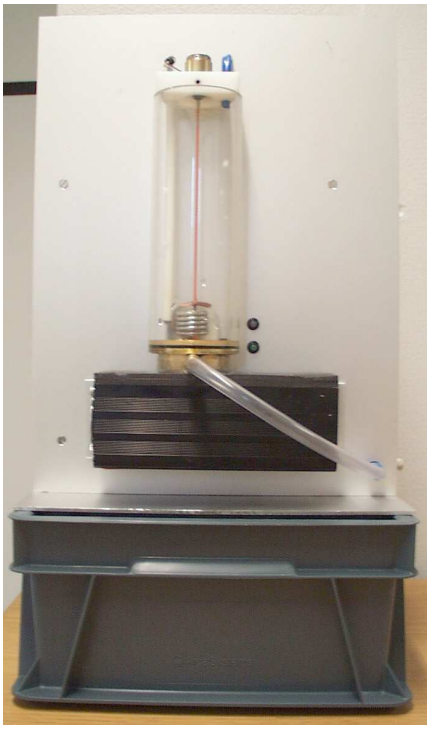
Miniprojekt i FRTN25 Processreglering

Reglering av en CSTR-process

Olof Garpinger, Martin Hast och Ola Johnsson
Institutionen för reglerteknik
Lunds tekniska högskola
Senast uppdaterad: april 2015



1. Inledning

Detta projekt behandlar styrning av en omrörd tank med kontinuerligt genomflöde (Continuous Stirred Tank, CST). Projektet innefattar modellering, regulatorimplementation, -inställning och sekvensbaserad styrning av ett system med flera in- och utsignaler (MIMO).

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 antas variera beroende på efterfrågan i något annat processteg. Efterfrågan kommer att ses som en störning, och regleringen måste kunna hantera dessa så att såväl temperatur som vätskenivå i tanken påverkas så lite som möjligt.

Valet av regulatorparametrar, för temperatur och vätskenivå, kommer göras med inställningsmetoder som är välkända inom processindustrin. Dessa baseras på enkla processmodeller som ni kommer ta fram. För att automatisera uppstart, drift, avslutning av processen samt rengöring av densamma kommer sekvensstyrning att användas.

Projekets mål

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

I projektet ska ni utifrån den givna mallen skapa ett program för styrning av CSTR-processen. Styrningen kan delas upp i tre steg:

1. Förberedelsesteg
2. Huvudsteg
3. Avslutningssteg

I förberedelsesteget ska processen, som namnet antyder, förberedas för den kontinuerliga driften i huvudsteget. I huvudsteget kommer en förbestämd störningssekvens påverka systemet. Målet är att med hjälp av reglering se till att systemet klarar av att hantera och undertrycka störningarna. I avslutningssteget ska regleringen stängas av, tanken ska tömmas och till sist rengöras.

Projektrapport och muntlig presentation

När alla uppgifter i denna manual är utförda ska det färdiga programmet demonstreras och bli godkänt av en projektassistent. Därefter ska en projektrapport lämna in, denna kan antingen skrivas enligt samma mall som används vid inlämningsuppgifterna eller som en laborationsrapport. Numret på den process som ert program testats på ska finnas angivet i rapporten och en fil innehållande det färdiga programmet ska skickas in tillsammans med rapporten. Projektrapporten ska innehålla de experimentella data, lämpligen i form av grafer, som krävs för att motivera era svar på de uppgifter som ska lösas.

Projektet ska även presenteras muntligt i diskussionsgrupper där arbetet, resultaten och reflektioner kring dessa ska diskuteras.

