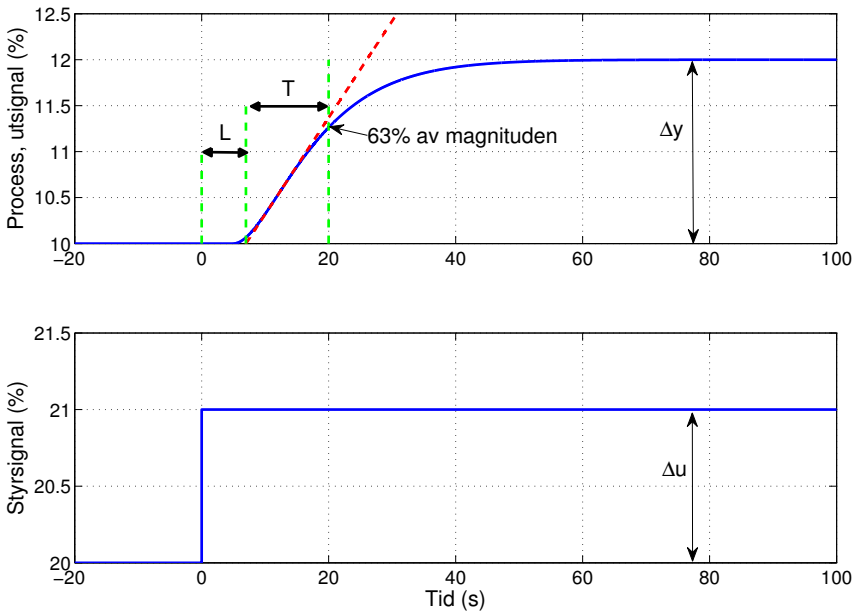
I exemplet i figur 4 fås följande värden:

$$\begin{aligned} K_p &= \frac{2\%}{1\%} = 2 \\ L &= 7 - 0 = 7 \text{ s} \\ T &= 20 - 7 = 13 \text{ s} \end{aligned}$$

Detta gör att processmodellen i exemplet kan defineras till

$$P(s) = \frac{2}{13s+1} e^{-7s} \tag{4}$$

En nackdel med stegsvarsmetoden är att testet blir känsligt för störningar, vilket är ett problem i CSTR-processen. Eftersom vätskan recirkuleras kommer den inkommande vätskans temperatur att bli allt högre så länge vätskan i tanken värms. Detta kan ses som en långsam integrerande dynamik i processen, vilken ger ett stegsvar liknande den blå heldragna kurvan i figur 5. Det röda streckade stegsvaret visar hur svaret skulle sett ut utan störning, alltså detsamma som i figur 4. Detta gör att modellstrukturen i (3) inte stämmer fullständigt men eftersom denna störande dynamik är förhållandevis långsam kan den bortses från vid modelleringen. Eftersom denna dynamik inte innefattas i modellen kan den betraktas som en störning, vilken försvårar bestämmandet av modellparametrarna vid stegsvarsförsök. Normalt är det dock möjligt att utifrån stegsvarets form avgöra ungefär hur stegsvaret skulle sett ut utan den integrerande dynamiken.



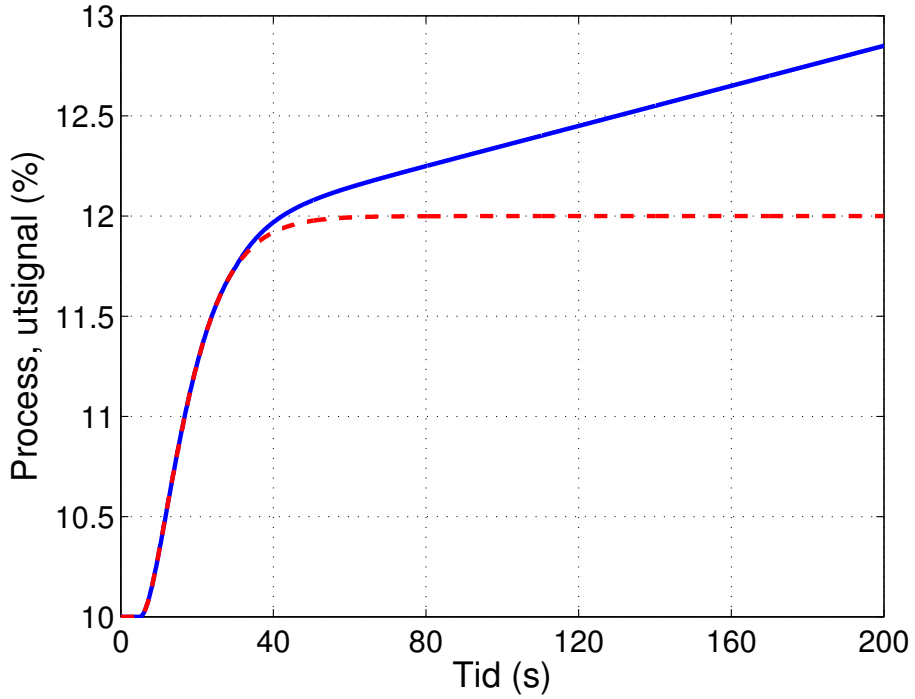
Figur 4 Stegsvar och modellering av en första ordningens process med dödtid.

