MALMÖ HÖGSKOLA
CENTRUM FÖR
TEKNIKSTUDIER (CTS)

Inbyggda system och signaler
Styr- och reglerteknik

Examinationsprojekt

Namn: Ali Rama, Matko Scapec-Kukina

Gion Koch Svedberg
Decemeber 2015

Innehåll

1. **Inledning …….**
2. **Syfte …….**
3. **Genomförande ……....**
   3.1 Elektronik ……...
4. **Reglering …….**
   4.1 P- regulator.....……...
   4.2 PI – regulator ……..……
   4.3 PID- regulator…….…….
   4.4 Implementering av PID- regleralgoritm……..
      4.4.1 Tumreglermetoden……...
5. **Pulse Width Modulation (PWM)……...**
6. **Kommunikation mellan Arduino och Matlab …….**
7. **FreeRTOS……**
8. **Resultat…….**

1. Inledning

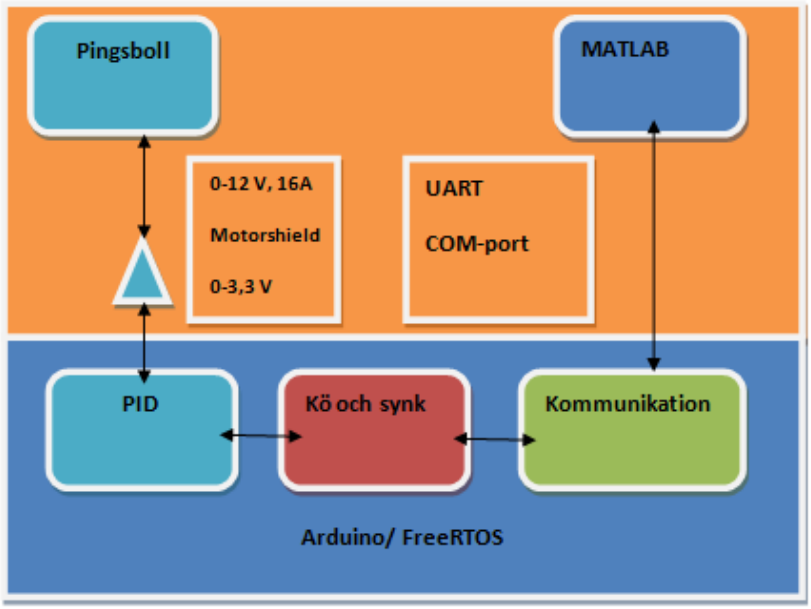
Syftet med detta examinationsprojekt är att praktiskt applicera olika klassiska regleralgoritmer på ett fysikaliskt system. Systemet består av en fläkt som har i syfte att hålla en pingisboll på en konstant avstånd på det slutna planet. För att kunna genomföra uppgiften enligt ovanstående beskrivning krävs förverkligandet av följande delmål:

- Programmering av ADC i Atmel i syfte konvertera analog till digitala mätvärden för sensorn för att avgöra avståndet på pingisbollen.

- Programmering av PWM i Atmel i syfte att reglera fläkten för olika hastigheter 0-12 V motsvarande 0-100% i duty cycle.

- Programmering av UART i Atmel/Matlab i syfte att upprätthålla kommunikation mellan Matlab och Arduino Due. Detta är i denna del som startvärden för regleralgoritmen kommer att anges samt att resultatet illustrerars grafiskt i real-time system. Inmatning av startvärden för regleralgoritmen utgörs av parametrarna *KP* för proportionell förstärkning, *KI* för förstärkning av integrerande delen, *KD* för förstärkning av den deriverande delen och slutligen börvärde *W* för ett visst avstånd på planen.

- Programmering av FreeRTOS i syfte att utföra real-time operationer simultant och under schemalagda prioriteringar.



