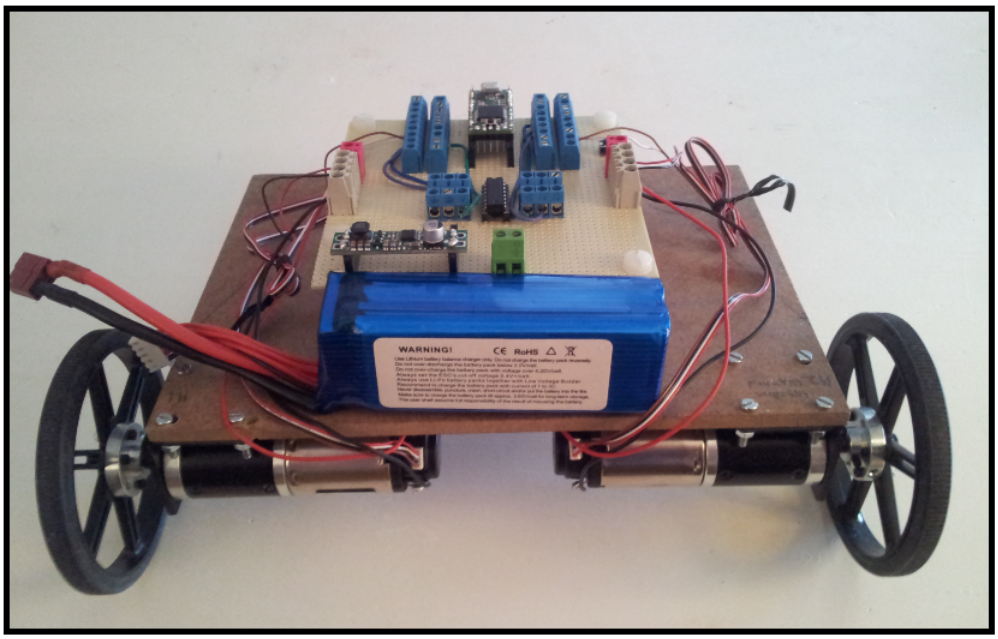
Konstruktion af robotkøretøj 3 ugers kursus

Forfatter: Støhrmann, Nilas
DTU s103313 Elektro Diplom

Vejleder: Axel Andersen, Nils
DTU Automation And Control

February 12, 2015



Contents

1 Indledning

Følgende rapport omhandler design og konstruktion af et simpelt robotkøretøj således, at dette kan klare nogle af de udfordringer der stilles i Robotcup. Fokus vil være begrænset til; valg af komponenter, konstruktionsdesign, software implementering og implementering af relevante regulatorer. Rapporten er altså tiltænkt, at opbygge robottens hardware samt dens styringsmuligheder i relation til robocup således, at der senere hen direkte kan implementeres avanceret funktionalitet.

2 Robotten

2.1 Komponenter

Tilrådighed er to phidgets DC gear motorer med quadrature enkodere monteret og i tabel 1. ses nogle af nøgleparametrene for disse type motor taget udgangspunkt i gear aksen. Motorerne har en max hastighed på 285 rmp som er mere end optimal til roborup formål, der anbefaler 51 rmp. Tilmed kan motorerne leverer ca. 12 kgcm moment hvilket er en passende mængde til denne robot type taget størrelse og antaget vægt i betragtning. Motor enkoderne har en god opløsning på 1440 cycle/rev, der kan derfor opnås nøjagtig positions- og hastighedsmåledata.

Max speed	Stall Current	Max V_a	Weight	Ratio	Torque	Encoder	k_e
285 RPM	1.5A	$12V$	174g	14:1	12kgcm	1140 C/rev	0.0226 $\frac{v}{\frac{rad}{s}}$

Table 1: Motor parameters gear akse

Som hovedstyringsenhed benyttes en teensy 3.1 som er en arm cotex m4, 32 bit mikrokontroller. Denne har bl.a. 72 MHz clockrate, 64 Kbyte ram, 34 GPIO pins heri 10 analog pins, 10 PWM(8 og 12 bit) pin og meget andet. Til styring af motorerne benyttes en l293d full dual H-bro, denne leverer kun 0.6 A pr motor og har derfor sine begrænsninger i forhold til motororen højere moment og stall torque. Beregninger af motor moment er fortaget i afsnit 2.3. Som 5 volt strømforsyning til diverse enheder benyttes en D15V35F5S3 step down spændings regulator fra pololu, som regulerer 12 volt batteri spænding ned til 5V, tilmed kan regulatoren leverer op til 3.5 A.

