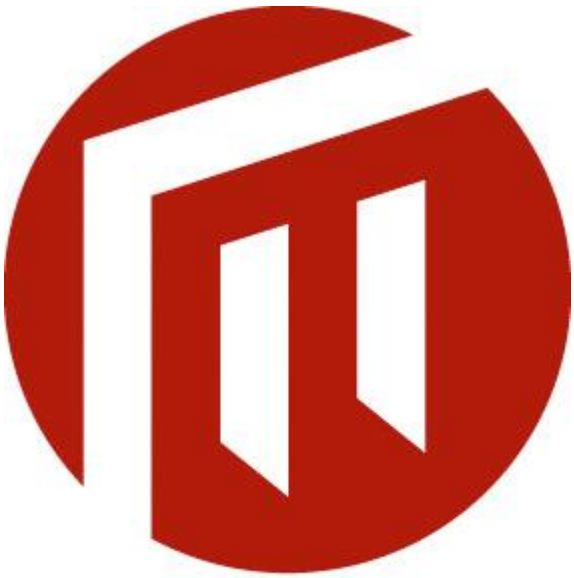


MALMÖ HÖGSKOLA

Inbyggda system och signaler

Teknisk rapport för
Examinationsprojekt

Prince Stevie-Ray Charles Balabis
Dator-ID: AB7271

Innehållsförteckning

Innehållsförteckning 1
1. Inledning 2
2. Syfte 2
3. Ansvar 2
4. Systemdelar 2
4.1 Distanssensor 2
4.2 Fläktmotor 3
4.3 Motor Shield 3
4.4 Digital signalbehandling 3
4.5 Reglering 4
4.6 UART/Matlab 6
4.7 RTOS 6
5. Kopplingsschema 7
6. Utföring 8
P-kalibrering 8
Uträkning av periodtiden 9
Uträkning av I- och D-konstanterna 9
7. Resultat 10
8. Diskussion 12
9. Källförteckning 13

1. Inledning

Vi har fått i uppgift att med hjälp av en distans sensor, och en fläktmotor, kunna reglera en pingisboll i ett bestämt avstånd från distanssensorn. Avståndet kommer bestämmas med hjälp av ett Matlab-skript. Som skickar bland annat avståndet(eller börvärdet) till pingpongsystemet. Matlab kommer även att kunna ta emot värden i realtid, och grafiskt skriva ut viktiga variabler, för kalibrering och felsökningssyften.

All kod kan hittas på min Github Repo. [4]

2. Syfte

Syftet med projektet är att kunna implementera ett autonomt system med en typisk sensor och motor för att kunna framgångsrikt reglera ett objekt, och på så sätt visa kunskap av teori som vi fått under kursens gång inom signalbehandling, filterdesign och reglerteknik.

3. Ansvar

Våra ansvar skiftade vid ett tillfälle pga interna orsaker när vi först fastnade med PID-regleringen. Ansvar som står på listan nedan är de jag listat som jag har med lycka implementerat i projektet, antingen av helt egen framföring, eller som begäran av hjälp utav partnern som tagit ansvar för ett område, som jag sedan byggt om och/eller omstrukturerat.

- Moving average filter
- PWM
- Koppling och implementering av Motor shielden
- Koppling och implementering av distanssensorn
- Koppling och implementering av nätaggregatet till motor shielden
- ADC initialisering och läsfunktion av distanssensorn
- UART kommunikation(fram och tillbaka) mellan Arduino och Matlab på en PC
- RTOS(trådar, samplingstid, semaforer etc)
- Implementering och kalibrering av PID-reglering

4. Systemdelar

4.1 Distanssensor

För att läsa var bollens position är på ett slutet plan, används en distanssensor. Distanssensorn är Sharp GP2Y0A21YK0F. Sensorn är en komplett modul som endast kräver tre kopplingar, grund(0v), 5v(4.5v-5.5v) för driftspänning och till slut en kabel kopplad till en analog till digital konverterare(ADC), för att läsa sensorns värden. Sensorn drar ström i stora, korta salvor, därför rekommenderas att man kopplar minst en 10 µF kondensator mellan 5v och GND för att säkerställa att sensorn fungerar korrekt. [1]. Modulen klarar av att läsa avstånd mellan 10-80cm framför sensorn. Ju med att det slutna planet har intervallet 0-50cm i längd, kommer endast mätningar mellan 10-50cm av det slutna planet vara korrekt. Det slutna planet har längdmarkeringar varje 10 cm(10cm,20cm,30cm,40cm,50cm) som används som mål för regleringen, därför lär det inte vara behöva ske noggranna mätningar under 10cm eller över 50cm då det är inte ett krav för funktionen av systemet.