Uppgift 8 Utför stegsvarsexperiment enligt beskrivningen och skatta samtliga modellparametrar i (3). Den parameter som styr värmarspänningen är `Heat` (0-100%). Detta kräver att det finns ett kontinuerligt vätskeflöde genom tanken; utpumpens spänning bör sättas till u_{ut}^{-} och den nivåreglering som tagits fram i uppgift 7 kan användas för att ge en jämn vätskenivå under försöket. För att ge en jämn uppvärmning av vätskan ska styrningen av omröraren vara påslagen; den slås på och av genom att parametern `AgitatorControlOn` sätts till 1 respektive 0.

Stegsvaren bör göras runt temperaturen 37°C; ett lämpligt tillvägagångssätt är som följer:

1. Hitta ett värde på `Heat` som ger temperaturen 37°C och benämn detta värde u_0.
2. Stoppa värmningen, låt vattnet svalna eller ersätt det med rumstempererat vatten.
3. Sätt `Heat` till $0.9u_0$ och låt temperaturen stabilisera sig.
4. Öka `Heat` till $1.1u_0$ och låt temperaturen stabilisera sig, detta ger ett temperatursteg runt 37°C som kan användas för modelleringen.
5. Vid behov kan steg 2-4 upprepas med större amplitud för steget, exempelvis från $0.8u_0$ till $1.2u_0$.

Visa tydligt i rapporten hur K_p, L och T har bestämts, utnyttja möjligheterna att logga data och sedan analysera den i Matlab. ◇



Figur 5 Stegsvar är känsliga för störningar. En process som egentligen skulle haft det rödstreckade stegsvaret kan komma att störas så att det ser ut som det blå svaret.

7. Temperaturreglering

En vanlig metod för val av regulatorparametrar till PI-regulatorer i processindustrin kallas för λ-metoden (lambda-metoden). Den förutsätter att en första ordningens modell med tidsfördröjning, (3), kan användas för att beskriva processens dynamik. λ-metoden beskrivs i appendix B.

Uppgift 9 Använd λ-metoden för att ta fram lämpliga regulatorparametrar för temperaturstyrningen utifrån er modell. Använd tidskonstanten $T_{cl} = T$ för det slutna systemet. ◇

Uppgift 10 Implementera temperaturstyrningen i JGrafchart med ett referensvärde på 37%. Även här måste ett flöde genom tanken samt omrörning av vätskan ske, lämpligen på samma sätt som i uppgift 8. Justera om nödvändigt T_{cl} och därmed även regulatorparametrarna om styrningen ger upphov till oscillationer i `Temp` eller om den upplevs som för långsam.

För att implementera temperaturstyrningen, gör på motsvarande sätt som i uppgift 7 men använd den arbetsyta som benämns `TempControl`. ◇

Uppgift 11 Gör en stegstörning i utpumpen så att denna går från u_{ut}^{-} till u_{ut}^{+}. Hur stort blir det maximala felet i temperaturen? Ge minst två förslag på åtgärder man skulle kunna vidta för att minska störningens inverkan. ◇

8. Framkoppling

För att förbättra vätskenivåregleringen i tanken kan en statisk framkoppling läggas till från utpump till inpump. Denna kan implementeras genom att i `LevelControl` ange ett värde för framkopplingens förstärkning FF, kopiera PI-regulatorn för nivåreglering till modulen `PIandFF` och i denna modul också lägga till en framkopplingsterm. På detta vis går det enkelt att slå på och av framkopplingen; i huvudfönstret trycks knappen **Level Feedforward On** på för att slå på framkopplingen och motsvarande off-knapp för att stänga av den.