2.2 Opbygning

I figur 1. ses opbygningen af prototypens styringsplatform. Designet er tiltænkt simpelt, alle porte på mikrokontrolleren og driveren er forbundet til skrueterminaler under print pladen, således er det enkelt at forbinde komponenterne, motor samt enkoder og skifte porte efter behov. Ligeledes er strømforsyning til de diverse enheder også forbundet på samme måde. I det nuværende design er knapperne ikke forbundet, disse skal senere hen bruges til reset og start af missioner. Platformen er monteret på en mdf plade med mål 22x16x0.6 cm herpå er motorerne monteret med phidgets mounting brackets. Pololu hjul med radius 4.6 benyttes på motor akse og foran er monteret et kuglehjul, drejestyringen foregår derfor udelukkede fra motorerne. Robottens væg bliver ca. 950 g ialt.

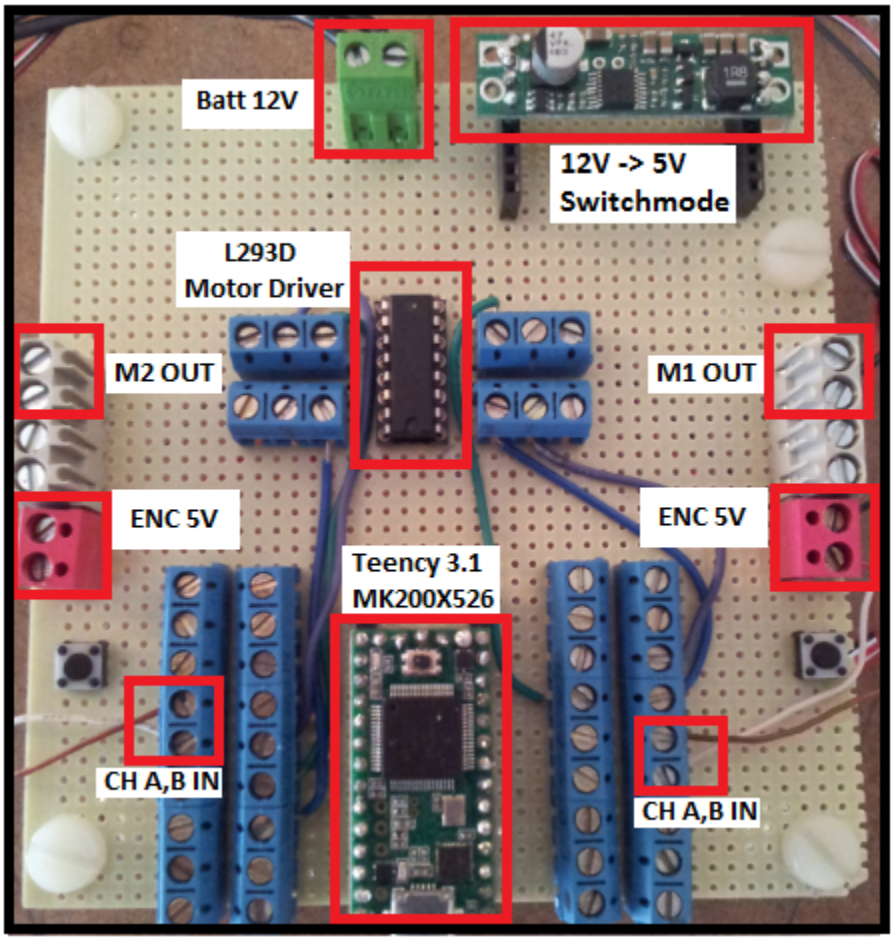


Figure 1: Prototype Styringsplatform

2.3 Moment og trækkraft

Som oplyst kan motor driveren l293d kun leverer 0.6A/motor, der kan ud fra denne værdi bestemmes det maksimale moment og kraft driveren kan leverer til hver motor ved