Figur 1: *Illustration över systemets alla delar och dess helhet*

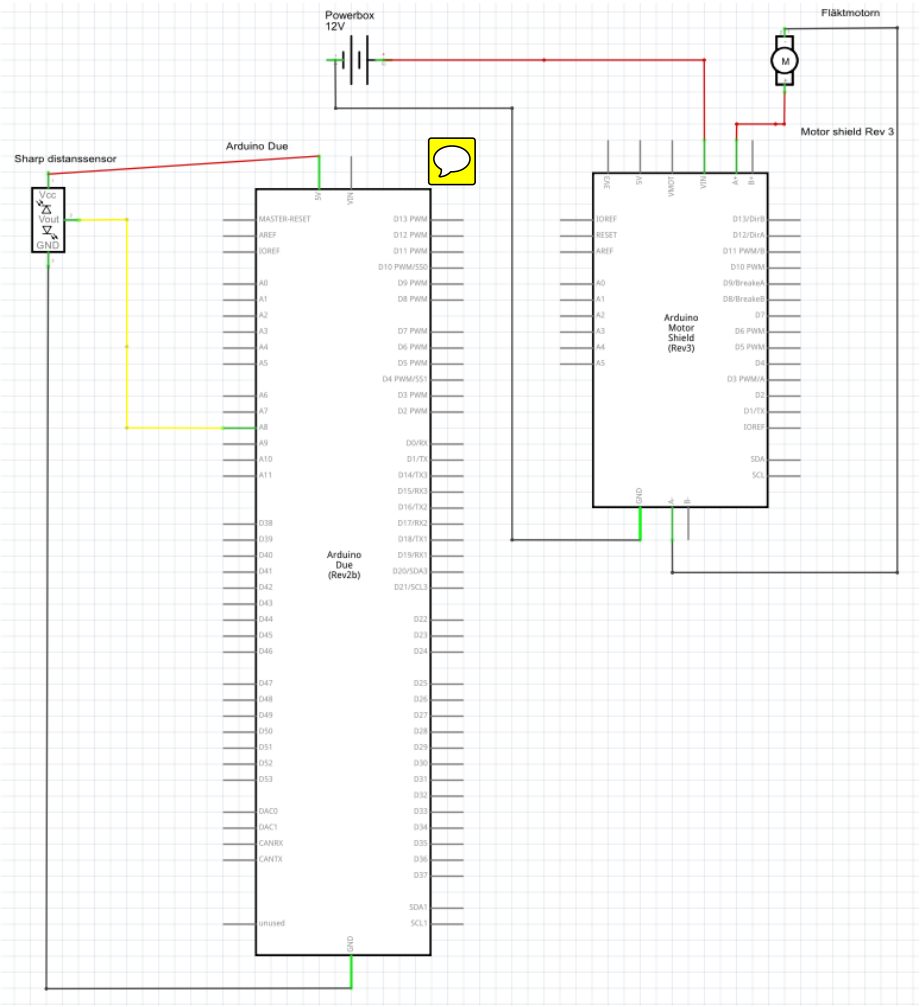
2. Syfte

I denna rapport skall den individuella gruppmedlem beskriva hur man gått tillväga för att lösa samtliga problem som man har ansvarat för i gruppen.

3. Genomförande

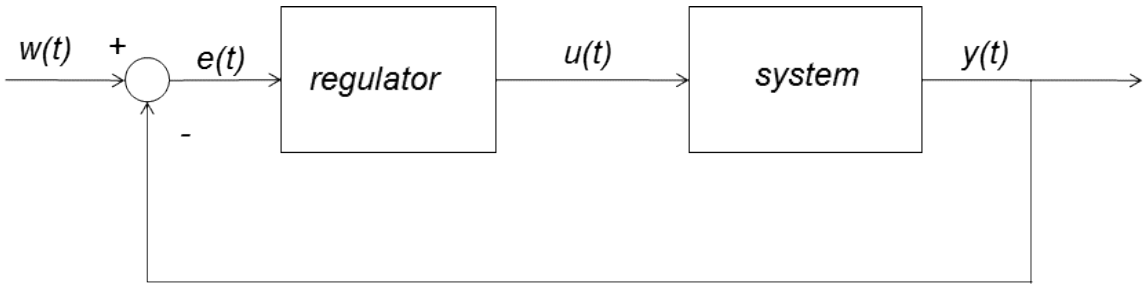
För att kunna styra fläkten används Arduinos motor-shield som ger möjlighet till att ansluta och styra en extern spänningskälla. Arduino Due styrs med 3,3V, högre spänning som kopplas till ingångarna bränner kortet. Eftersom detta inte räcker för att driva fläkten på 0-12V och 1,6A samt distanssensor med 5V används motor-shield. Till motor-shielden kopplas bl.a. powerboxen genom Vin och Ground som försörjer fläkten. Fläkten PWM-styrs genom *kanal A*. För att kunna PWM-styra *kanal A* används pinnen PWM *A pin3* via pinnen *pin 7*. Distanssensors värden läses via pin A8 på duen och även denna kopplas till Vin och ground.