Förberedelser

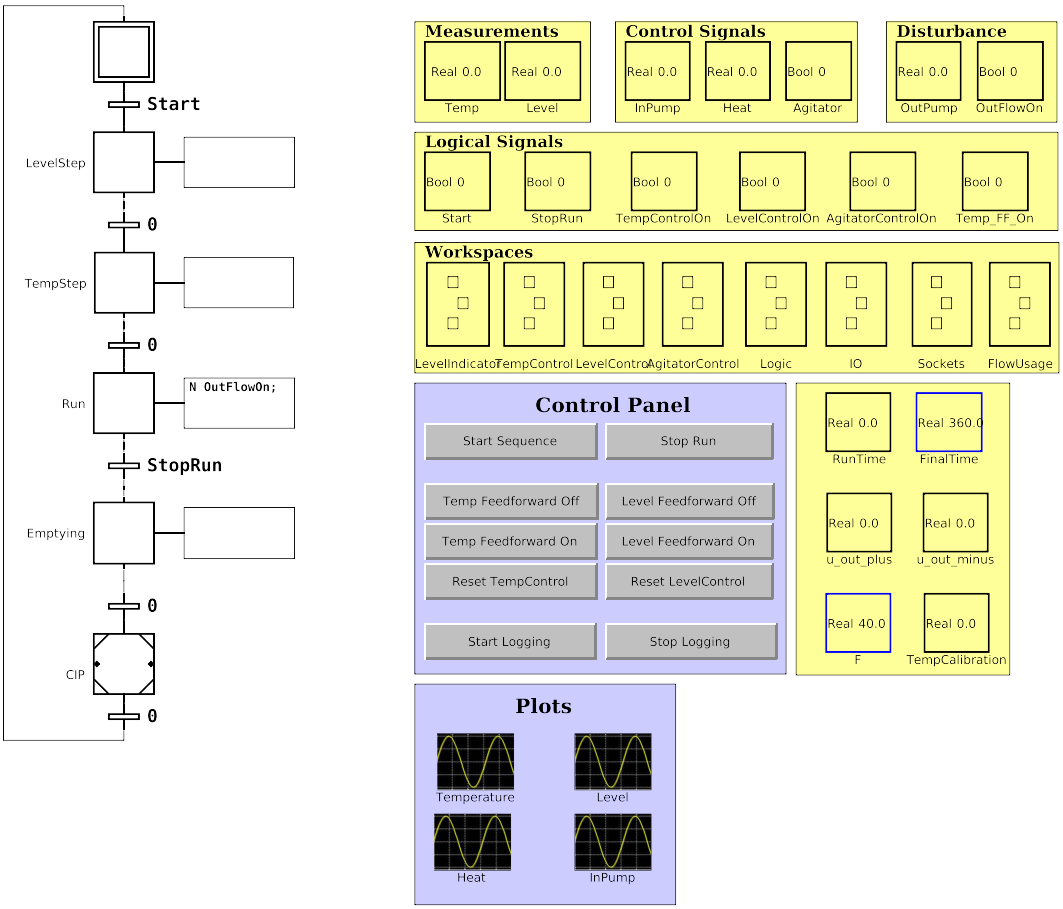
Innan arbetet med processen påbörjas ska **hela** projektbeskrivningen läsas igenom. Det gäller särskilt de delar som beskriver hur JGrafchart fungerar, se appendix A.

2. Programvaror och filer

Utvecklingen av styrsystemet för CSTR-processen ska göras i JGrafchart, som är en grafisk Java-implementation av GRAFCET. Arbetet kommer att utgå från filen `start.xml` som är grunden till det program som ska skapas under projektets gång. För loggning av data från processen används det separata programmet `SockLog`. Dessa filer kan laddas ner från kurshemsidan. 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`, `SockLog.sh` och `SockLog.jar` från kurshemsidan.
3. Starta JGrafchart genom att öppna ett terminalfönster och skriva `JGrafchart`.



Figur 1 Huvudarbetsytan i start.xml.

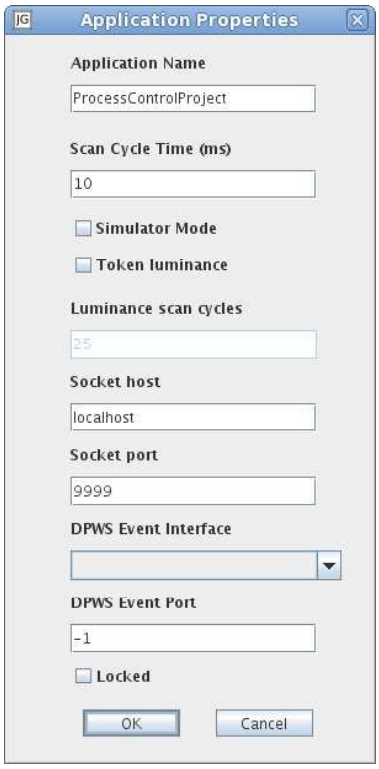
4. Öppna start.xml i JGrafchart.
5. Verifiera att gränssnittet ser ut som i figur 1.
6. Klicka på knappen *Properties* och verifiera att *Simulator Mode* är avmarkerat, se figur 2.
7. Klicka på knappen *OK* för att återgå till huvudfönstret.