$$P = \tau_{dmax}\omega = I_{dmax}V_{amax} = \tau_{dmax}\frac{V_{amax}}{k_e} \Rightarrow \tag{1}$$

$$\tau_{dmax} = I_{dmax}k_e * G = 0.6 * 0.0226 * 14 = 0.18984kgcm \tag{2}$$

Den maksimale kraft fra driveren bliver

$$F_{dmax} = \tau_{dmax}r = 0.18984kgcm * 4.6cm = 0.873264N \tag{3}$$

Den kraft der ydes ved stall torque bestemmes til

$$F_{stall} = \tau_{stallgear} r = 12kgcm * 4.6cm = 55.3N \tag{4}$$

Der skal altså benyttes en driver der kan leverer >55.3N således at driveren ikke brænder sammen ved stall torque, nuværende driver er altså ikke optimal. For at robotten skal kunne klare rappekørsel i robocup ved hældning 25 grader, bestemmes det nødvendige momoent der skal kunne leveres. Ved at analyserer hvilket krafter der påvirker robotten på en hældning i x retning bestemmes grænse momentet, dette gøres i (5) og (6).

$$\sum F_x = F_{motor} - gm \sin\phi = 0 \Rightarrow F_{motor} > gm \sin\phi \tag{5}$$

$$F_{motor} = \tau_{motor} r \Rightarrow \tau_{motor} > \frac{gm \sin\phi}{r} = \frac{0.950kg * 9.8\frac{m}{s^2} * \sin\frac{25*2\pi}{360}}{4.6cm} = 0.8553426122kgcm \tag{6}$$

Der skal altså leveres større moment end 0.85 kgcm for at robotten skal kunne bevæge sig opad rappen. L293d kan altså heller ikke leverer moment nok til at klare denne udfordring.

2.4 Dekodning

For at kende hastighed og position skal motor enkodernes feedback signaler dekodes. Som set i tabel 1. giver enkoderen 1440 cycles/rev på gearaksen svarende til 20160 pulser/rev. Pulserne for hver kanal opfanges ved high/low GPIO pin interrupt på forbundet pins. I kanallens tilhørende ISR (interrupt service rutinen) tages tidsmålinger med en opløsning svarende til tiden mellem to pulser i mikrosekunder (målt på kanal A på hver motor).Derved fås følgende konvertering for hastighed

$$\omega_{\frac{rad}{sek}} = \frac{\frac{2\pi}{20160}}{(t_{cur} - t_{last})10^{-6}} = \frac{623.33}{t_{\mu s_{(2puls)}}}, \omega_{\frac{rev}{min}} = \frac{623.33}{t_{\mu s_{(2puls)}} 2\pi} 60 = \frac{5952.38}{t_{\mu s_{(2puls)}}} \tag{7}$$

Der er med oscilloskop målt ca. $20\mu s$ mellem de 2 pulser i en af A kanallerne ved 30 rad/sek. Quadrature enkodernes hastighed vil have perioder der ændres ca. målt til $1 - 2\mu s$ udsving og dette skaber højfrekvente målestøj i hastighedsmålingerne. En løsning til dette problem kunne være at benytte en højere tidsopløsning f.eks. i nanosekunder, denne opløsning er ikke altid supporteres i mikrokontrollere. I stedet for er der opnået bedre måleresultater ved at midle målingerne mellem sampletider. Der er også forsøgt implementeret et digitalt 1.ordens lavpasfilter ved 1Kz sampletid og knækfrekvens ved 60 rad/sek, hvilket ikke vidst nogen bedring, det kan da der naturligvis arbejdes videre senere. Nedenstående 1.orden lavpasfilter er forsøgt implementeret

$$y_i = \alpha_i + x_i(1 + \alpha)y_{i-1}, \alpha = \frac{Ts}{\frac{1}{2\pi f_c} + Ts} = 0.2737789035 \tag{8}$$

For at dekode position og rotationsretning af akse kan alle pulser i kanal A og B tælles, dette gøres også i ISR. Nedenstående flow i figur 2. viser hvordan dekodningen af position implementeres i ISR for begge enkoder kanaller i en motor. ISR opdaterer en fælles variable som tæller pulserne og alt afhængig af hvilket pin state(high/low) kanal A og B har i forhold til hinanden, justeres denne variable op eller ned.