3.1 Elektronik



Figur 2: *Schema över kopplingen för System, DUE+Motor-shield, Powerbox m.m*

4. Reglering



Figur 3: *Illustration ett blockdiagram av en enkel regelkrets*

Börvärdet *w(t)* och Ärvärdet *y(t)* är de värden vars differens bidrar till reglerfelet/störning vid differenspunkten i blockdiagrammet. Reglerfelet/störningen *e(t)* är den insignal som påverkar signalen *u(t)* d.v.s. styrsignalen för att justera för aktuella avvikelser. Styrsignalen *u(t)* är i sin tur den insignal som påverkar utsignalen *y(t)* d.v.s. anger det aktuella värdet efter justering.

Reglersystemet har i uppgift att kompensera för störningar *e(t)* så att dessa inte får för stor inverkan på den reglerade storheten (systemets utsignal) d.v.s. ärvärdet *y(t).* Regulatorn ska snabbt upptäcka avvikelser mellan börvärdet *w(t)* och reglerade storheten *y(t)* och vid behov justera styrsignalen så att avvikelsen försvinner. Reglersystemet har i uppgift att följa ändringar i börvädet. Om börvärdet ändras ska regulatorn se till att systemets reglerade storhet svänger in sig till detta nya värde. (Thomas B, 1997, s. 11). För att kunna justera pingisbollens position togs beslutet att använda PID- regulatorn för att uppnå detta ändamål. Valet av PID- regulator valdes p.g.a. av den goda kännedom vi har om regulatorn från tidigare arbetsuppgifter i skolan men även för de flertal fördelar som denna bidrar till jämfört med andra kombinationer regulatorer.

4.1 P-regulator

När det gäller proportionell reglering (p-reglering) är förändringarna i styrsignalen u proportionella mot reglerfelet e d.v.s. insignalen till regulatorn. Sambandet mellan felsignal e och styrsignalen u kan beskrivas med följande: $u = u_0 + K \cdot e$. Styrsignalens normalvärde u_0 är det värde på styrsignalen har då felet = 0. Förstärkningen K bestämmer hur mycket regulatorn ska ge för att justera för felet som uppkommit, d.v.s. hur mycket styrsignalen u ska förändras när felet e ökar med en enhet. När det gäller konstanten u_o bör man utgå från att den motsvarar normalt börvärdet. Konstanten k bör väljas till så litet som möjligt för att ett lägre värde ger ett

system en god stabilitet men inte lika god snabbhet och ett större värde på konstanten *k* ger vice versa. Sambandet som för felsignalen *e* och styrsignal *u* gäller för ”ideal” P-regulatorer medan det i praktiken finns ett gränsvärde för hur höga respektive låga värden styrsignalen kan anta. Man brukar säg att styrsignalen är proportionell mot felsignalen endast inom ett specifikt område s.k. proportionella bandet (Thomas B, 1997, s. 51, 199).

Om e > 0

$$u = K \cdot e$$

Annars

$$u = 0$$

4.2 PI- regulator

När P- och I-delen kombineras kallas man det för PI-reglering. Detta för att skapa en kombination med fördelarna från varje regulator typ. Sambandet kan beskrivas enligt följande:

$$u(t) = K\left[e(t) + \frac{1}{T_I}\int_0^t e(t)dt\right]$$

Förstärkningen *K* påverkar båda termerna i PI-regulatorn. Integreringstiden T_I anger hur snabbt I-delens utsignal ändrar sig jämfört med P-delens vid ett bestämt reglerfel. Integreringstiden T_I väljs vanligtvis hyfsat stor vilket leder till att I-delens utsignal ändrar sig ganska långsamt jämfört med P-delen. Integreringstiden T_I anger därmed också tiden det tar för I-delens utsignal att bli lika stor som P-delens utsignal (Thomas B, 1997, s. 60, 203)