Arduino Due har ett komplett analog till digital konverter(ADC) som vi använder för att läsa sensorns värden. ADC klockan är satt på 20MHz, vilket är det högsta möjliga klockfrekvensen för ADCn för Arduino Due, och borde ge oss bästa möjliga prestanda och förbättra regleringen. Upplösningen kan gå upp till 12bitar, vilket ger oss värden mellan 0 till 4095, mycket mer rum än 10 bitar som ligger på 1024. Men för lättare programmering bestämde vi oss för 10 bit ADC(0 till 1023). ADC kanal 10 på SAM3X8E går till pin 'A8' på Duen[2], det är pinnen vi använder för att läsa sensor-värdena.

4.2 Fläktmotor

För att kontrollera vilken position pingisbollen har på det slutna planen, används en fläktmotor. Motorn körs på 12v, men man kan enkelt reglera spänningen mellan 0v-12v för att bestämma rotationshastigheten på fläkten, som sedan bestämmer luftflödet ut från motorn. Desto högre rotationshastighet, desto närmare lär pingisbollens position är till sensorn vara. Då vi vill programmerbart kunna kontrollera fläktmotorn, använder vi Arduino Motor Shi eld R3, som är en dubbel kanalig H-brygga som lätt monteras fast på Duen. Läs mer om den på nästa avsnitt.

4.3 Motor Shield

Arduino Motor Shield R3, är en dubbel kanalig H-brygga som lätt monteras fast på Duen. Eftersom det endast är en fläktmotor vi vill kontrollera, använder vi bara en av H-bryggorna(Kanal A). H-bryggan är egentligen inte nödvändig, då H-bryggor är till för att kunna vända på rotationen hos motorer. Eftersom vi vill att vår fläktmotorn endast blåser ut och inte in, sätter vi pin 12 till hög under hela programexekveringen. Med shielden kan vi alltså styra rotationshastigheten på motorn med hjälp av PWM. Shielden tar emot en 0v-3.3v PWM signal och reglerar spänningen från en 12v nätadapter till att ge ut en spänning mellan 0-12v enligt PWM-signalen. Vår PWM har en duty cycle som kan sättas mellan 0-100%.

4.4 Digital signalbehandling

Vi använder oss av antingen ett FIR filter eller ett glidande medelvärde filter för att jämna ut värdena från distanssensorn, båda existerar för testningens skull, men antingen eller kan användas då de av snabba tester visar sig ge liknande resultat. Vi använder oss av på så sätt en av filtrarna åt gången för att ge slätare resultat av PID-regleringen, men inte båda då filter-buffrarna lägger till en fördröjning mellan sensorn och PID-regulator-koden som är möjligtvis prestanda-nedsättande, samt att båda filtrarna försöker ge liknande resultat, vilket gör det till en viss del "overkill".

Efter signalens filtrering konverterar vi signalen till ett linjäriserat värde som sedan skickas vidare till PID-regulatorn. Signalen konverteras till centimeter, vilket fungerar för PID-regleringen och är lätt att plotta ut och förstå på Matlab.

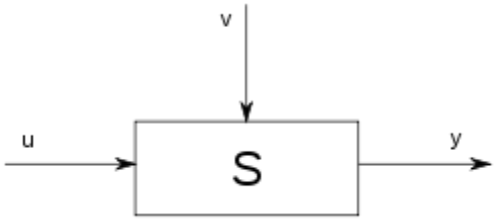
4.5 Reglering

Ett system som har styrsignaler, mätsignaler, störningar, och mätfel har reglerproblem. För att lösa detta kan man använda en regulator.

Regulatorn(S) på bilden nedan tar emot en insignal(u), från exempelvis en sensor.

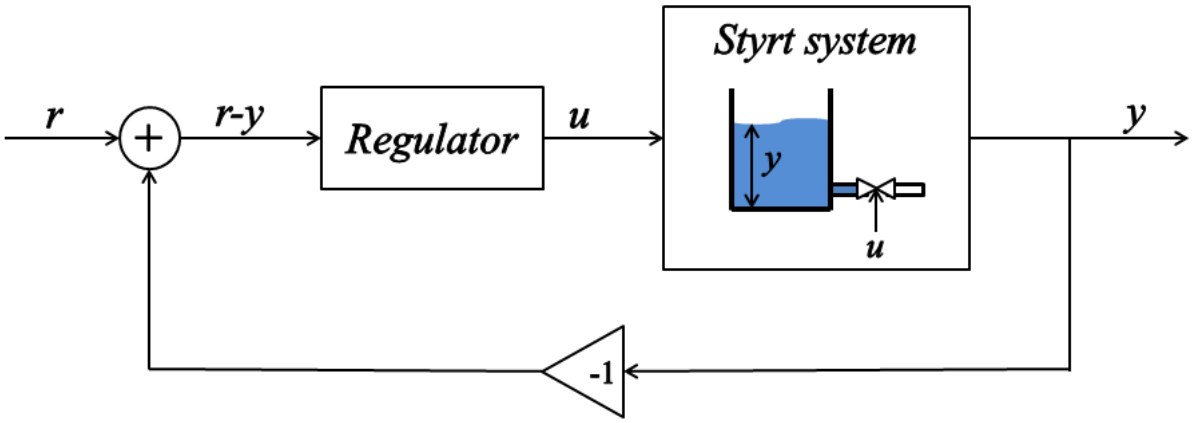
Regulatorn(S) har även ett börvärde(v). Som är ett värde man önskar regulatorn styrde systemet mot. Börvärdet kan vara den önskade insignal för systemet som man vill uppnå