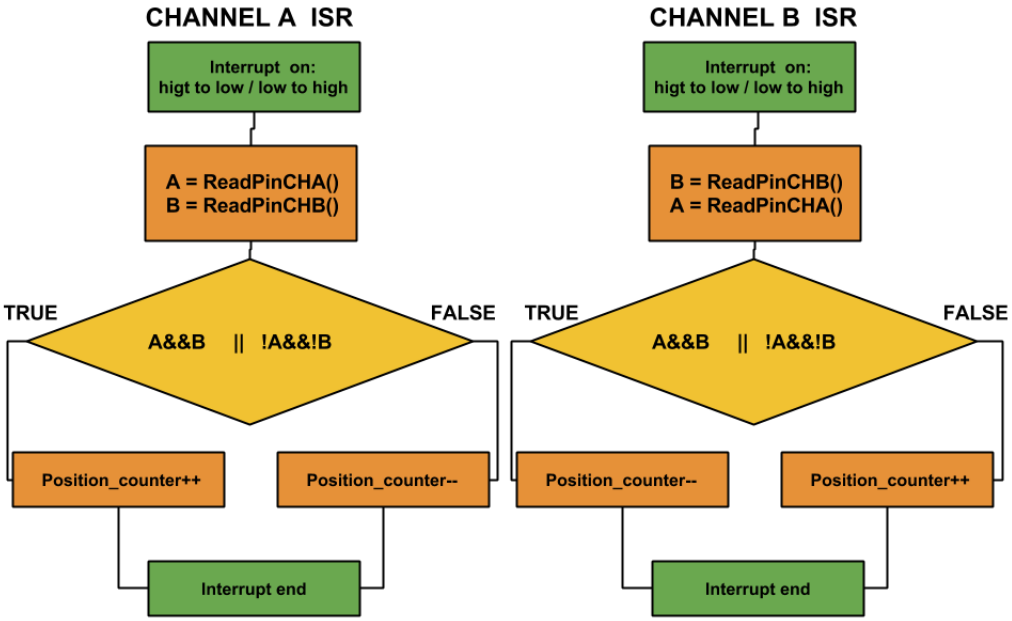


Figure 2: Position- og retningsdekodning

Med en radius på 4.6 cm fås følgende position konvertering på gear aksen

$$\phi_{radprpuls} = \frac{2\pi}{20160} P_{pulses} = 0.0003116 P_{pulses} \tag{9}$$

$$p_{cm_{pprpuls}} = \frac{2\pi}{20160} r P_{pulses} = 0.001433663314 P_{pulses} \tag{10}$$

3 Regulator design

3.1 Model

En model af DC motor på en given konstruktion kan tilnærmes med en lineiriseret DC model som vidst på figur 3. Modellen simulerer hvordan en ankerspænding over motoren beskrevet som et filter, genererer en strøm I_a som skaber et drejninsmoment. Motoren vil have et intermoment J på aksen alt afhængingt af konstruktionen, tilmed forekommer dynamisk og statisk friktion.

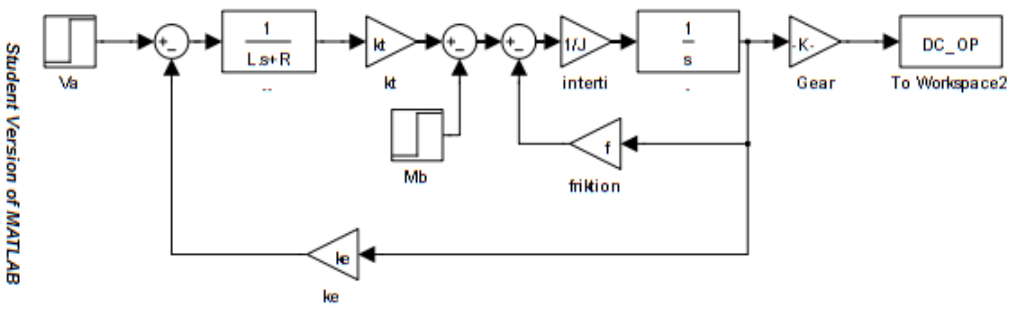


Figure 3: Lineiriseret DC model

Overføringsfunktionen for DC modellen kan bestemmes til at være

$$T_{s_{model}} = \frac{\frac{k_t}{RJ}}{s + \frac{1}{J}(f + \frac{k_t k_e}{R})} \tag{11}$$