Om e > 0

$$u = K(e + (\frac{dT}{Ti} \cdot (summan\ av\ felsignaler))$$

Annars

$$u = 0$$

4.3 PID- regulator

När det gäller D-blocket har den funktionen att dess utsignal är skild från noll endast när insignalens derivata är skild från noll. Detta innebär att desto större förändring av insignalen d.v.s. själva felet desto större utsignal. Detta innebär också att en konstant insignal ger en utsignal som är noll. En PID-regulator består av summan av resultatet från samtliga tre block. Sambandet mellan insignalen *e* och utsignalen *u* kan formuleras enligt följande (Thomas B, 1997, s. 61, 211).

$$u(t) = K\left[e(t) + \frac{1}{T_I}\int_0^t e(t)dt + T_D \cdot e'(t)\right]$$

Om e > 0

$$u = K(e(t) + T_D \cdot \left(\frac{e(k)-e(k-1)}{d_T}\right) + \int_0^t e(t) \cdot \left(\frac{d_T}{T_I}\right))$$

Annars

$$u = 0$$

Först beräknas felvärdet *error* genom att beräkna avståndet mellan börvärdet *disantceSetCM* och ärvärdet *distance*. *I_SET* utgör den integrerade delen och *D_SET* utgör den deriverande del i regleringen som tillsammans bidrar till utvärdet *output_value*. Därefter görs en koll på om värdet är för högt respektive för lågt och slutligen skrivs PID värdet till PWM för att styra fläkten

```
// Propertionell-del
error = (setPoint - distanceSensor);

// Integrerande-del
sumOfError = (double)sumOfError +
(double)((double)error*(double)DT_SECONDS);
double I_Output;
if(kI == 0)
{
            I_Output = 0;
            } else {
            I_Output = (double)kI*sumOfError;
}

// Deriverande-del
double D_Output;

if(error == 1)
{
            D_Output = 0;
            } else {
            D_Output = (double)((double)((double)kD*(double)(error -
lastError))/(double)DT_SECONDS);
}
lastError = error;

// P, I and D outputs
output_value = (kP*error)+I_Output+D_Output;
```

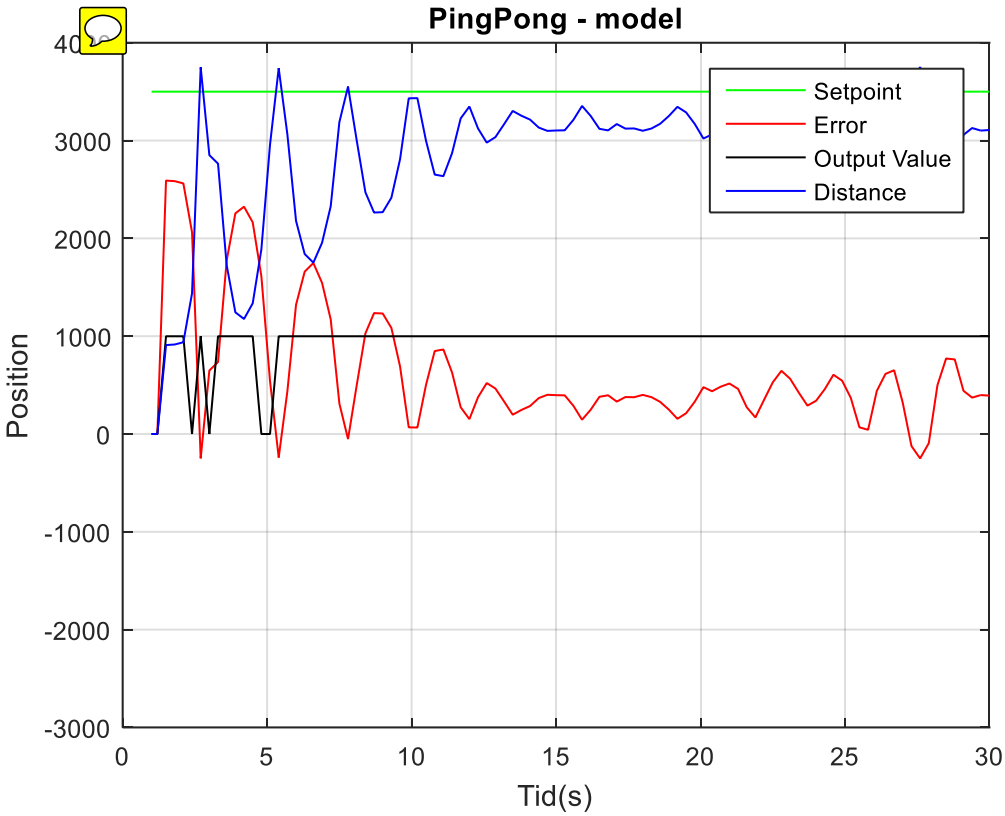
4.4 Implementering av PID- regleralgoritm