Uppgift 13 Designa en statisk framkoppling från utpumpspänning till inpumpspänning så att inverkan på vätskenivån från en störning i utpumpspänningen elimineras, utifrån processmodellen. Hur bör framkopplingen relatera till κ_1 och κ_2? Hur bör framkopplingens påverkan på styrsignalen sättas i relation till styrsignalmättning och motverkan av integratoruppvridning?

Implementera framkopplingen och kör processen med nivåstyrning aktiverad och ett utflöde på u_{ut}^-. Gör en stegstörning i utflödet med amplituden 20 procentenheter. Hur mycket lägre blir det maximala felet i vätskenivån jämfört med när framkopplingen inte används?

Vad sker omedelbart när processen körs utan framkoppling och denna sedan kopplas på? Vad beror detta på och hur skulle det kunna undvikas? ◇

9. Färdigställande av styrsekvensen

Alla parametrar som ska användas för reglering av processen är nu kända och det är dags att sammanfoga dem till en helhet i JGrafchart. Detta ska göras steg för steg i den JGrafchart-fil där nivå- och temperaturstyrningen implementerats, enligt den procedur som beskrivs här. I många fall finns det flera olika sätt att utföra detta, där vissa sätt är mer effektiva än andra.

9.1 Nivåsensor

Vätskenivån kan variera kontinuerligt mellan 0 och 100 procent. En struktur som med booleska variabler informerar om ifall tanken är full samt om den är tom är önskvärd då den underlättar flera funktioner i programmet.

Uppgift 14 Gå in i arbetsytan Sensor. Den innehåller tre booleska variabler samt en struktur för att styra parametern `LevelLow` som används för att säkerställa att värmning och omrörning inte körs när vätskenivån är för låg. Skapa en ny struktur i denna arbetsyta med villkor och kommandon så att följande är uppfyllt:

- `Full` ska vara 1 för `Level >= 70`, annars 0.
- `Empty` ska vara 1 för `Level <= 10`, annars 0.

◇

9.2 Förberedelsesteg

Innan processen kan köras med ett kontinuerligt flöde genom tanken ska den förberedas; den är redo för att köras när nivån och temperaturen ligger vid eller lite över referensvärdet och omröraren är påslagen. Detta måste göras i flera steg eftersom fyllning och uppvärmning inte kommer att vara klara vid samma tidpunkt om de startas samtidigt.

Vid fyllning av tanken kan inpumpen styras direkt genom att sätta `InPump = F`. Ett värde för F finns redan angivet och detta rekommenderas att användas. Omröraren sätts igång genom att sätta parametern `AgitatorControlOn` till 1, dock kommer omröraren att stå stilla om inte parametern `LevelLow` har värdet 0. Detsamma gäller för temperaturregleringen; som kan ses i TempControl aktiveras den endast om `LevelLow` är 0.

Uppgift 15 Bestäm ett lämpligt sätt att uppnå det önskade tillståndet och skapa en procedur för detta i JGrafchart-programmet, med utgångspunkt i den struktur som finns till vänster i huvudfönstret. ◇

9.3 Huvudsteg

Huvudsteget i processen benämns Run i start.xml, det är viktigt att det inte döps om. Detta steg fortgår tills parametern StopRun får värdet 1, vilket sker när parametern `RunTime`, som mäter tiden under vilken huvudsteget pågått, nått värdet som angetts i `FinalTime` eller när användaren trycker på knappen märkt **Stop Run**. Under detta steg ska ett varierande utflöde vara igång, vilket är implementerat i mallen genom att parametern `OutFlowOn` ges värdet 1. Det initiala utflödet är u_{ut}^{-} och en tid in i detta steg ändras utflödet till u_{ut}^{+}.

