Inbyggda system och signaler

Styr- och reglerteknik

Examinationsprojekt PingPong

Namn: Stefan Angelov

Gion Koch Svedberg

December 2015

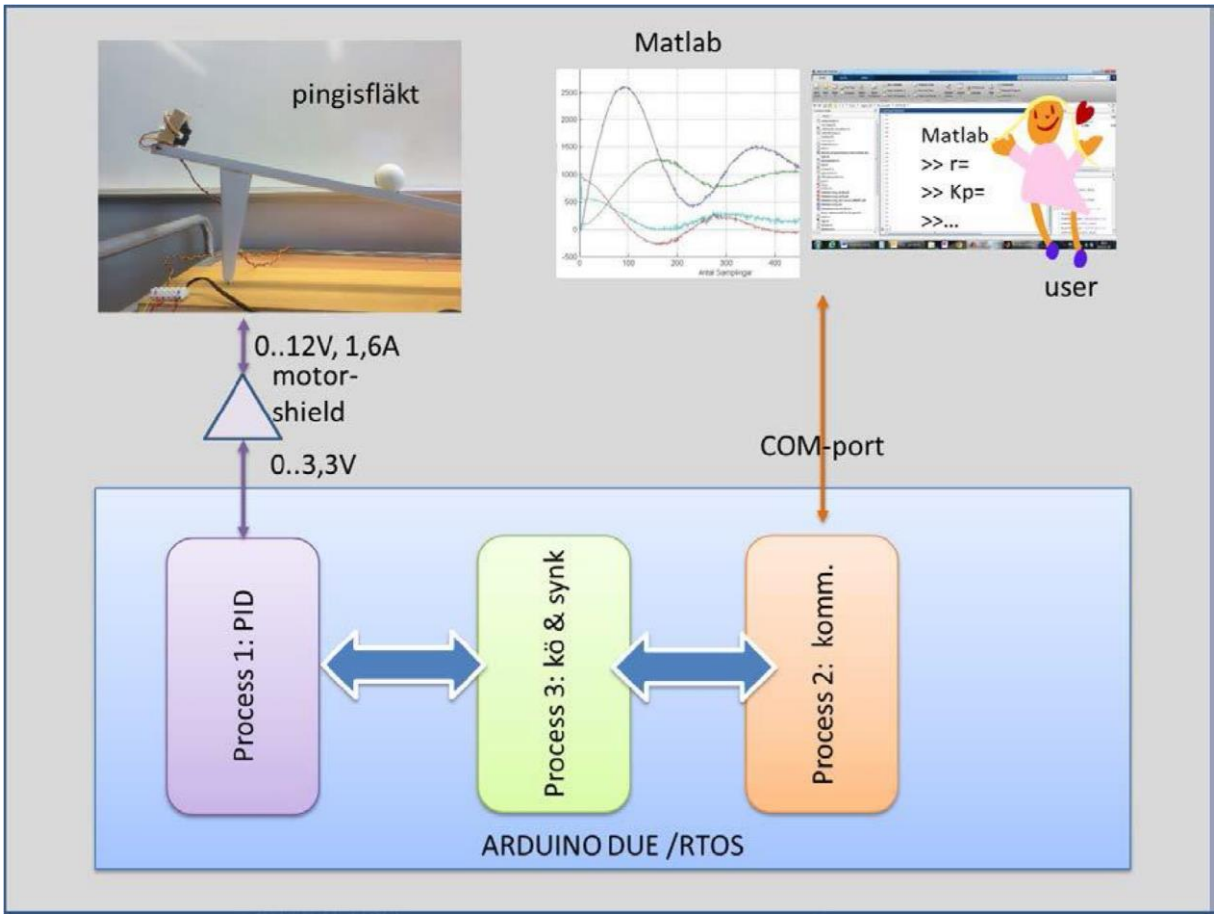
Innehållsförteckning

1. Inledning

Denna examinationsuppgift har till syfte att visa att eleven, jag, kan tillämpa olika regleralgoritmer på ett fysikaliskt system i praktik.