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. (Ett felmeddelande kommer att visas om SockLog inte har startats först, se nedan. Om du inte tänker använda loggning kan du trycka *Ignore* för att fortsätta.)
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. Kör SockLog med kommandot `sh SockLog.sh`.



Figur 2 Fönstret *Application Properties*.

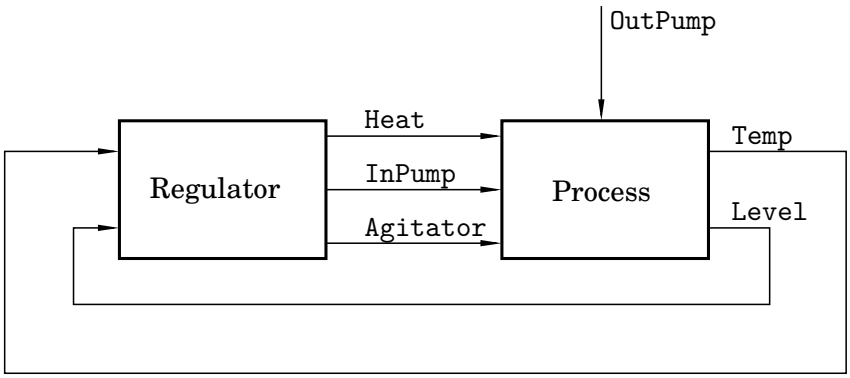
3. Starta exekvering av programmet enligt ovan.
4. Tryck på knappen *Start logging*.
5. Utför försök.
6. Tryck på knappen *Stop logging*, namnge och spara filen.
7. Gör fler loggningar eller stoppa exekveringen enligt ovan.
8. Öppna och kör de sparade filerna i MATLAB och analysera dem.

Observera att endast variabler som ändrats under den tid då loggningen utförts kommer att sparas. Detta gör att antalet variabler i den sparade filen kan variera.

3. CSTR-processen

Alla experiment kommer att utföras på Reglertekniks labprocesser. Det finns sex 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. Processen slås på och av med strömbrytaren på dess sida. Kontrollera vätskenivån i vattenkaret; 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 en signal som skickas till processen går emot någon av de interna förreglingarna:



Figur 3 Blockdiagram över systemet och dess signaler. Signalernas benämningar är de samma som används i JGrafchart-mallen.

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

När laborationssalen lämnas ska processen stängas av med strömbrytaren på dess sida.

Systemets signaler

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

1. Vätsketemperatur, `Temp`.
2. Vätskenivå, `Level`.

Insignalerna är spänningarna över inpump, utpump, omrörare och värmare. Eftersom vi i det tänkta scenariot inte kan påverka utflödet ska det ses som en störning som verkar på processen. Följande tre insignaler återstår för regulatorerna att använda:

1. Spänning över inpump, `InPump`.
2. Spänning över värmare, `Heat`.
3. Spänning över omrörare, `Agitator`.

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[1].

Den sensor som mäter temperaturen i tanken har en bias som måste korrigeras genom kalibrering.

[1]Eftersom värmaren har svag effekt anger temperaturmätaren ett högre värde än det faktiska. På så vis simuleras en högre värmareffekt än vad som egentligen är fallet.

UPPGIFT 1
Kalibrera temperatursensorn genom att att ändra `TempCalibration` så att `Temp` visar ungefär 20°C. □

UPPGIFT 2
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 $\mathrm{d}V/\mathrm{d}t$ i tanken är lika med differensen mellan inflöde q_{in} och utflöde q_{ut}, vilket beskrivs av massbalansekvationen

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

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 3
Ta fram överföringsfunktionerna från både in- och utpump till vätskenivån i tanken, h. Ange uttryck för förstärkningen från inpump till nivåändring (κ_1) och från utpump till nivåändring (κ_2). □

Både in- och utpump har en tröskelspänning, det vill säga lägsta insignalvärden för vilka de pumpar igenom vätska. På grund av pumparnas konstruktion kan tröskelspänningarna variera något beroende på om det redan flödar vätska genom pumpen eller inte.

En pumps tröskelspänning kan exempelvis bestämmas genom ett enkelt experiment. Sätt insignalen till pumpen så att den ger ett flöde och minska sedan insignalen i steg tills det att flödet upphör. Det värde på insignalen för vilket flödet upphörde motsvarar tröskelspänningen. Försöket kan sedan upprepas med mindre steg i insignalen för att få högre noggrannhet.