Hvilket viser en 1 ordens system. Ved steady state analyse antages omdrejningshastigheden ω_0 at være konstant og ud fra modellen fås derfor

$$V_a - I_a R = \omega_0 k_e \Rightarrow k_e = k_t = \frac{V_a - I_a R}{\omega_0} \tag{12}$$

Det antages tilmed, at i steady state vil friktionen f være lige så stor som det moment motoren leverer. Ud fra denne antagelse fås

$$I_a k_t = f\omega_0 \Rightarrow f = \frac{I_a k_t}{\omega_0} \tag{13}$$

Fra (11) kan polen i modellen direkte aflæses og derpå kan inertimomentet bestemmes som

$$\frac{1}{\tau} = \frac{1}{J} \Rightarrow J = \tau(f + \frac{k_t k_e}{R}) \tag{14}$$

Fra (11) kan den statiske forstærkning bestemmes og fås til

$$Ks = \frac{k_t}{Rf + k_t k_e} \tag{15}$$

For at bestemme polens tidskonstant fra (14) i DC modellen første orden system fortages step repsonser på hhv 15,20,25,30 Rad/Sek med en samplefrekvens på 1KHz over 1 sekund. Derpå kan tidskonstanten aflæses ved 63 procent gain.

På figur 4. ses måleresultater af det fire step responser. Som omtalt i afsnit 2.4 svinger periodetiden af enkoderen på ca 1-2 μs og derved opstår der støj i hastighedsmåledata. Nedenstående måling er derfor midlet over sampleperiode. På trods af midling er støjen stadig fremtrændnde men i mindre grad. Måletøjen kunne undertrykket endnu mere ved en udføring af et lavpasfilter. Det ses også, at der forekommer hastighedspeaks ved opstart, dette kunne skyldes, at step responserne er blevet foretaget på en glat overflade ved relativ høj acceleration.

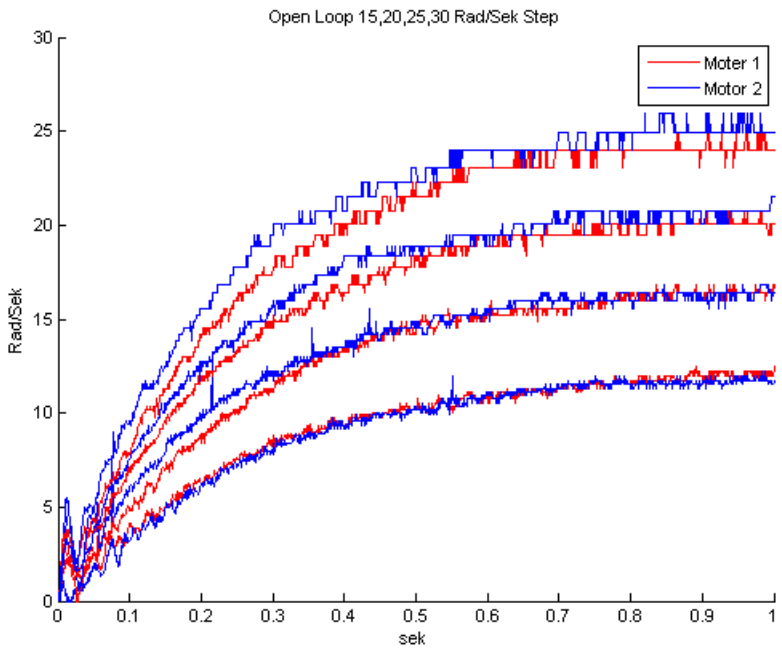


Figure 4: Step repsonses

Det ses på figuren at der forekommer rimelig ens tidskontanter og at der i alle tilfælde ikke opnås ønskede steady state gain. Tilmed varierer hastigheden mellem motorerne, især ved højere hastigheder. Disse problemer kunne forbedres med en hastighedsregulator. Tidskonstaten estimeres til $\tau = 0.24s$. I tabel 2. ses måleresultater af anker strøm I_a og spænding V_a ved konstant hastighed på 30 rad/sekund, motorernes modstand R og tidskonstant samt tilhørende pol.