Systemet består av en distans sensor, som känner av pingisbollens avstånd på rälsen och en fläkt som blåser pingisbollen in i position. Dessa två mekanismer ska PID-regleras och för att uppnå ett slutresultat där pingisbollen håller sig i förvalt position måste man inkludera ett antal andra delsystem:

- De olika processerna ska schemaläggas i en RTOS.
- Läsa av värdet från distanssensorn genom ADCn på Arduino Due kortet.
- PWM styra fläkten genom PWM kanalen på Arduino Due kortet.
- Reglera systemet med hjälp av PID så att det önskade positionen på pingisbollen uppnås autonom.
- Uppnå tvåvägs kommunikation mellan Matlab och Arduino Due kortet.
- Koppla ihop de olika komponenterna, dvs. Arduino Due kortet, Motor Shield, powerboxen och systemet.



Figur 1: Representation av uppgiften [2]

2. Syfte

Syftet med denna rapport är att visa hur de olika problemen hos prototypen har lösts, samt integrera de olika delsystemen i ett fungerande helsystem.

3. Ansvar

Från början var planen att mitt ansvar skulle ligga i Motorshielden, kopplingen, regleringen och Matlab kommunikation.

Men eftersom vi hade problem med regleringen mot slutet, som jag kommer berätta om nedan under diskussion och slutsats, har jag gått igenom varje funktion och system del vi har implementerat.

4. Systemdelar

4.1. RTOS

RTOS som användes var FreeRTOS som kommer som en extension i Atmel Studio(libraryn finns även att ladda hem separat). Denna RTOS:en möjliggjorde schemaläggningen av de olika processerna så att de kunde köras simultant.

Vi har två trådar som körs under programmets gång, en som hanterar PID-regleringen(PIDTask.c, PIDTask.h) med högre, och en som hanterar kommunikationen mellan Matlab och Due kortet(UARTTask.c, UARTTask.h). Av de två, tasken som kör PID-regleringen har högre prioritet eftersom denna är mer tidskänslig än UART 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 mellan 50ms-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.

4.2. ADC

ADC kanalen som används är ADC_CHANNEL_10 som är A8 Pin på Arduino Due kortet. Den körs på den maximala frekvensen för ADCn vilken är 20MHz och upplösningen är 12bit alltså är resolutionen 0-4095.

4.3. Signalbehandling

För behandling av signalen har jag skrivit 2 funktioner, en ett FIR – filter och en moving average buffert. Det finns ingen märkvärdig skillnad på utvärdet när man jämför från båda filtrarna så vilken som helst kan användas.

4.4. PWM

Den interna PWMn på Due kortet användes. Denna fanns det lite problem med eftersom det fanns knappt någon information hur den används i samband med SAM3X8E, alltså processorn på Due kortet. Och eftersom det finns ingen intern PWM på PIN 3 som motorshielden använder som PWM

styrningen till kanal A som vi använder, fick jag ”jurry rigga” denna genom att koppla en sladd från PIN7 till PIN3. Pinnen från PIN3 på motorshielden är utdragen så den aldrig når Arduino Due kortet.

PWM Clock A är satt till 1Mhz och Master Clock till 84Mhz. För bättre upplösning sätter jag perioden till 999.

4.5. Motor Shield R3

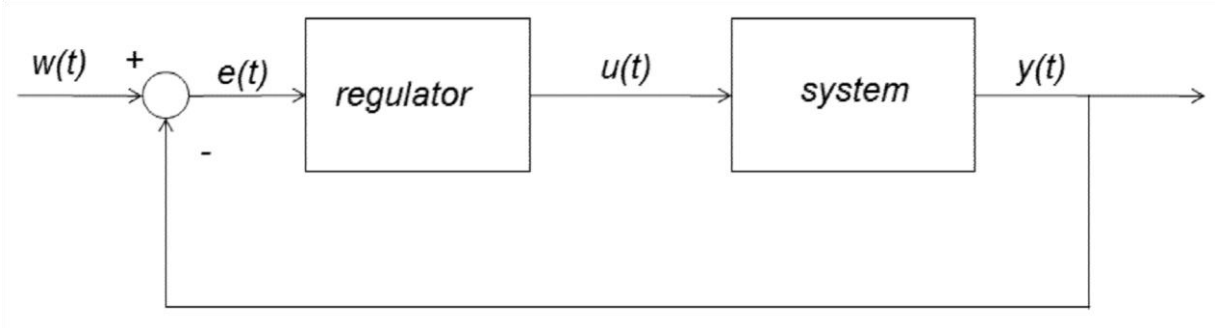
Eftersom Due kortet inte kan sänka 12V som behövs för att driva fläkten, använder jag ett Motor Shield R3[4] som jag fick låna av skolan för att åstadkomma detta. På denna använder jag kanal A som i sin tur använder PIN 12(Direction setting) och PIN 9(Brake) på Due kortet. Brake användes aldrig.