4.4.1 Manual tuning

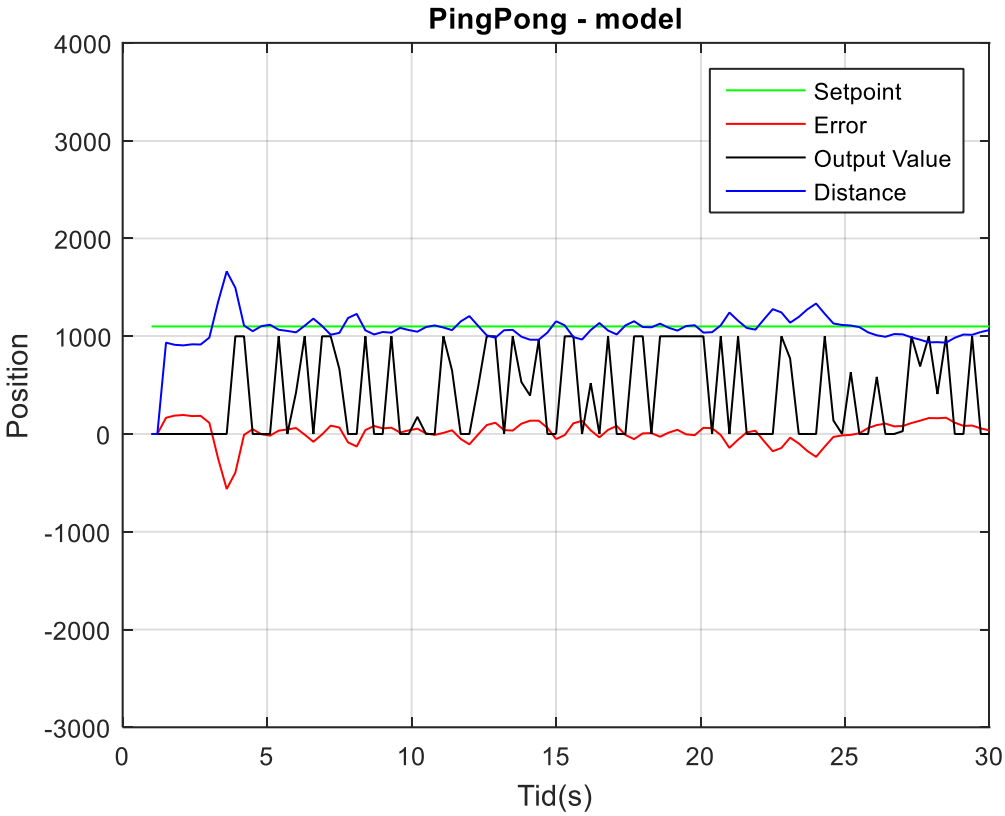
Manual tuning eller manuell inställning bygger på att man först ställer in K_i och K_d värden till noll och samtidigt ökar $K_p$fram till det att bollen oscillerar. Man ökar sedan K_i fram till att avvikelser korrigeras och är i någorlunda god tid för processen. Denna parameter bidrar till mycket instabilitet och för begränsas kraftigt. Slutligen ökar man K_d fram till det att börvärdet kan nås tillräckligt snabbt efter en tid av obalans.

Denna metod fungerar väldigt bra under testerna och vi lyckades ta fram värden för K_p, K_i, K_d som bidrag till en god reglering mot börvärdet. I samband med *”pingis-modellen*” har vi under tester lagt märke till att vi är i behov av höga värden på K_d då *”pingis-modellen”* oscillerar mer än andra tidigare testade system och det finns därför behov av kraftigare felkorrigeringar som i vanliga fall resulterar till en ostabil reglering som t.ex. med ”*vatten-modelen*”. Det kraftigare oscillerandet i systemet beror också på att sensorn är olinjär vilket medför att mätvärden från sensorn skjuter i höjden vid högre setpoint > 30 på banan och påverkar PID-regleringen negativt. Detta innebär bl.a. att olika värden för PID-regleringen krävs för setpoint > 30 och setpoint < 30.