Regulatorns jobb är att justera utsignalen(y) så att insignal (u) möter samma värde som börvärdet(v). Utsignalen(y) kan skickas till exempelvis en motor.



'Black box' princip för en regulator

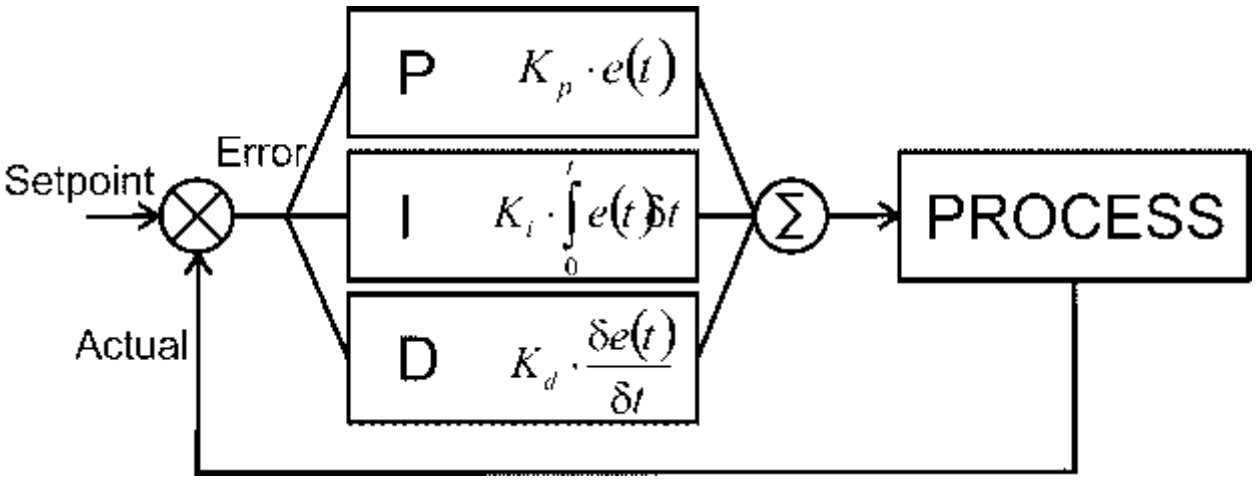
Här är ett exempel på en allmän reglerkrets kallad återkoppling. En vattentanks vattennivå regleras. 'r' står för börvärdet. Regulatorn tar emot r-y, där 'y' är ärvärdet av systemet. Formeln räknar ut felvärdet, vilket betyder att börvärdet(r) – ärvärdet(y) = felvärdet. Regulatorn tar emot det nya felvärdet, och räknar ut en ny utsignal(u)



Reglerkrets med vattentank som styrt system

Vi använder en PID-regulator i vårt system, för att reglera pingisbollen. PID reglering står för Proportionell–Integrerande-Deriverande reglering. Regleringen kan delas upp i tre separata element som kan varsin konfigureras med en vikt.

Var och en av de tre elementernas utsignaler adderas till en enda utsignal.



PID-regulator i en reglerkrets

Proportionell reglering, eller P-reglering, är en reglering där styrvärdet är proportionellt mot styrvärdet. Felvärdet multipliceras med en konstant, för att få ut utsignalen

Utsignal = P-konstant * Felvärdet

Att endast ha en P-reglering, orsakar en oscillation i systemet. Detta beror på att regleringen inte kan retardera insignalen innan felvärdet blir noll. Retarderingen och accelereringen tillbaka sker efter felvärdet noll har nåtts.

Integrerande reglering är en reglering där utsignalen är en integral av felvärdet. För denna typ av reglering mäter man hur lång tid det tar för I-regleringen att möta P-regleringen. Den här regleringen tillsammans med P-reglering löser problemet som P-regleringen har.

PI-reglering är blank de vanligaste, men för en bättre reglering, kan man även köra deriverande reglering. PID-reglering tillåter dig att ha större P och I konstanter och ändå hålla en stabil reglering, medan det ger dig snabbare reaktionstid och prestanda. Den här regleringen kräver ett filter, för att fungera bra.