Uppgift 16 Implementera reglering av nivå och temperatur i processens huvudsteg. Även omröraren ska vara igång under detta steg. ◇

9.4 Avslutningssteg

När huvudsteget är avslutat ska all reglering samt utflödet stängas av och tanken tömmas. Tömningen ska göras genom att sätta parametern `OutPump` till ett lämpligt värde, till exempel F. När tanken är tom (`Sensor.Empty` är 1) ska rengöring av tanken ske vilket görs i makrosteget som benämns CIP (Cleaning In Place), makrosteget öppnas på samma sätt som en arbetsyta och i det kan flera vanliga processteg användas för att skapa en struktur. I detta makrosteg ska tanken först fyllas, för detta kan parametern `Sensor.Full` användas, och därefter åter tömmas; när detta har skett betraktas tanken som rengjord och processen kan återgå till sitt utgångsläge.

I en riktig process skulle rengörning av tanken givetvis inte göras med samma vätska som används i tanken under körning. I detta fall representeras rengörningen dock av detta eftersom uppställningen endast har ett in- och ett utflöde.

Uppgift 17 Designa processens avslutningssteg enligt ovanstående specifikationer. ◇

10. Testning av processen

Starta processen genom att trycka på knappen **Start Sequence**. Säkerställ att rätt JGrafchart-kommandon getts vid rätt tillfällen så att processens beteende är det önskade. Utnyttja möjligheterna att logga data och sedan analysera den i Matlab för att generera grafer till er rapport.

11. Slutsatser

I detta projekt har ett program skapats för styrning av en CSTR-process. Programmet innehåller både sekvensstyrning och PI-regulatorer för nivå- och temperaturstyrning. Denna kombination av regulatorer och logikbaserad sekvensstyrning är vanlig och återfinns i många olika typer av processer, från industriprocesser till vardagliga apparater såsom tvättmaskiner.

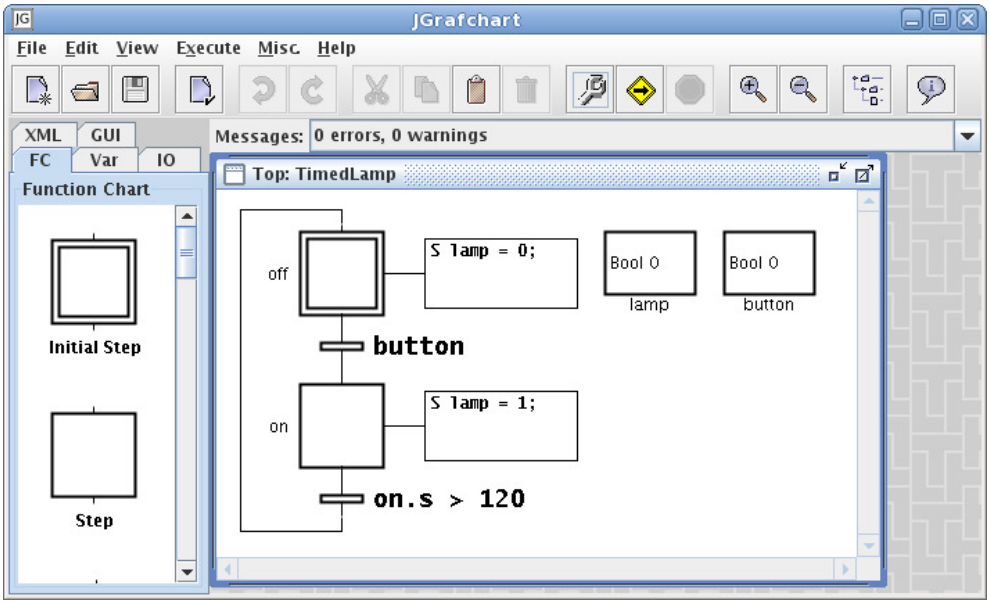
Uppgift 18 Reflektera över det arbete som utförts i detta projekt. Diskutera modelleringen, regleringen och sekvensstyrningen; vad har gjort att den fungerat bra eller mindre bra och hur skulle den kunna förbättras? Vad har ni lärt er under projektets gång, både övergripande och vad gäller detaljer i regleringen? ◇

A. Introduktion till JGrafchart

JGrafchart är ett program för att utveckla sekvensstyrning i en grafisk miljö, det är gratis och utvecklat på institutionen för reglerteknik på LTH. Detta appendix kommer endast att behandla den delmängd av alla funktioner i JGrafchart som är relevanta för projektet.