UPPGIFT 4
Utför experiment på både in- och utpump och bestäm därigenom de båda pumparnas tröskelspänningar. För att göra detta, exekvera programmet och ange önskade värden (0–100%) genom att ändra `InPump` respektive `OutPump`.

Senare i projektet ska utpumpsspänningen i huvudsteget variera mellan att vara 5 och 10 procentenheter högre än dess tröskelspänning. Dessa nivåer benämns u_{ut}^- respektive u_{ut}^+, dvs

$$\begin{aligned} \texttt{u\_out\_minus} &= \texttt{u\_out\_threshold + 5} \\ \texttt{u\_out\_plus} &= \texttt{u\_out\_threshold + 10} \end{aligned} \tag{2}$$

Ange värden för dessa parametrar i mallens huvudfönster. □

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 3 användas. Det

är därför viktigt att modellparametrarna bestäms för den arbetspunkt som ska användas.

Om pumparna har stått stilla en längre tid eller om man råkat köra luft igenom dem kan pumparnas effekt initialt vara lägre än om de varit i kontinuerlig drift.

Den enklaste metoden för att bestämma förstärkningen är att utgå från differentialekvationen som beskriver höjden i tanken och som användes för att härleda överföringsfunktionen i uppgift 3. Genom att integrera både sidor av differentialekvationen från en tidpukt t_0 till en annan tidpunkt t_1 erhålls

$$h(t_1) - h(t_0) = \kappa_1 \int_{t_0}^{t_1} u_{in}(t)\mathrm{d}t - \kappa_2 \int_{t_0}^{t_1} u_{ut}(t)\mathrm{d}t. \tag{3}$$

UPPGIFT 5
Beskriv ett experimentförfarande som undviker pumparnas olinjära dynamik och som utnyttjar (3) för att bestämma κ_1 och κ_2. □

UPPGIFT 6
Bestäm lämpliga värden på insignalerna u_{in} och u_{ut}, utför de försök som ni föreslagit i föregående uppgift och bestäm κ_1 och κ_2. Verfiera med experiment att parametrarna inte skiljer sig nämnvärt för andra insignaler nära arbetspunkterna. Utnyttja möjligheterna att logga och analysera datan i MATLAB samt för att generera grafer till er rapport. □

5. Vätskenivåreglering

Från ekvation (1) ser vi att nivådynamiken kan betraktas som integrerande. En metod för design av PI-regulatorer för integrerande processer är *arresttidstrimning*, se appendix B.

UPPGIFT 7
Använd arresttidstrimning för att ta fram lämpliga regulatorparametrar för nivåstyrningen. Använd en arresttid $T_a = 2$ s. □

UPPGIFT 8
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 ställdonen skadas krävs att utsignalen mättas av regulatorn. Därtill måste integratoruppvridning motverkas i de fall då en I-del används.

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 9
Implementera PI-reglering för vätskenivån i JGrafchart. Använd ett referensvärde på 40%. Uttrycket för att uppdatera I-delen blir vid användning av diskretiseringmetoden ”framåt-Euler”:

```
Ipart = Ipart + K*h/Ti*(Lref-Level) + h/Tt*(u-v1)
```

Notera att uttrycket innehåller ett uppvridningsskydd för integratorn. Använd tumregeln $T_t = 0.5T_i$ för att undvika att få ytterligare en parameter att ställa in. Styrsignalen ska begränsas så att den hela tiden ligger mellan inpumpens tröskelspänning och 100%.

För att implementera nivåstyrningen:

1. Öppna `LevelControl`.
2. Sätt `umin` till ett värde som ligger strax under tröskelspänningen för inpumpen. Det är viktigt att inpumpen inte pumpar vatten när dess styrsignal är `umin`.
3. Ange önskade värden för K, T_i, T_t och referensvärdet L_{ref}.
4. Öppna modulen `PI` som finns i `LevelControl`.
5. 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 T_a och därmed även regulatorparametrarna tills det att en stegändring i utpumpen från u_{ut}^- till u_{ut}^+ inte ger en nivåavvikelse på mer än 2% av tankens fulla höjd. Observera att T_a endast är en designparameter som används för att ta fram regulatorparametrar utifrån modellen; den verkliga arresttiden kan därför avvika från T_a beroende på att modellen inte är fullständig. Hur väl stämmer teori med verklighet beträffande värdet på T_a? □