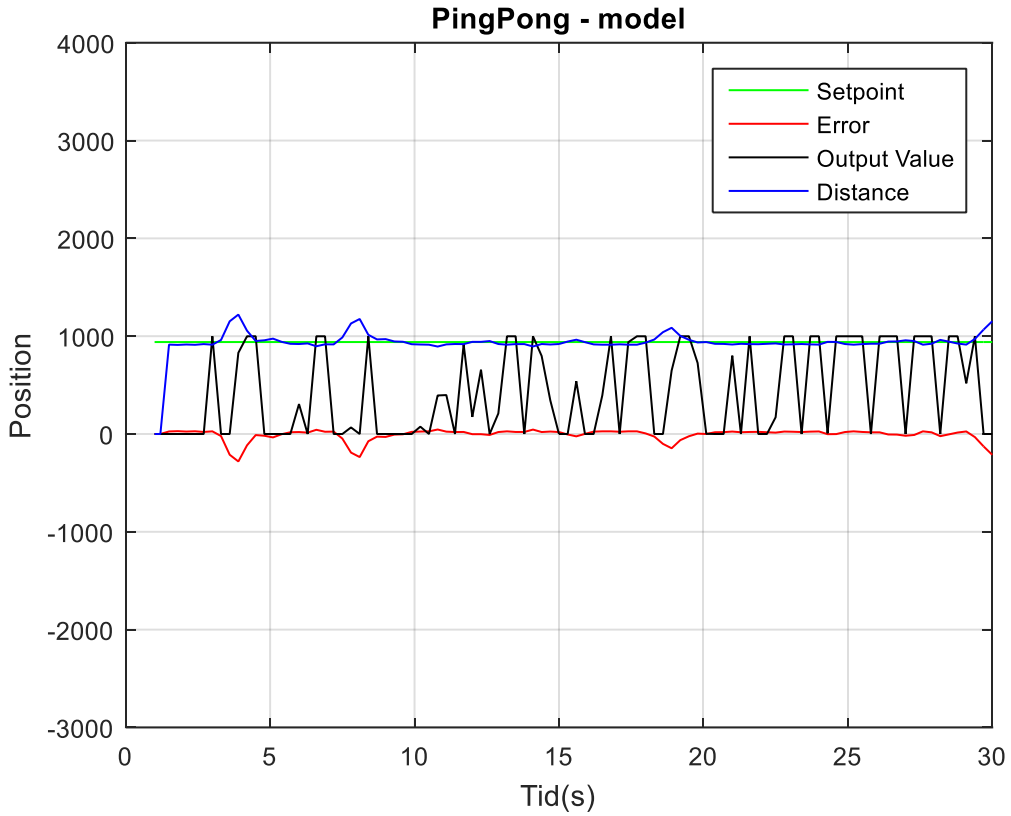
```
%startPID('COM4', 0.3, 30, 10, 1.1, 2.2, 16.0)
```