PID-reglering ger den här formeln, där utsignalen av P, I och D adderas till en gemensam utsignal.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de}{dt}$$

Ett stort problem som uppstod med PID-regleringen är ett så kallad “windup reset”. Detta hör till I-delen av regleringen. När summan av felen adderas, kan summan bli så stor eller liten att det tar en stor mängd tid för summan att inverteras, när bollen passerar börvärdet.
Detta löstes genom att sätta en min och max värde för fel-summan. Då blev responstiden så mycket bättre.

4.6 UART/Matlab

För att sätta PID-variablerna, börvärde av PID-regulatorn samt hämta och plotta värden från Arduino behövs en kommunikation mellan Arduino Due och Matlab(dator). Vi använder oss av UART. UART kan enkelt implementeras till vårt Atmel projekt tack vare ASFs UART driver, och möjliggör därefter kommunikation mellan Due och PC genom en USB sladd, emulerad till en seriell port. Vid initalisering av kommunikationen mellan systemet och Matlab, måste Arduinon först vara igång och kopplad till datorn. Då är systemet i 'viloläge'. När Matlab-skriptet exekveras, medföljer parametrar som PID-variablerna och börvärde(för PID-reguleringen). UART kommunikationen utfärdas till systemet och dessa variabler skickas en efter en till systemet. För att se till att det inte sker några konflikter i synkronisering av läs och skrivningarna mellan systemet och Matlab, kommer systemet att fastna i loopar som ständigt kollar om RX-buffern är tom. När Matlab skickar exempelvis P-variabeln, kommer systemet att gå förbi loopen och exekvera läsning och sparning av P-variablen från UART. Denna process repeteras tills alla variabler är mottagna, och när de är, kommer PID-regleringen att starta(Läs mer på RTOS-avsnittet för att läsa om hur detta sker). För att Matlab ska kunna i realtid hämta värden, skickar Matlab en slumpmässig variabel till systemet. Systemet kommer hela tiden att kolla om den har tagit emot något i RX-buffern, när den har, skickar systemet felvärdet, ut-värdet av PID-regleringen, avståndet samt börvärdet.

4.7 RTOS

RTOS står för real-time operating system och används för att schemlägga processer i ett system. Med RTOS kan tidskänsliga processer utföras med större precision och både underlättar utveckling och debuggning av ett system. Vi använde FreeRTOS för vårt projekt, då det är gått med dokumentation, utveckling(då det är öppen källkod), och allra viktigast, kan enkelt integreras med ett Atmel-projekt, tack vare ASF Wizard, som är en grafisk kontrol-panel för att importera moduler som FreeRTOS, ASFs PWM driver eller ASFs ADC driver för ett projekt.

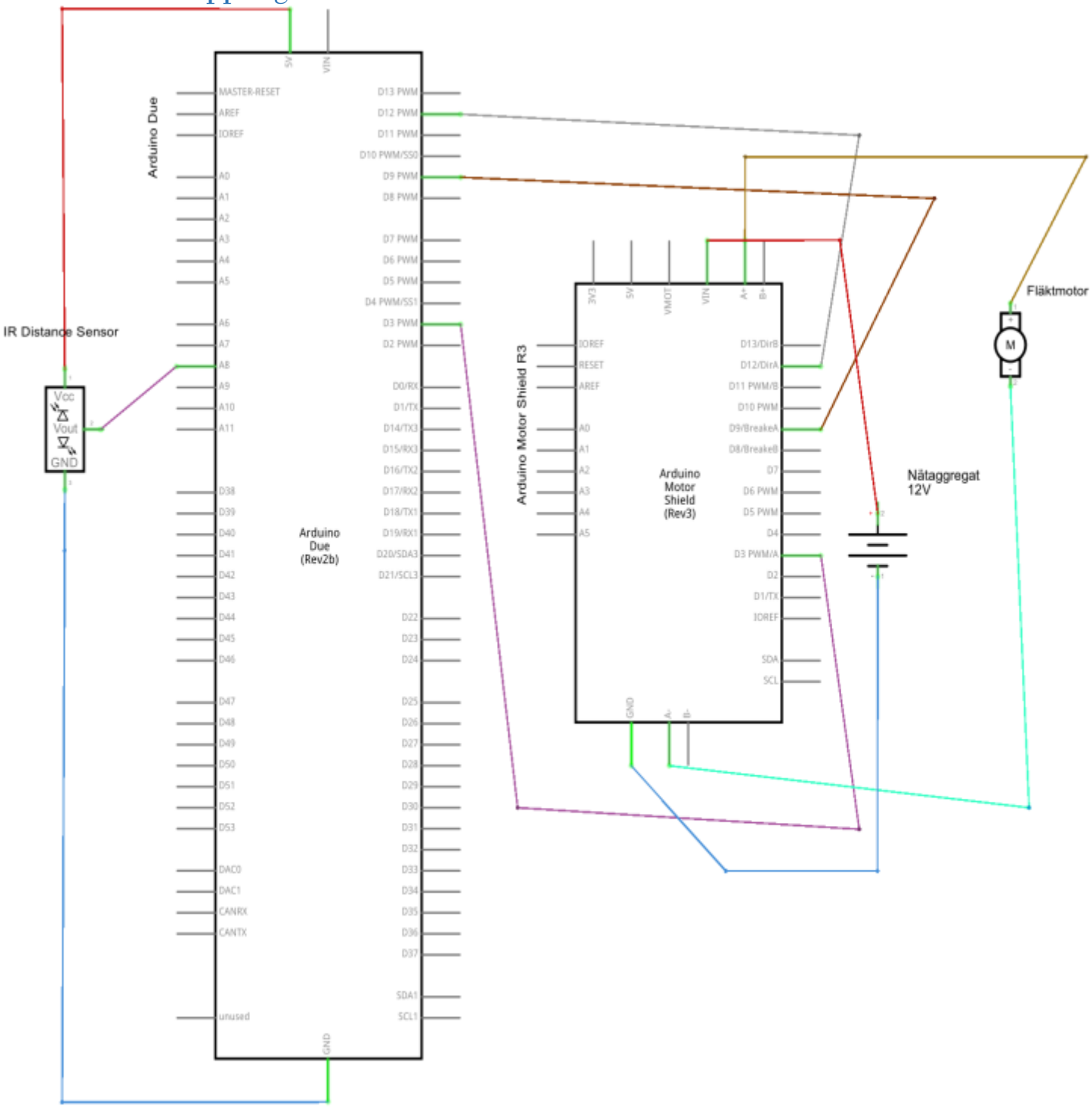
PID-regleringen och UART/Matlab kommunikationen har var sin tråd, detta för att säkerställa att PID-regleringen utförs i ett regelbundet intervall(50-100ms) och vice versa för UART/Matlab kommunikationen. PID-regleringen är mer tidskänslig än UART/Matlab kommunikationen, eftersom den direkt påverkar systemets praktiska reglerings-prestanda och har större uppdateringshastighet än UART/Matlab kommunikationen.
PID-regleringstråden har prioritet '2' medan UART/Matlab kommunikationen har prioritet '1'. Detta ger PID-regleringen en jämnare uppdateringsfrekvens, då PID-regleringens aktiviteter prioriteras högre än UART/Matlab kommunikationen.