A.1 Exempel

Ett exempel på ett enkelt JGrafchart-program visas i figur 6. Programmet går ut på att en lampa ska vara påslagen i två minuter efter att en knapp tryckts på. De booleska



Figur 6 Ett exempel på hur JGrafchart fungerar i form av en lampa som slås på och därefter är tänd under 120 sekunder innan den åter släcks. Alla tillgängliga steg och övergångar finns till vänster i bild, själva programmet ligger till höger om dessa medan kompileringsmeddelanden är listade precis ovanför programmet.

variablerna `lamp` och `button` innehåller information om ifall lampan är på, 1, eller av, 0, respektive om knappen är nedtryckt, 1, eller ej, 0. De två stegen (*steps*) `off` och `on` svarar mot om lampan är av eller på. När `button` sätts till 1 sker en övergång (*transition*) från steget `off` till steget `on`, varpå lampan slås på, `lamp=1`. När lampan har lyst i mer än 120 sekunder sker ytterligare en övergång, denna gång från `on` till `off`. I samband med detta kommer lampan släckas eftersom `lamp` sätts till 0. Notera att programexekveringen startar i initialsteget, i detta fall `off`, som är markerat med dubbel ram. Det steg som är aktivt för tillfället markeras med en svart cirkel i programmet när det körs, detta syns ej i bilden.

A.2 Programmering i JGrafchart

Programmering i JGrafchart görs genom att klicka på steg, övergångar, variabler, etc. i menyn till vänster och sedan dra dessa till programmet till höger. Det går att markera, ta bort, kopiera, klippa ut och klistra in samtliga objekt precis som i många andra standardtillämpningar. Steg och övergångar kopplas samman genom att klicka och dra i de korta streck som sitter i över- och underkanten på de olika objekten. Vissa regler måste följas, det går till exempel inte att koppla ihop två steg utan att det finns en övergång mellan dem.

A.3 Kompilering

Innan man kan exekvera (köra igång) ett program måste det kompileras, vilket görs genom att antingen trycka på *Compile*-knappen i verktygsfältet eller genom att välja *Compile* i *Execute*-menyn. Eventuella kompileringsfel indikeras med röd text och dyker upp som fel bland kompileringsmeddelandena, *Messages*, precis ovanför programmet.

Två typer av fel kan inträffa vid kompilering, nämligen syntaxfel (syntactic errors) och semantiska fel (semantic errors). Till exempel skulle villkoret `y OR z` leda till ett syntaxfel eftersom `y OR z` kommer uppfattas som tre på varandra följande variabler snarare än villkoret att `y` *eller* `z` ska vara lika med 1. Istället måste man skriva `y | z`. Semantiska fel uppstår när man försöker kompilera något som inte är möjligt, till exempel ifall variabeln `y` i `y | z` inte är definierad.

När ett program ska startas, genom att *Execute*-knappen trycks på eller *Execute* väljs i *Execute*-menyn, kommer den senast kompilerade versionen av programmet att startas. Detta innebär att förändringar i programmet som gjorts efter den senaste kompileringen kommer inte att påverka det program som körs.

A.4 Exekvering

Programmen i JGrafchart exekveras periodiskt. I varje cykel sker följande:

1. Digitala och analoga ingångar läses av, i detta projekt temperatur- och nivåvärden.
2. De övergångar vars villkor uppfyllts markeras.
3. Alla markerade övergångar slås till så att först de avaktiverade stegen slutexekveras och sedan de aktiverade stegen startexekveras.
4. Uppdatering av stegklockor, som ges tillträde till med <stegNamn>.t eller <stegNamn>.s, för att exekvera periodiska åtgärder och förbereda booleska åtgärder.
5. Updatera variabler som påverkas av booleska åtgärder.

A.5 Syntax

JGrafcharts syntax är lik den som finns i Java. En viktig skillnad är att 0 och 1 används i såväl booleska variabler för sant/falskt som för siffrorna 0 och 1 i heltalsvariabler. Det är sammanhanget som avgör betydelsen.

Följande operatorer stöds i JGrafchart: +, -, *, /, ! (negation), & (och), | (eller), == (ekvivalens), != (skilt från), <, >, <=, >=.

Uttryck kan innehålla referenser till variabler som syftar till en viss del av ett program, en så kallad arbetsyta (*workspace*). Till exempel är en variabel `Y` i arbetsyta `W1` skild från en variabel `Y` i arbetsyta `W2`. Referenser mellan arbetsytorna kan ske genom punktnotation. En åtgärd i ett steg i W1 kan alltså referera till variabeln `Y` i `W2` genom att man skriver `W2.Y`.