τ	a	V_a	I_a
0.24	4.1667	$10.84V$	$95mA$

Table 2: Model Målte parametre

Ud fra disse målinger kan de ukendte parametre udledt i (11)-(15) bestemmes, resultatet ses i tabel 3.

k_e	J	f	K_s	b
$0.0226\frac{V}{\frac{rad}{sek}}$	$9.7011 * 10^{-6} kgm^2$	$5.1011 * 10^{-6}$	$38.7454mag$	161.4391

Table 3: Model Beregnet parametre

Derpå bliver overføringsfunktionen for hver DC model med belastning

$$T_{s_{model}} = \frac{161.4391}{s + 4.1667} \tag{16}$$

I figur 5. sammenholdes model og måledata ved de samme fire step responser. Det ses at modellen følger data rimeligt, måledata gain er dog en smule mindre i forhold til model, hvilket kan skyldes at modellen ikke tager højde for motor columb friktion.

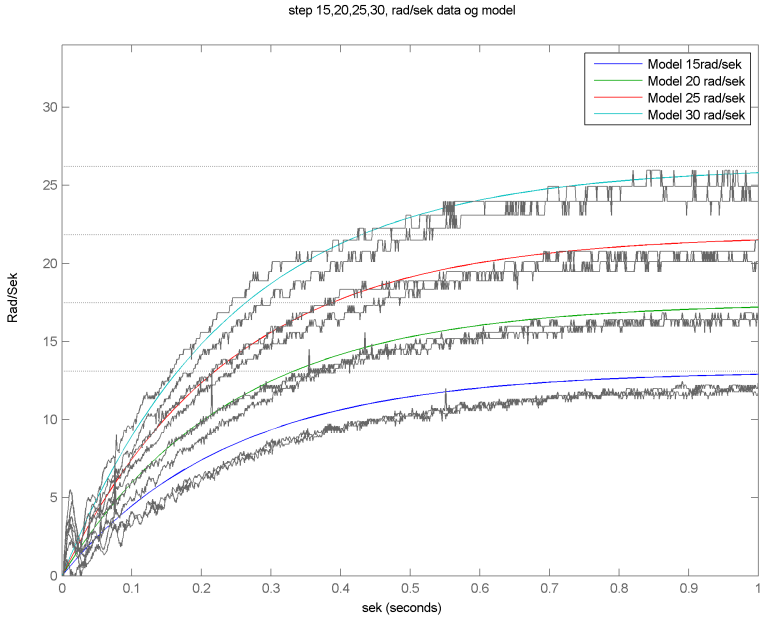


Figure 5: Lineiriseret DC model

3.2 Hastighedsregulator

For at robotten ikke drejer uønsket må der være konstant ens hastighed på begge hjul og for at opnå dette designes en hastighedsregulatorer. Først kan der designes en p- regulator ud fra en steady state error $e_{ss} = 0.05$ ved 10.84 reference. Idet at DC modellen ingen frie integratorer har fås kp ved

$$ess = \frac{V_a}{1 + K_0}, K_0 = k_p K_s \Rightarrow \tag{17}$$

$$kp = (\frac{V_a}{e_{ss}} - 1)(\frac{b}{a}) = (\frac{10.84V}{0.05} - 1)(\frac{161.4391}{4.1667}) = 0.4904 \tag{18}$$

På figur 6. vises åben(øverst)/lukket(nederst) sløjfe bode plot af modellen med kp regulatoren. Fasemargin er 93 og der forekommer et lukket sløjfe steadystate gain på -0.5 dB som følge af kp stationære fejl. Tilmed er båndbredde på ca. 72 rad/sek, så en samplerate på 1KHz skulle være rigeligt.