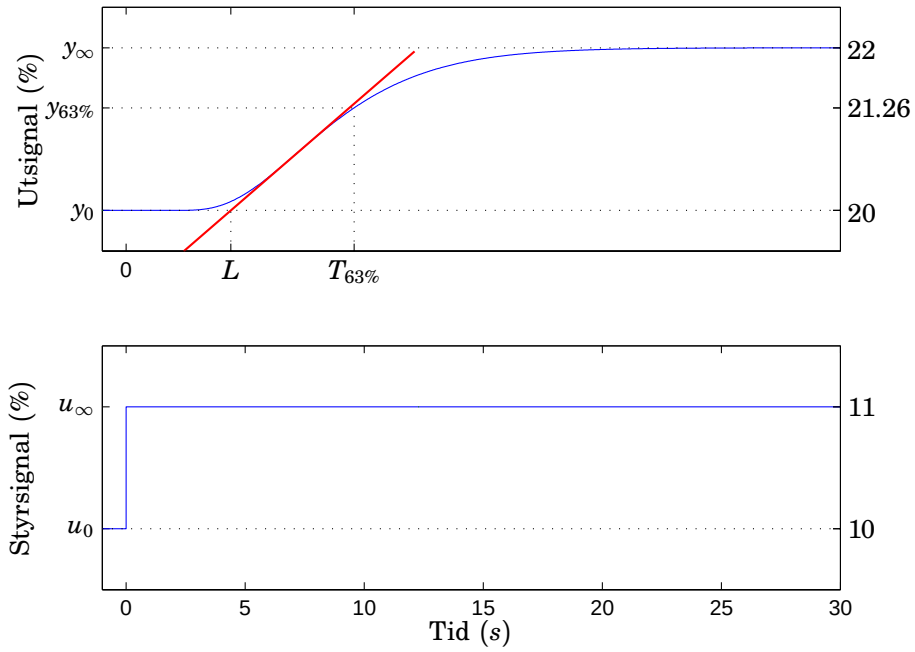
6. Modellering av temperaturdynamiken

Liksom i fallet med nivåregleringen i förgående kapitel ska vi utgå från en modell när vi reglerar temperaturen i reaktorn. En av de stora fördelarna med återkoppling är att vi ofta kan upppnå god prestanda även om vi baserar vår regulatordesign på en approximativ modell. I fallet med nivåreglering kunde vi härleda en enkel modell genom att ställa upp en massbalans. Här ska vi istället hitta en enkel modell genom att studera hur systemet reagerar på en stegändring i insignalen.

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

$$G_p(s) = \frac{K_p}{sT + 1} e^{-sL} \tag{4}$$

För att bestämma en modell av typen (4) för en process med okänd dynamik kan ett stegsvarsexperiment utföras. Detta görs genom att lägga processens utsignal i jämvikt nära den arbetspunkt den ska köras vid. Detta eftersom en olinjär process kan ha varierande beteende vid olika arbetspunkter. Sedan görs ett steg



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

i insignalen. I figur 4 ges ett exempel på ett sådant försök, där den önskade arbetspunkten ligger vid 20%. I figuren har ett steg på 1% gjorts i styrsignalen, vilket ger upphov till förändringen av processens utsignal.

För att hitta de tre processparametrarna i (4) utifrån ett stegsvarsexperiment kan följande procedur tillämpas:

1. Bestäm förstärkningen K_p genom att dela förändringen i utsignal, Δy, med förändringen i styrsignal, Δu efter det transienta förloppet,

$$K_p = \frac{\Delta y}{\Delta u} = \frac{y_\infty - y_0}{u_\infty - u_0} \tag{5}$$

2. Dra en tangent till utsignalen där dess lutning är som störst, motsvarande den röda 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, y_0.
4. Bestäm den tid, $T_{63\%}$, för vilken förändringen i y har nått 63% av det stationära värdet dvs $y(T_{63\%}) = y_0 + 0.63\,(y_\infty - y_0)$.
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:

$$\begin{aligned} K_p &= \frac{2\%}{1\%} = 2 \\ L &= 4.40 - 0 = 4.40 \\ T &= 9.58 - 4.4 = 5.18 \end{aligned}$$