Uttrycket <stegNamn>.x returnerar 1 om steget är aktivt och 0 annars, medan uttrycket <stegNamn>.t returnerar antalet tidscykler sedan steget senast aktiverades, 0 om det inte aktiverats. Uttrycket <stegNamn>.s ger tiden, i sekunder, sedan steget senast aktiverades och 0 ifall det inte aktiverats.

A.6 Åtgärder

Till varje steg hör ett antal åtgärder (*actions*) som exekveras i samband med att steget aktiveras, körs eller deaktiveras.

Sammanlagt stöds fyra olika åtgärdstyper i JGrafchart:

- **Ingångsåtgärd**: Åtgärd som exekveras när ett steg aktiveras.
  `S <åtgärd>;`
- **Periodisk åtgärd**: Åtgärden exekveras periodiskt, en gång per tidscykel, under den tid steget är aktivt.
  `P <åtgärd>;`
- **Utgångsåtgärd**: Åtgärden exekveras direkt efter att steget deaktiverats.
  `X <åtgärd>;`
- **Boolesk åtgärd**: Kopplar tillståndet hos en boolesk variabel till huruvida steget är aktivt eller ej så att variabeln blir 1 när steget aktiveras och 0 när steget deaktiveras.
  `N <boolesk variabel>;`

<åtgärd> indikerar antingen en tilldelning eller ett anrop:

- **Tilldelning**: <variabel> = <uttryck>
- **Anrop**: <metod>()

A.7 Villkor

Till varje övergång hör ett villkor som ger tillbaka antingen 0 eller 1. Blir det 0 kommer övergången inte kunna ske, medan 1 ger möjlighet till övergång ifall steget innan övergången är aktivt.

A.8 Byggblock i JGrafchart

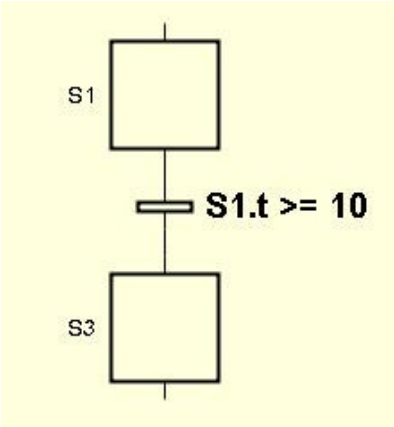
Steg Stegets namn visas på dess vänstra sida, de åtgärder som sker i steget visas till höger om dessa. Genom att högerklicka på ett steg kan man dölja/visa de åtgärder (*actions*) som hör till steget, ändra åtgärderna (*edit*) samt ändra stegets namn.

Obs! En åtgärd måste alltid avslutas med ett semikolon.

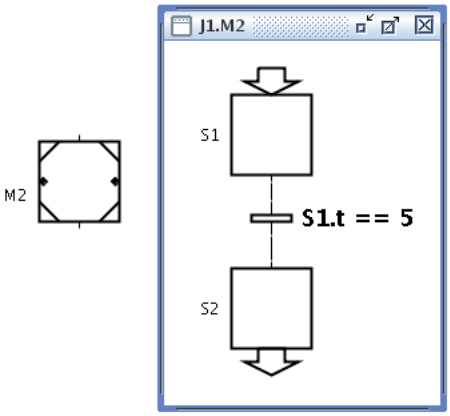
Initialsteg Initialsteg är de steg som aktiveras när exekveringen av ett funktionsdiagram startas.

Övergångar Övergångar (*transitions*) kopplas till villkor eller händelser som ska vara sanna, 1, för att möjliggöra en övergång mellan två steg.

Macrosteg Ett macrosteg (*macro step*) representerar en hierarkisk abstraktion som innehåller en egen arbetsyta, se figur 8. Arbetsytan öppnas eller stängs genom att högerklicka på steget och välja `Show/Hide Body`. Det första steget i ett macrosteg är ett specifikt ingångssteg där exekveringen startar. På samma sätt innehåller macrosteget ett utgångssteg där exekveringen av steget slutar. Både ingångs- och utgångssteg kan kopplas till olika åtgärder på samma sätt som andra steg. En stor fördel med att använda macrosteg är att det möjliggör uppdelning av ett större program i många mindre delar, vilket ger större överskådlighet och tydlighet.