Båda dessa trådar har en stack size på 2048 bytes vilket mer än räcker för de variabler som lagras och körs under deras gång. Detta innebär att det rymmer gott om variabler innan Dues maximala SRAM på 96kb är uppfylld. Totalt ska man kunna deklarera 128(16bit) ints innan stack sizes är fyllda för var sin tråd.

Vi har även implementerat en semafor som har till uppgift att signalera PID tasken att köras då Due kortet har mottagit PID variablerna. Detta görs så att Matlab hinner skicka alla PID variabler innan regleringen börjar. När dessa värden har mottagits släpps semaforen som PID tasken tar och PID – reglering startar.

PID tasken samplas med en periodicitet på 100ms, vilket bestäms genom en parameter som Matlab skickar. UART taskens samplingstid beror på vilket värde variabeln dT får, som även den sätts som en av Matlab programmets parametrar. Denna task är dock frusen tills den tar emot en etta från Matlab (skickas med en periodicitet av dT) och först då skriver den ut värden som ska plottas.

5. Kopplingsschema



fritzing

6. Utföring

Börvärdet är konstant över PID-kalibreringen och ligger vid ’30 cm’-märket, för att ge ett stort utrymme för oscilleringen att äga rum över rälsen. Varje test var 25-30 sekunder långa, vilket verkade vara lagom med tid för att observera konstanter och regleringar ur graferna. Grafen uppdateras med en samplingstid på 0,3s, vilket gav en relativt mjuk graf utan att sätta för mycket stress på Arduinons processor eller UART.

En annan punkt att tänka på är att sensorn inte ger ut en linjär spänning för distansens värde. Värdet ökar avsevärt desto närmar bollen når sensorn. Därför kan inte grafen eller värden som man samlat in korrekt hänvisa till distansen i verkliga världen. Sensorns ADC värden kan visa att den accelererat avsevärt långt från börvärdet men i den verkligheten har den endast passerat minimalt.