Detta gör att processmodellen i exemplet kan defineras till

$$G_p(s) = \frac{2}{5.18s + 1} e^{-4.4s}. \tag{6}$$

UPPGIFT 10
Utför stegsvarsexperiment enligt beskrivningen ovan och beräkna samtliga modellparametrar i (4). Insignalen till värmaren är `Heat` (0–100%). Säkerställ att det finns ett kontinuerligt flöde genom reaktorn. Detta kan göras genom att sätta utpumpens insignal till `u_out_minus` och aktivera nivåregleringen från föregående uppgift. För att få en jämn uppvärmning av vätskan måste omröraren vara påslagen. Den slås på och av genom att parametern `AgitatorControlOn` sätts till 1 respektive 0.

Stegsvaren bör göras omkring 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. Sätt `Heat` till $0.8u_0$ och låt temperaturen stabilisera sig.
3. Öka `Heat` till $1.2u_0$ och låt temperaturen stabilisera sig. Detta ger ett temperatursteg omkring 37°C som kan användas för modelleringen.
4. Vid behov kan steg 2-4 upprepas med större steg.

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

7. Temperaturreglering

En vanlig metod för val av regulatorparametrar till PI-regulatorer i processindustrin kallas för λ-metoden (lambda-metoden). Den baseras på att processen kan beskrivas med en första ordningens modell med tidsfördröjning, se ekv. (4). λ-metoden beskrivs i föreläsning 9 och kapitel 8 i *Process Control*.

UPPGIFT 11
Använd λ-metoden för att ta fram lämpliga regulatorparametrar för temperaturstyrningen utifrån er modell. Använd tidskonstanten $\lambda = T$ för det slutna systemet. □

UPPGIFT 12
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 10. Justera om nödvändigt λ och därmed även regulatorparametrarna om styrningen ger upphov till oscillationer i temperaturen eller om den upplevs som för långsam.

För att implementera temperaturstyrningen, gör på motsvarande sätt som i uppgift 9 men använd den arbetsyta som benämns `TempControl`. □

UPPGIFT 13
Gör en stegstörning med 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

Om man kan mäta den signal som verkar som störning på processen kan vätskenivåregleringen förbättras genom framkoppling. I det enklaste utförandet används en så kallad statisk framkoppling. Det innebär att man till styrsignalen adderar en term som är proportionell mot störningen.

UPPGIFT 14
Utgå ifrån överföringsfunktionen från uppgift 3 och utforma en statisk framkoppling från utpumpspänning till inpumpspänning så att inverkan på vätskenivån från en störning i utpumpspänningen elimineras. 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? □

Den statiska framkopplingen kan implementeras genom att i `LevelControl` ange ett värde för framkopplingens förstärkning `FF`. Kopiera PI-regulatorn för nivåreglering till modulen `PIandFF` och lägg där till en framkopplingsterm. På detta sätt går det enkelt att slå på och av framkopplingen med huvudfönstrets knapp **Level Feedforward On**.

UPPGIFT 15
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? Utnyttja möjligheterna att logga data och sedan analysera den i MATLAB för att generera grafer till er rapport.

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. 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.

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 ska användas. Omröraren sätts igång genom att sätta `AgitatorControlOn` till 1. Observera att det finns ett inbyggt skydd som hindrar omröraren och värmaren att köra om variabeln `LevelLow` i arbetsytan `LevelIndicator` är 1.

UPPGIFT 16
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. □

Huvudsteg

Huvudsteget i processen benämns `Run` i start.xml, och det är viktigt att det inte döps om. Detta steg fortgår tills `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 utflödet variera enligt ett förbestämt mönster för att simulera en varierande efterfrågan i nästa processteg. Scenariot startas i och med att variablen `OutFlowOn` ges värdet 1. Det initiala utflödet är u_{ut}^{-}, och en tid in i detta steg ändras utflödet till u_{ut}^{+}.

UPPGIFT 17
Aktivera regleringen av nivå och temperatur i processens huvudsteg. Observera att omröraren fortsatt ska vara igång under huvudsteget. □

Avslutningssteg

När huvudsteget är avslutat ska all reglering samt utflödet stängas av och tanken tömmas för att sedan rengöras. I detta steg kommer vi att behöva utöka strukturen i `LevelIndicator` för att även kunna avgöra om tanken är tom respektive full. För det ändamålet behövs en nivåindikator.

Nivåindikator. 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 avslutningssteget.