Figur 7 En övergång



Figur 8 Ett macrosteg M2 och dess tillhörande arbetsyta som innehåller ingångssteget S1, utgångssteget S2 samt en övergång däremellan.

Variabler En variabel i JGrafchart kan vara av typen *reell, boolesk, heltal* eller *textsträng* (*real, boolean, integer* eller *string*) och har alltid ett värde och ett namn kopplat till sig. Variabler representeras av boxar där variabelns namn står under boxen och dess värde i boxens mitt.

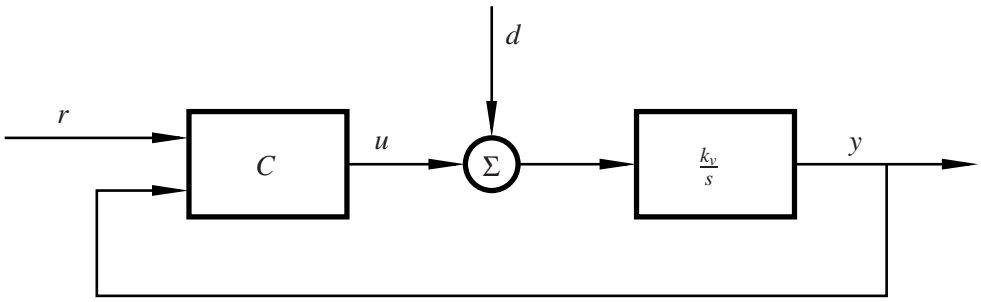
Åtgärdsknappar En åtgärdsknapp (*action button*) utför en bestämd åtgärd när man klickar på den.

Arbetsytor En arbetsyta (*Workspace Object*) skapar en egen avgränsad del av programmet och används för att strukturera programmet. Bland annat PI-regulatorerna kommer att implementeras i sådana arbetsytor under detta projekt.

Text *Text*-objekt kan användas som kommentarer i programmet.

A.9 Dokumentation

För mer detaljerad information om JGrafchart rekommenderas att öppna *Online Help* under *Help*-menyn. Man kan också snabbt få mer information om ett visst objekt i JGrafchart genom att klicka på knappen *Object Help* och sedan klicka på det objekt man önskar veta mer om.



Figur 9 Blockdiagram för styrning av en integrerande process.

B. Design av PI-regulatorer

I detta appendix beskrivs teorin för de två metoder för regulatordesign som används i projektet, arresttidstrimning och lambda-metoden.

B.1 Arresttidstrimning

Arresttidstrimning är en enkel designmetod för PI-regulatorer, anpassad för begränsning av laststörningar i processer där överföringsfunktionen från insignal till mätsignal kan betraktas som en integrator:

$$Y(s) = \frac{k_v}{s} U(s) \tag{5}$$

Reglering av en integrerande process illustreras i figur 9. Processens arresttid, T_a, definieras som den tid det tar för processens utsignal att vända tillbaka mot börvärdet efter en stegstörning på processingången, d, detta illustreras i figur 10. De värden på regulatorparametrarna som krävs för att ett önskat värde på T_a ska ges beräknas enligt