Figur 4: *Illustration av PID-reglering med börvärdet 10.*



Figur 5: *Illustration av PID-reglering med börvärdet 30.*

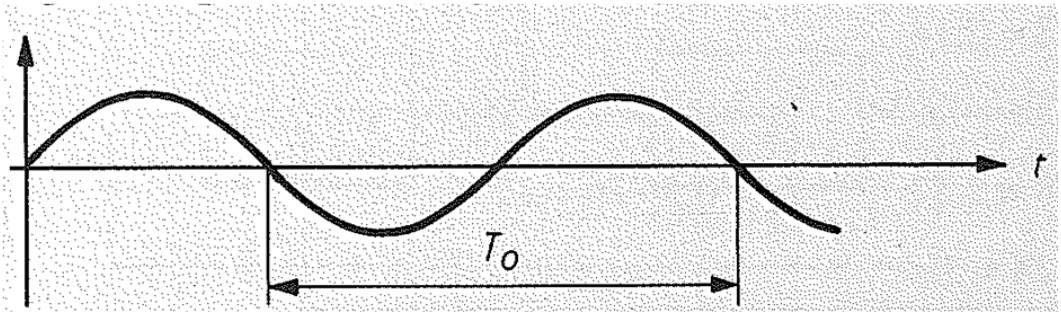


Figur 6: *Illustration av PID-reglering med börvärdet 40.*

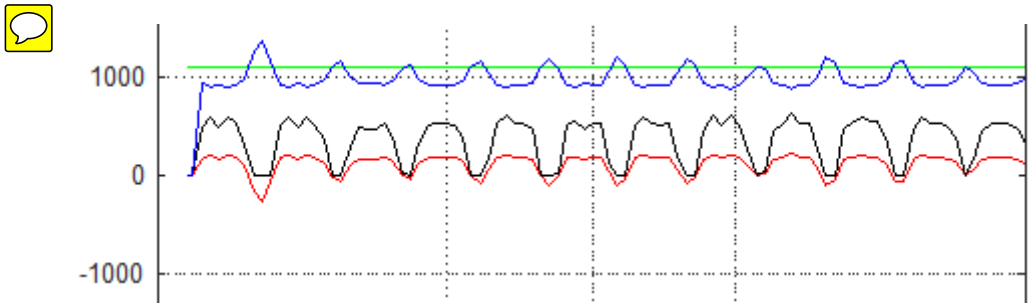
4.4.2 Tumreglermetoden

Tumreglermetoder kan beskrivas som enklare inställningsmetoder som inte kräver att man känner till den överföringsfunktion för processen som ska regleras. De används då främst i fall där kraven för regleringen inte är så höga. Fördelen med tumregler-metoder är just att man undan kommer omfattande teoretiskt arbete. Nackdelen är att det endast leder till en grovinställning av en regulator. Resultatet blir en blandning av olika egenskaper som t.ex. stabilitet, snabbhet, dämpning av störningar m.m (Thomas B, 1997, s. 190).

Ziegler-Nichols svängningsmetod innebär att man först ställer PID-regulatorn som en ren P-regulator med låg förstärkning d.v.s. att *Td* och *Ti* sätts till noll respektive oändligheten. Därefter ökar man förstärkningen *K* gradvis fram till dess att reglersystem börja självsvänga vilket sker när *K=K0.* Därefter mäter man upp den periodtid *T0* för självsvängningen. Och därefter ställer man inte regulator parametrarna enligt angiven tabell för metoden.



Figur 7a: *Illustration av självsvängning enligt teori då K=K0.*



Figur 7b: *Illustration av självsvängning i praktiken då K=K0.*

Type	k_p	T_i	T_d
P	1/a		
PI	0.9/a	3τ	
PID	1.2/a	2τ	0.5τ

(a) Step response method

Type	k_p	T_i	T_d
P	$0.5k_c$		
PI	$0.4k_c$	$0.8T_c$	
PID	$0.6k_c$	$0.5T_c$	$0.125T_c$

(b) Frequency response method

Figur 4: *Illustration av tumregler för att bestämma PID-parametrar (FRM)*

När vi tillämpade svängningsmetoden i syfte att ta fram PID-parametrar var vi medvetna om att det endast skulle bidra till grovinställningar och inte optimala värden. Detta kan tydas på de bristande PID-regleringen under testerna med Ziegler-Nichols svängningsmetod.

startPID('COM4', 0.3, 30, 30, 2.9, 0, 0)

5. Pulse Width Modulation (PWM)

I uppgiften används för att styra motorn till fläkten baserad på PID- regleringens insignal. I uppgiften används den inbyggda PWMn på DUE brädan vars information om användning var begränsad i samband DUEN processor. PWM Clock A angavs till 1Mhz och Master Clock till 84Mhz.

6. Kommunikation Arduino & Matlab

Matlab klassen d.v.s. funktionen *startPID* innehåller parameter som bl.a. tar emot COM-porten för PC, dT, börvärdet, Kp, Ki, Kd och samplingstiden. Det är även i denna funktionen som värden skickas och tas emot från DUEN och slutligen plottas.

```
function [] = startPID(port, dT, T, setpoint, Kp, Ki, Kd, sampletime)
```

```
%Send value to Arduino
fwrite(arduino, Kp, 'int8');
pause(1);
fwrite(arduino, Ki, 'int8');
pause(1);
fwrite(arduino, Kd, 'int8');
pause(1);
fwrite(arduino, sampletime, 'int8');
pause(1);
fwrite(arduino, setpoint, 'int8');
pause(1);
```

```
%Read values from Arduino
    etemp = fscanf(arduino,'%d')
    otemp = fscanf(arduino,'%d')
    dtemp = fscanf(arduino,'%d')
    sptemp = fscanf(arduino, '%d')
```