UPPGIFT 18
Gå in i arbetsytan `LevelIndikator`. 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.
- `LevelLow` ska fungera på samma sätt som tidigare. □

Tömning och rengöring. När nivåindikatorn är implementerad kan vi fortsätta färdigställandet av styrsekvensen. Tömningen av tanken ska göras genom att sätta `OutPump` till ett lämpligt värde, till exempel `F`. När tanken är tom (`LevelIndicator.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 rengörningsproceduren implementeras enligt följande ordning.

1. Tanken ska först fyllas. För att avgöra om tanken är full ska `LevelIndicator.Full` användas.
2. Tanken ska sedan tömmas. Använd `LevelIndicator.Empty` för att avgöra om tanken är tom.
3. Tanken är nu rengjord och ska återgå till sitt utgångsläge.

UPPGIFT 19
Utforma processens avslutningssteg enligt ovanstående specifikationer. □

10. Testning av styrsekvensen

Alla processteg är nu färdiga för testning.

UPPGIFT 20
Starta styrsekvensen 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, allt från industriprocesser till vardagliga apparater som tvättmaskiner.

UPPGIFT 21
Reflektera över det arbete som utförts i detta projekt. Diskutera modelleringen, regleringen och sekvensstyrningen; vad har fungerat bra eller mindre bra och hur skulle de olika delarna kunna förbättras? Vad har ni lärt er under projektets gång, både övergripande och vad gäller detaljer i regleringen? □

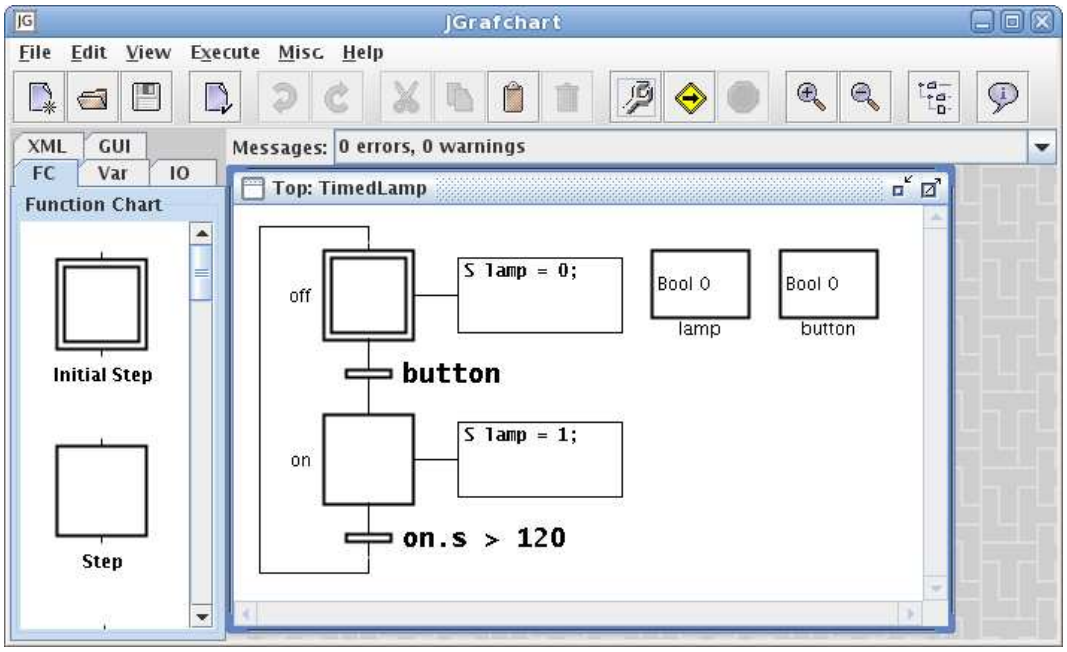
UPPGIFT 22
Hur kan projektet förbättras så att det blir mer lärorikt för framtida kursdeltagare? □

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.

Exempel

Ett exempel på ett enkelt JGrafchart-program visas i figur 5. Programmet går ut på att en lampa ska vara påslagen i två minuter efter att en knapp tryckts på. De booleska variablerna `lamp` och `button` innehåller information om ifall lampan



Figur 5 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.

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.

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.

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.

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.

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.

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>()

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.