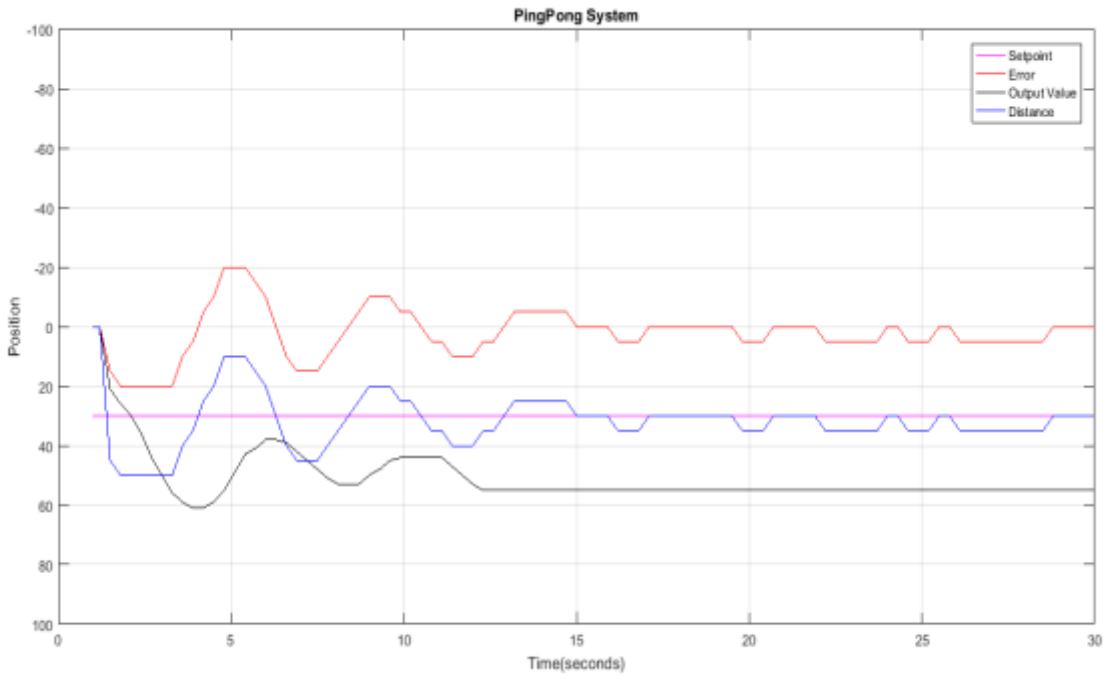
P-kalibrering

För P-kalibreringen är det viktigt att bollen har en självsvängning där topp till topp är tätt intill lika höga.

Jag började med en P-konstant på kP=0.05. kD och kI var båda noll. Då ville motorn knappt öka. Jag höjde sedan till 0.1 och då skedde en viss oscillering.

Som simpel regel för systemet lade jag att bollen får ej slå till på kanten vid sjunkning eller ökningen.

Matlab-funktionen som kördes:
startPID('COM4', 0.3, 30, 30, 0.1, 0.0, 0.0)



Oscillationen där kP=0.1, Ki=0, kD=0.

Uträkning av periodtiden

För att räkna ut periodtiden tittade jag på ett intervall där en periods topp-till-topp når liknande amplituder. Med dessa kriterier valde jag intervallet mellan 9-13.8 sekunder. Periodtiden(T0) blir då 4.8s

Uträkning av I- och D-konstanterna

T0 = 4.8 sekunder

K0 = 0.1

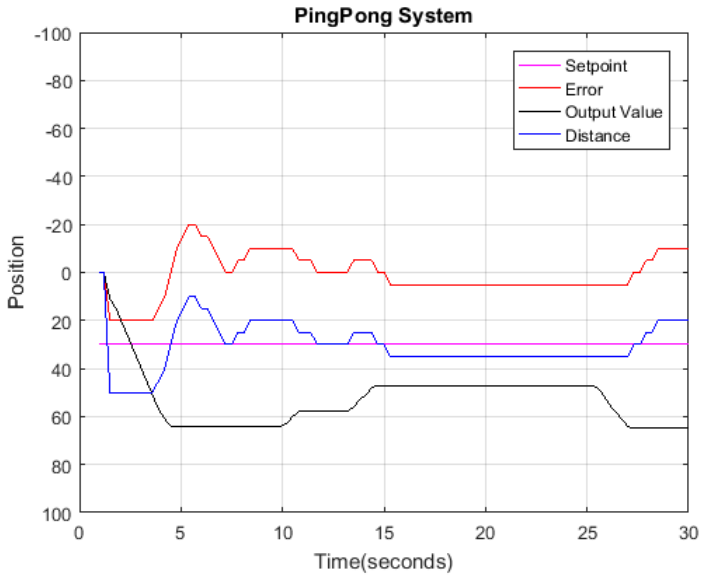
	Kc	Ti	Td
P	0.5*K0 = 0.05		
PI	0.4*K0=0.04	0.80*T0=3.84	
PID	0.6*K0=0.06	0.5*T0=2.7	0.125*T0=0.6

Ziegler Nichols kalibrerings guide och beräkningar[3]

Med hjälp av tabellen kunde vi räkna ut kI och kD. Vi får ut att kp=0.06, kI=2.7 och kD=0.6.