7. FreeRTOS

RTOS håller reda på två trådar varvid den ena är för PID-regleringen och den andra är UART-kommunikationen mellan Matlab och Arduino. Uppgiften för PID-regleringen har fått höst prioritering p.g.a. av PID-regleringen är mer tidsberoende för systemet än U-ART kommunikationen. För att kunna kontrollera när PID-regleringen skall starta används en binär semafor vid initialiseringen av programmet. Detta ger Matlab tid att skicka samtliga parametrar och undviker därmed att PID-regleringen startar i förtid i bakgrunden. När Matlab har skickat

alla parametrar körs ett kommande som släpper den semafor som PID-tråden tagit och på sätt startar PID-regleringen med det värden togs emot från Matlab.

```
void PIDTask (void *pvParameters)
void ComTask (void *pvParameters)
```

```
xSemaphoreTake(sem, portMAX_DELAY);
xSemaphoreTake(sem, portMAX_DELAY);
```

PID-tråden har en samplingstid mellan 50-100 ms och anges i Matlab funktionens parameterhuvud s.k. *sampletime*. UART-trådens samlings tid däremot beror på parametern *dT* i Matlab funktionens parameterhuvud. Denna tråd är under tiden fryst och inväntar klartecken från Matlab genom UART, när ett sådant klartecken uppfattas skickar PID-tråden resultaten till Matlab för att återigen börja om.

Funktionen *getPIDValues* tar emot värden från Matlab och *sendValues* funktionen skickar värden till Matlab. I samma klass anges även de förväntade värden för de olika avstånden på systemet som används för att bl.a. beräkna felmarginal. I *getPIDValues* lagras värden som sedan används i *PIDReglering*.

```
void getPIDValues()
{
		const uint8_t divider = 100;
		const uint8_t timeDivider = 10;

		while (!uart_is_rx_ready (CONF_UART)){
			vTaskDelay(1);
		}
		uart_read(CONF_UART, &P_CONSTANT_temp);
		while (!uart_is_rx_ready (CONF_UART)){
			vTaskDelay(1);

		P_CONSTANT = (double) ((double) P_CONSTANT_temp / divider);
		I_CONSTANT = (double) (I_CONSTANT_temp / divider);
		D_CONSTANT = (double) (D_CONSTANT_temp / divider);
```

```
void sendValues(){
		// printf distance, error, output_value
		printf("%i\n\r", error);
		printf("%i\n\r", output_value);
		printf("%i\n\r", distance);
		printf("%i\n\r", distanceSetCM);
```

8. Resultat

När det kommer till resultat kan man gott säga att vi lyckades. Den koppling som vi satt upp för uppgiften var inga svårigheter. Det vi fick tänka på var att fläkten skulle PWM-styras med olika hastigheter mellan 0-12 V och att Arduino Duen endast fick matas med 3,3 vilket innebar att vi var i behov av en motorshield. FreeRTOS och tillhörande trådar fungerade utmärkt även om det var en svårt att få rätt på till en början.

När det kommer till PID försökte vi finna rätt värden med främst Ziegler-Nichols svängningsmetod genom att börja med en enkel P-regleringen och sedan bygga vidare på detta enligt tabellen. Vi lyckades få en bra svängning för p-regleringen men när vi fortsatte med att tillämpa värden enligt tabellen spårade detta ut. Vi tror att det beror på att sensorn är olinjär och sätter därför PID-regleringen ur spel med värden från tabellen. Det som däremot fungerade bättre var manuell inställning av PID-värden som till en start påminner om metoden för Ziegler-Nichols. Vad som fungerade bättre med denna metod var just det att vi kunde anpassa PID-regleringen efter varje regulator del för att få bästa möjliga sammanslagning av p,i,d för det aktuella systemet. Nackdelarna med manuell inställning är att den inte heller kunde justera för de olinjära värdena som sensorn orsakade. Vad man kunde lägga märke till när det gäller implementeringen av PID var det att man antingen fick bra resultat av implementera PID-regleringen för börvärden 10-30 eller 40-50 på systemets ramp.

När det gäller gruppen har vi haft en god samarbete internt i vår grupp fastän vi haft en del motgångar vad gäller utrustning och tillgång till elektroniklabb salarna. Den bristande möjligheten till utrustning avseende arduino due och motorshield har också tagit tid och fokus från själva uppgiften.