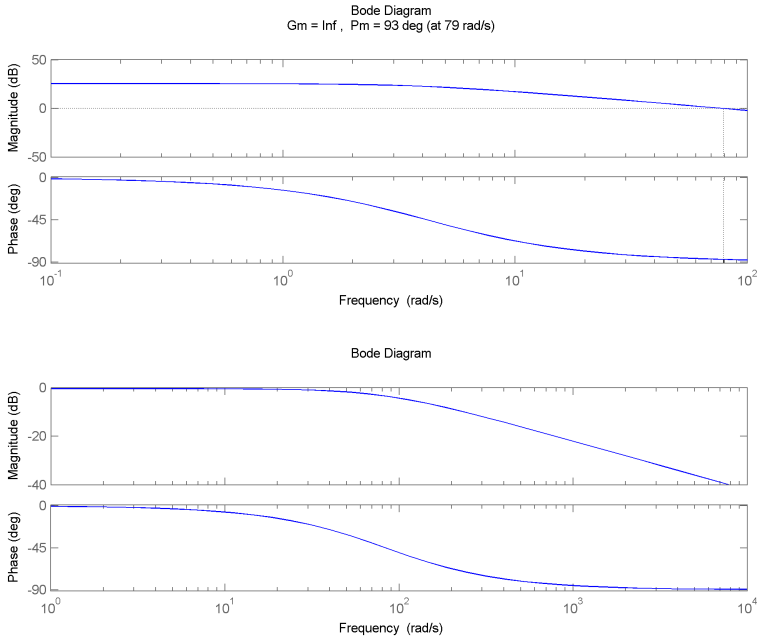


Figure 6: Åben og lukket sløjfe bode plot på model

Da en steady state fejl i det her tilfælde er uønsket forsøges designet en PI-regulator. På figur 6. aflæses krydsfrekvensen til $\omega_c = 79.9$ rad/sek. Derpå dimensioneres PI regulatoren ved, at sætte nulpunktet knæk ω_k 4 gange så lavt som ω_c for at bestemme tidskonstanten til regulatoren. Resulatatet bliver da

$$T(s)_{Pi} = kp\frac{\tau_i s + 1}{\tau_i s}, \omega_k = \frac{\omega_c}{4}, \tau_i = \frac{1}{\omega_k} = 0.0506, T(s)_{Pi} = \frac{0.02483s + 04904}{0.5063s} \tag{19}$$

PI-regulatorens diskrete overføringsfunktion bestemmes ved tustin approximation og modellens ved zero order hold i og ses i (20)

$$T(z)_{Pi} = \frac{0.4951 - 0.4855z^{-1}}{1 - z^{-1}}, T(z)_{z_{model}} = \frac{0.1611z^{-1}}{1 - 0.9958z^{-1}} \tag{20}$$

På figur 7. ses diskret åben sløjfe bode plot af model med PI-regulatoren øverst og lukket sløjfe (kontinueret) nederst. Det ses i åben sløjfe bode plot at fasemargin er stabil på 77 grader. På lukket sløjfe aflæses båndbredden til 56 rad/sek hvilket er mere end rigelig da robottens maksimale hastighed er 30 rad/sek.