Matlab-funktionen med PID-konstanterna som kördes
startPID('COM13', 0.3, 30, 30, 0.06, 2.7, 0.6)

Video:
https://www.youtube.com/watch?v=YJSUNWXjdEs



Ziegler Nichols PID-kalibrering

7. Resultat

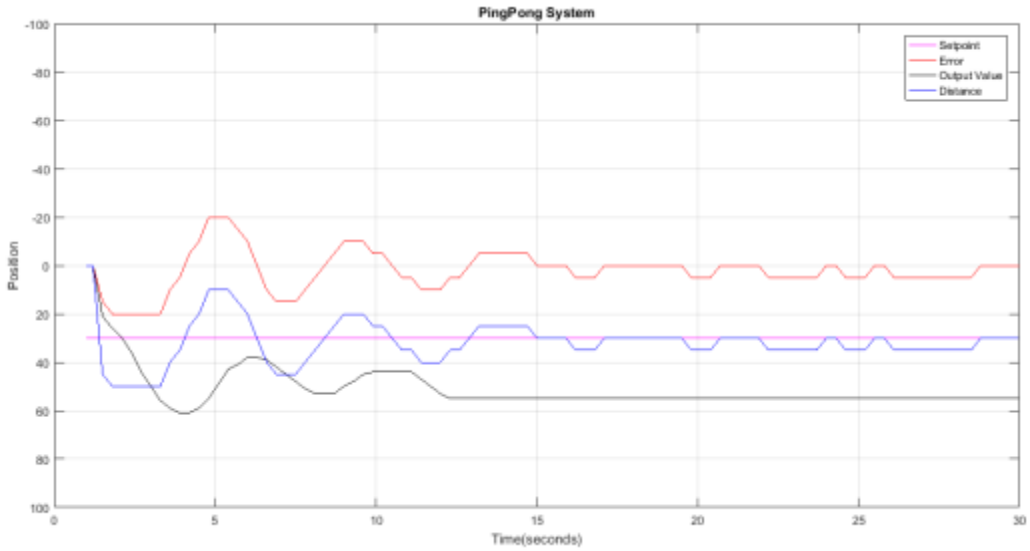
Nichols kalibrering gav oss värden kp=0.06, kI=2.7 och kD=0.6.

Ziegler Nichols-värdena gav en relativt okej reglering. Bollen oscillerar ovanpå börvärdet till en början men sedan stabiliserar. Från ca 27 sekunder in på testet(om man tittar på grafen ovan), kan man se att utsignalen av PID höjs avsevärt. Detta är ett driftproblem, och bollen till en viss del oscilleras ut ur börvärdet men hittar kort tillbaka. Ziegler Nichols är inte en komplett kalibrering men konstanterna kan användas som utgångspunkt för en bättre reglering. kI sänktes till kI=0.9 och det var redan en extrem förbättring, mindre oscillation och driftproblem. För att ge en snabbare respons på regleringen kunde vi höja kP till 0.1

Bästa värden efter justering av PID med Ziegler Nichols var kP=0.1, kI=0.9 och kD=0.6

Matlab-funktionen som kördes med mina slutgiltiga PID konstanter och börvärde 30cm:
startPID('COM4', 0.3, 30, 30, 0.1, 0.9, 0.6)

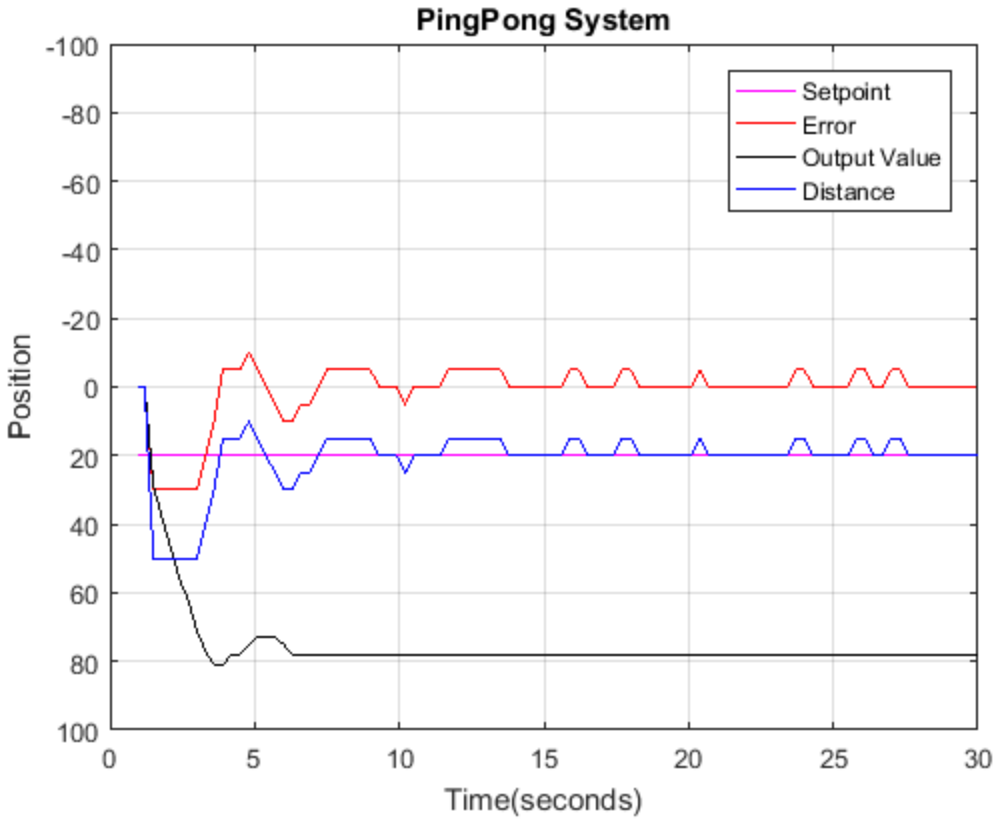
Video för börvärde 30cm:
https://www.youtube.com/watch?v=JQA9Zl80sYc