$$K = \frac{2}{k_v T_a}, \tag{6}$$

$$T_i = 2T_a, \tag{7}$$

B.2 Lambda-metoden

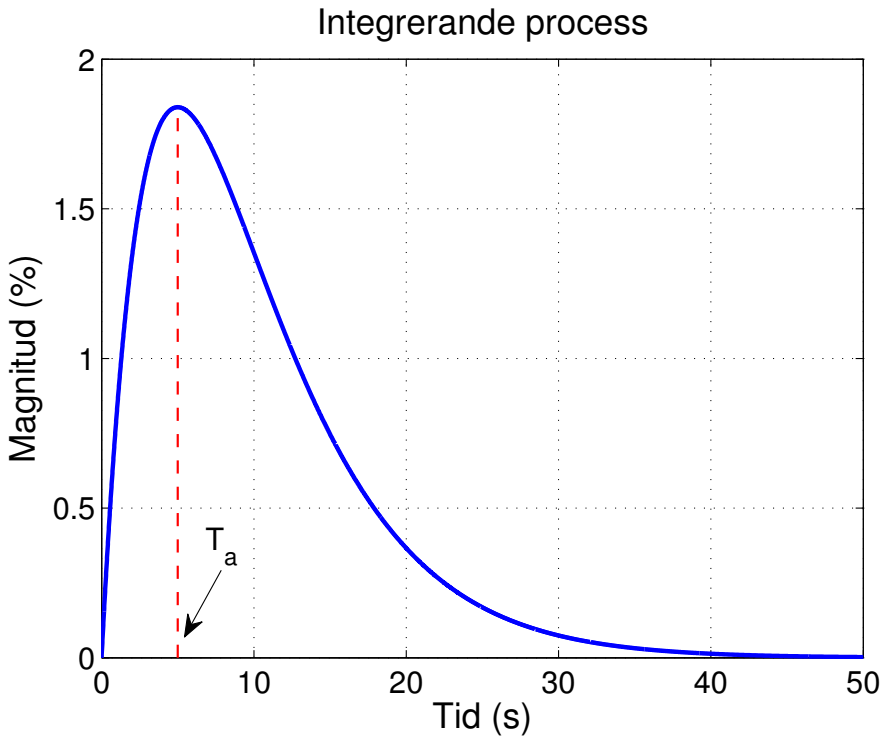
λ-metoden (lambda-metoden) är en vanlig metod för inställning av PI-regulatorer för processer som kan beskrivas med en första ordningens modell med tidsfördröjning:

$$P(s) = \frac{K_p}{sT + 1} e^{-sL} \tag{8}$$

Genom att ange ett önskat värde för det slutna systemets tidskonstant från referens till processens utsignal, T_{cl}, beräknas regulatorparametrarna enligt

$$K = \frac{1}{K_p} \frac{T}{L + T_{cl}} \tag{9}$$

$$T_i = T. \tag{10}$$



Figur 10 Arresttiden är den tid det tar innan utsignalvärdet efter en laststörning på processingången till en integrerande process vänder mot börvärdet igen.

Regulatorn får då överföringsfunktionen

$$C(s) = \frac{1}{K_p}\frac{T}{L+T_{cl}}\left(1+\frac{1}{sT}\right) \tag{11}$$

Det slutna systemets överföringsfunktion från referenssignal till utsignal blir då

$$G_{cl}(s) = \frac{PC}{1+PC} = \frac{\frac{1}{K_p}\frac{T}{L+T_{cl}}(\frac{sT+1}{sT})\frac{K_p}{1+sT}e^{-sL}}{1+\frac{1}{K_p}\frac{T}{L+T_{cl}}(\frac{sT+1}{sT})\frac{K_p}{1+sT}e^{-sL}} = \frac{e^{-sL}}{s(L+T_{cl})+e^{-sL}} \tag{12}$$

Genom att integraltiden sätts till samma värde som tidskonstanten förkortas processmodellens pol bort. Att T_{cl} benämns som det slutna systemets tidskonstant blir tydligt för system vars tidsfördröjning är 0:

$$G_{cl_{L=0}}(s) = \frac{1}{sT_{cl}+1} \tag{13}$$

Detta innebär att för system vars tidsfördröjning L är tillräckligt liten i förhållande till tidskonstanten T för att L ska kunna betraktas som 0 är överföringsfunktionen från referens till utsignal av första ordningen, med förstärkningen 1 och tidskonstanten T_{cl}. Parametern T_{cl} benämndes ursprungligen λ, vilket är det som gett inställningsmetoden dess namn.

T_{cl} kan i de flesta fall relateras direkt till såväl hastigheten som robustheten hos det slutna systemet; i regel kan man säga att ju snabbare det slutna systemet görs desto sämre robusthet kommer det att ha. Ju mer pålitlig modellen är desto mindre robust

behöver det slutna systemet också vara och regleringen kan göras snabbare; i praktiken är många processmodeller dock inte helt pålitliga och god robusthet krävs.

Ett vanligt val av värde för T_{cl} är processens tidskonstant, $T_{cl} = T$. Om tidsfördröjningen är stor i förhållande till tidskonstanten kan det vara bättre att istället sätta $T_{cl} = L$. Om tidsfördröjningen är förhållandevis liten kan $T_{cl} = T$ tvärtom vara alltför konservativt och lägre värden på T_{cl} kan då användas.