Skölden initieras av funktionen shieldInit().

Jag upptäckte även att Vin pinnen ska dras ut ur denna eftersom strömmen från USB porten överförs till shielden. Även om power boxen inte var igång så visade det den ca 3-4V på mätaren. För säkerhetsskäl drogs denna ut åt sidan.

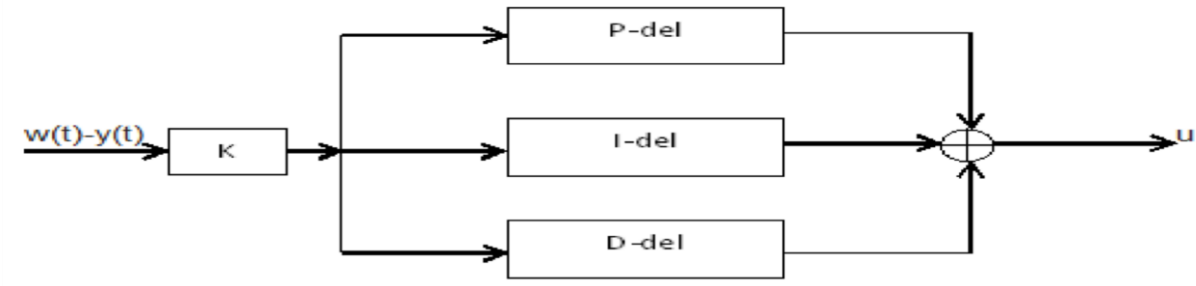
4.6. Reglering

Reglerteknik är en lära som man använder när man vill göra ett tekniskt system autonom, så att den själv uppfyller kraven användaren ställer på den. Den gör så genom att kontrollera en eller flera storheter och att reglera den kräver en regulator[1]. Där i sker regleringen, som nedanstående blockdiagrammet visar. w(t) står för börvärdet som kommer igenom regulatorn och in i systemet, för att sedan kommer ut som ärvärdet som är y(t) på blockdiagrammet. Detta värde återkommer in och jämförs med börvärdet som skillnad som ger felvärdet märkt e(t) = w(t) – y(t) på blockdiagrammet. Regulatorn ger ut u(t) vilket är styrsignalen.



Figur 2: Blockdiagram av reglerkrets[3]

Det som skulle regleras i denna uppgift är pingisbollens position, och efter att ha kommit fram till slutsatsen att PID-regulator fungerade bäst i en av tidigare labbar så valdes den även här. Ett blockdiagram på hur en PID-regulator, används mest i industrin[2], ser ut kan ses nedan:



Figur 3: Blockdiagram för en PID-regulator

4.6.1. P-Reglering

P-reglering där styrvärdet är proportionell mot (e) alltså felvärdet. Denna justeras genom att man multiplicerar den med konstanten Kp, som är den proportionella förstärkningskonstant.
$U = K_p * e$, $U = styrsignal$; $K_p = proportionella\ förstärkningskonstant$; $e = felvärdet$
När den proportionella förstärkningskonstanten ökar så ökar även snabbheten, men stabiliteten minskar. Enligt teori kan man inte lösa problemet med endast P-reglering eftersom systemet behöver ett styrvärde som inte är noll. Detta innebär att när bollen är på plats, alltså felvärdet blir 0, så blir även styrsignalen 0.

4.6.2. I-Reglering

I-reglering där utsignalen är integral av felvärdet e. Eftersom integralen är en viss utsignal i en viss tidpunkt, så är tiden väldigt viktig i regulatorn och den beror på felvärdet i just den tidpunkten.

Genom att kombinera P-reglering och I-reglering, kan vi utnyttja P-regulatorns goda egenskaper och bli av med problemet som sker då styrvärdet blir 0 eftersom I-regulatorn inte tillåter det.

När man minskar den integrala konstanten K_i så får man en bättre elimination av de kvarstående felen.

4.6.3. D-Reglering

D-reglering där regleringen är beroende av derivatan av insignalen. Detta innebär att när derivatan är skild från 0, blir även utsignalen från D-regulatorn det. Men utsignalen blir 0 när felvärdet blir konstant.

4.6.4. PID-Reglering

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

Figur 4: Den allmänna PID-formeln[5]

Koden som skrevs i C är modellerad efter denna formel.