Börvärde 30cm

Matlab-funktionen som kördes med mina slutgiltiga PID konstanter och börvärde 20cm:
startPID('COM13', 0.3, 30, 20, 0.1, 0.9, 0.6)

Video för börvärde 20cm:
https://www.youtube.com/watch?v=d-fYXl6HN6Y



Börvärde: 20cm

8. Diskussion

Syftet med projektet är att visa kunskap av teori som jag fått under kursens gång inom signalbehandling, filterdesign och reglerteknik. Med en lyckad avläsning och förståelse av distanssensorerna, implementering av ”glidande medelvärde”-filter, en fungerande PID-reglering, samt massa mer, visar jag att jag har förståelse i vardera område.

Med denna simpel implementering och kalibrering, har jag kunnat uppnå en fungerande reglering. Lägger man in mer tid, kan man säkert uppnå en ännu bättre reglering.

Först när jag försökte implementera PID-reglering med systemet, trodde jag att det skulle fungera med råa olinjära sensor-värden. Jag gav alltså helt råa 10 eller 12 bitars sensor värden direkt till PID-regleringen. Detta var ett stort misstag, då PID-regleringens ”styrka” varierade stort beroende på bollens position i rälsen. Ifall pingisbollen är i botten av rälsen accelererar P-regleringen.

När jag löste detta hade jag dock fortfarande ett problem, som jag fastnade länge på. Detta var ”reset windup”. Reset windup orsakade att bollen oscillerade extremt stort över rälsen, och tog lång tid innan regleringen ville justera när bollen passerade börvärdet. Det var svårt att förstå vad felet var då man alltid bara antog att man hade implementerat PID-formeln fel. Det verkade som att vad man än satte I-konstanten på, ville inte det här ”reglerings-fördröjningen” försvinna. Det krävdes lite ”utanför boxen”-tänkande, där jag behövde inse att det är inte hur jag skrivit PID-formeln, utan om det integrerades rätt i mitt system. När jag hade det tankesättet och började undersöka vilka absurda värden I-regleringen gav, upptäckte jag att hela problemet kunde lösas med en minimum- och maximum-begränsning. Detta minskade eftersläpningen tillräckligt mycket så det nästan var omärkbart.

Ett av de största orsakerna till våra svårigheter med våra utföringar var generella buggar med Atmel Studio eller Atmel Software Framework. Då och då släpps uppdateringar till antingen Atmel Studio IDPn eller Atmel Software Framework. Varje ny uppdatering, ska ha kompatibilitet med gamla projekt och ASF-bibliotek, men detta betyder inte att det inte ingår buggar. Flera gånger behövdes projektet byggas om, då projektet antingen inte ville öppna, eller kompilera koden. Sådana problem, kan vara väldigt tidskrävande. Det enda sättet att undvika framtida problem med kompatibilitet, är att endast använda oss utav samma Atmel Studio- och ASF-version under hela utföringen.

Mitt implementerade filter var snabbt implementerat med en buffert på 5 läsningar. Jag testade aldrig om buffrets storlek gjorde någon stor skillnad på regleringen, så det är möjligt att förbättra regleringen genom att kalibrera filter-bufferten.

Samlingstiden för PID-regleringen ligger på 100ms. Detta satte jag så att vi har gott om tid att utföra våra beräkningar, och på så sätt inte behöva oroa oss för oregelbunden trådexekveringar. Med lite kodförbättringar och samplingstidstester, kan man säkert förbättra regleringen genom att lägga samplingstiden på 50ms, vilket är inom projekt-kraven.

9. Källförteckning

[1] https://www.pololu.com/product/136/specs

[2] http://www.robgray.com/temp/Due-pinout-WEB.png

[3] Ziegler_Nichols practical method från ”Lab 3c”

[4] Github Repo: https://github.com/PrinceBalabis/PingPongProject