Byggblock i JGrafchart

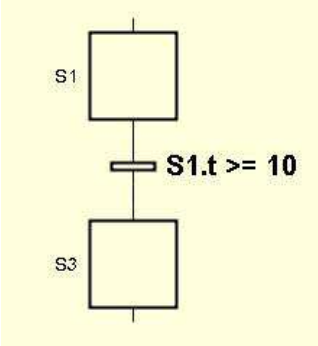
Steg. Stegets namn visas på dess vänstra sida, de åtgärder som sker i steget visas till höger om dessa. Genom att högerklicka på ett steg kan man dölja/visa de åtgärder (*actions*) som hör till steget, ändra åtgärderna (*edit*) samt ändra stegets namn.

Obs! En åtgärd måste alltid avslutas med ett semikolon.

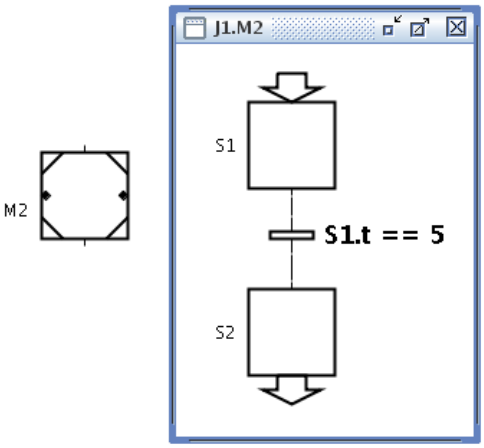
Initialsteg. Initialsteg är de steg som aktiveras när exekveringen av ett funktionsdiagram startas.

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 7. 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 6 En övergång



Figur 7 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.

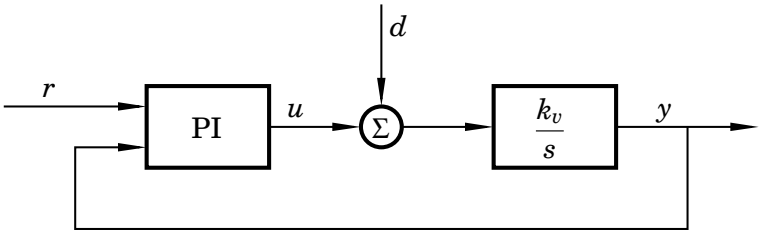
Åtgärdsknappar. En åtgärdsknapp (*action button*) utför en bestämd åtgärd när man klickar på den.

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.

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 8 Blockdiagram för PI-reglering av en integrerande process.

B. Arresttidstrimning

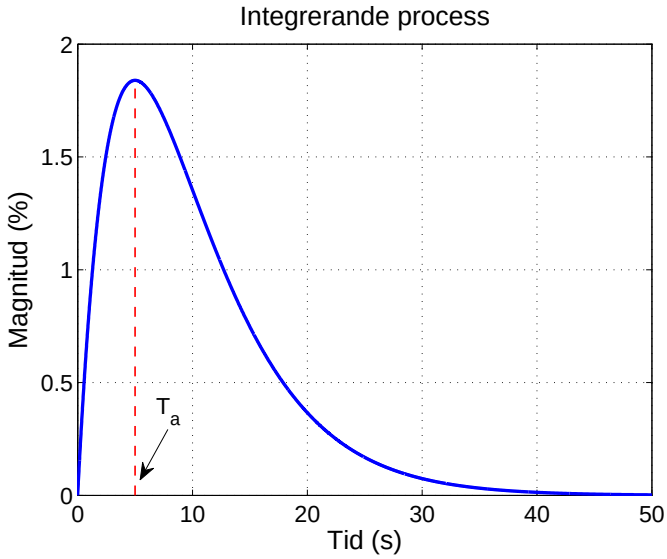
Arresttidstrimning är en enkel designmetod för PI-regulatorer, anpassad för begränsning av laststörningar i processer där överföringsfunktionen från insignal till mätsignal kan betraktas som en ren integrator:

$$Y(s) = \frac{k_v}{s} U(s) \tag{7}$$

Reglering av en integrerande process illustreras i figur 8. 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 9. De värden på regulatorparametrarna som krävs för att ett önskat värde på T_a ska ges beräknas enligt

$$K = \frac{2}{k_v T_a}, \tag{8}$$

$$T_i = 2T_a. \tag{9}$$



Figur 9 Arresttiden T_a är den tid det tar innan utsignalvärdet efter en laststörning på processingången till en integrerande process vänder tillbaka mot börvärdet igen.