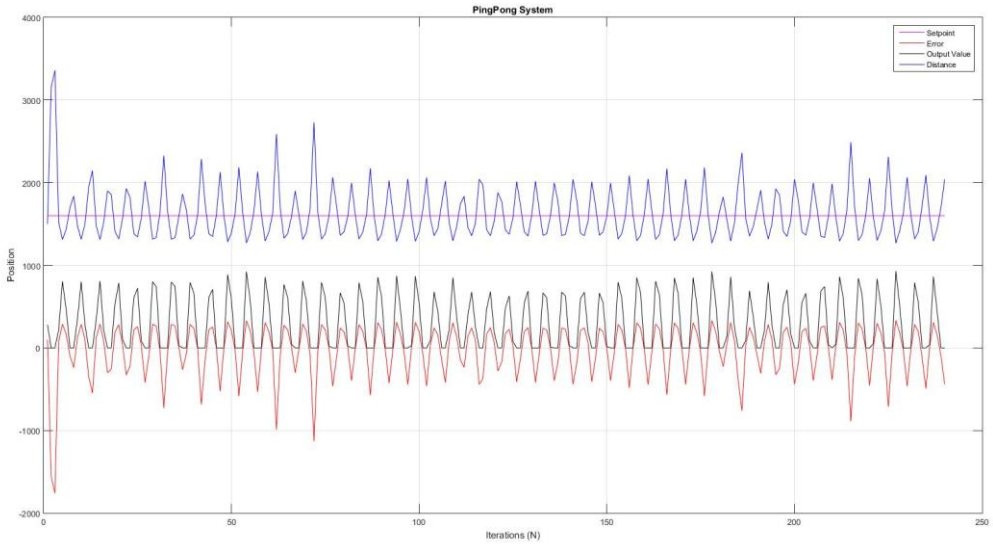
4.6.5. Inställning av parametrar

Vid inställning av parametrar användes Ziegler – Nichols metod för att ställa in PID variablerna.

Tumreglerna för inställning är följande:

- PID-regulatorn ställs först in som en P-regulator
- Man ökar den proportionella konstanten K_p fram tills $K_p=K_0$ då systemet börjar oscillera.
- Efteråt mäter man T_0, periodtiden för oscillationen.

Värdet på Kp blev 2.8 vilket gav följande graf:



Figur 5: Plotten för P-reglering

Därefter beräknades konstanterna enligt denna tabell:

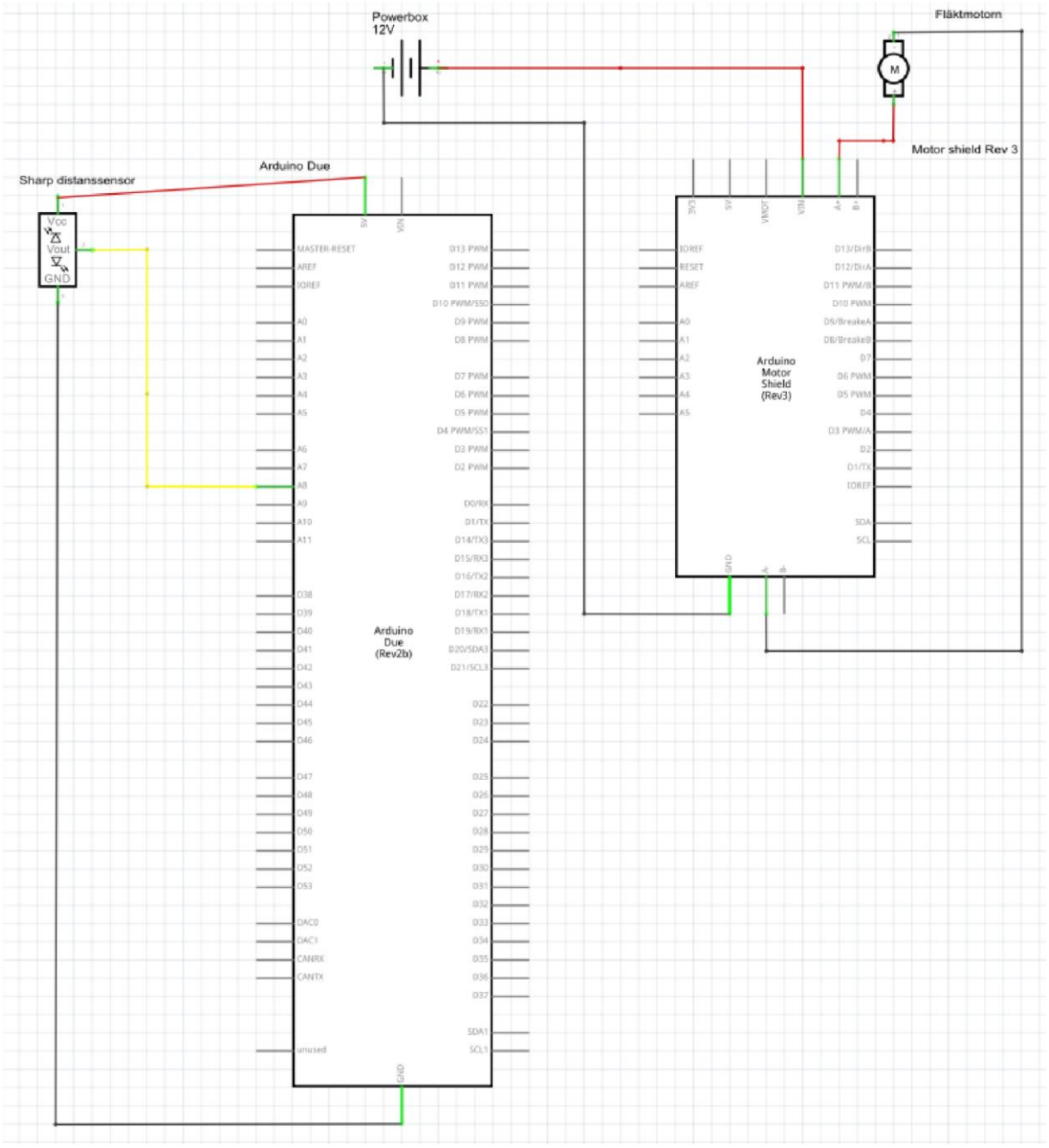
	K	Ti	Td
P			
PI			
PID	1.65	1.25	0.31

4.7. Kommunikation mellan Arduino Due och Matlab

Denna utförs genom UARTen. Matlab skickar först parametrar för PID reglering vilka är *börvärdet, samplings tiden och PID konstanterna*. Eftersom RHR på Due kortet kan bara hålla 8 bitar åt gången[5], får dessa skickas som int8 och för att få en float eller double delas dessa i Due kortet med en *divider* variabel som är på 10. Dessa variabler behöver även typecastas eftersom de alla skrivs som chars[5]. I och med att talen för börvärdet överskrider int8 skriver Matlab istället värdet i cm, som senare går igenom en switch sats och sätter börvärdet till dess motsvarighet i ett av värdena jag har mätt på rälsen.

Returvärdena skrivs som vanliga ints till konsolen vilket då Matlab fångar upp och plottar en graf. Förutom de variabler som det finns krav på, skriver Due även börvärdet.

5. Kopplingschema



6. Resultat

Resultaten blev att PID regulatorn inte fungerade som det ska. Allt annat i programmet fungerar och har testats.

7. Diskussion

Jag har försökt varje variation av kod, debuggat hela programmet, testat olika system eftersom jag trodde att det var fel på sensorn och/eller motorn men det är först nu jag förstår var felet uppstår. Som det syns i grafen i figur 3 och på denna demo videon: https://www.youtube.com/watch?v=u7hBuCg73p0 så beter sig regulatorn som en P-regulator utan någon inbromsning trots att variablerna som skickas från är alla tre konstanter från PID. Den träffar rätt börvärde på rälsen och rätt samplingstid men det verkar inte som alla parametrar för PID fungerar.

Denna slutsats drar jag eftersom det sker ingen inbromsning då felvärdet blir noll, och det ska det bli om I delen är fungerande.

Alltså kan det finnas två fel:

- Att värdena som skickas från Matlab inte typecastas som de ska.
- Att formeln för PID reglering i C är fel.

8. Källförteckning

[1] B. Thomas, Modern Reglerteknik.

[2] G.K. Svedberg, Examinationsuppgift 2014

[3] G.K. Svedberg, ”Lab3c”

[4][Online] https://www.arduino.cc/en/Main/ArduinoMotorShieldR3

[5][Online] http://www.atmel.com/images/atmel-11057-32-bit-cortex-m3-microcontroller-sam3x-sam3a_datasheet.pdf