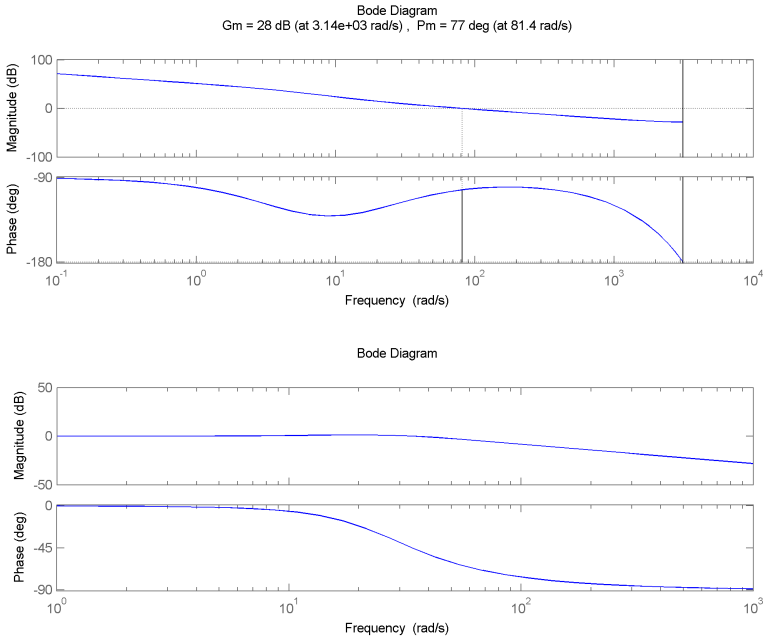


Figure 7: Øverst åben bode diskret model m. PI, lukket bode kontinuert model m. PI

PI-regulatoren implementeres i funktionen VelocControl()(se Appendix B kilde kode) ved direct realization. På figur 8. ses step responser på hhv 15,20 rad/sek på lukket sløjfe model og måledata med PI-regulatoren på.

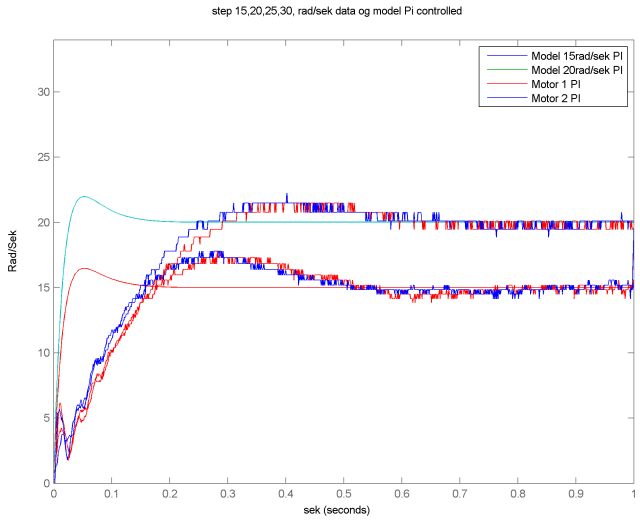


Figure 8: Lineiriseret DC model

Det ses på figuren at begge motorer når deres reference hastighed med rimelig ens hastighedsforløb som ønsket, der er dog lidt oversving dette kunne skyldes den høje acceleration. Der kunne justeres på kp for at

mindste dette. Det ses tilmed at modellen er hurtigere end måledata, dette kunne skyldes at modellen ikke tager højde for begrænsning i output fra regulatoren. Der kunne indsættes en begrænser for at opnå et mere sammenfaldende resultat.

3.3 Positionsregulator

Der ønskes også opnået en given position med en vis præcision, f.eks ved, at robotten skulle køre til given position eller en dreje en given vinkel. For at opnå bedre positionsresultater designes også en position regulator. Motorernes position kan igen beskrives ud fra den liniariseret DC model som ses i figur 9.

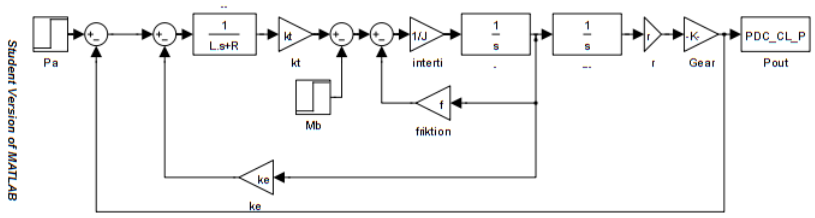


Figure 9: Lineiriseret DC model position

Overføringsfunktionen for modellen bliver

$$T(s)_{pmodel_{open}} = \frac{53.04}{s^2 + 4.1667s}, T(s)_{pmodel_{closed}} = \frac{53.04}{s^2 + 4.1667s + 53.04} \tag{21}$$

På figur 10. ses åben sløjfe bode plot på positionsmodellen øverst. Det ses at system har en ringe fasemargin på 31.8 grader, derfor designes en kp regulator således at der opnås en fasemargin på 60 grader. Dette gøres ved at sænke gain -13.6 dB som opnås ved multiplisering med $Kp_{position} = 0.2089$. Nederst på figur 10. ses åben sløjfe bode plot med kp regulator.

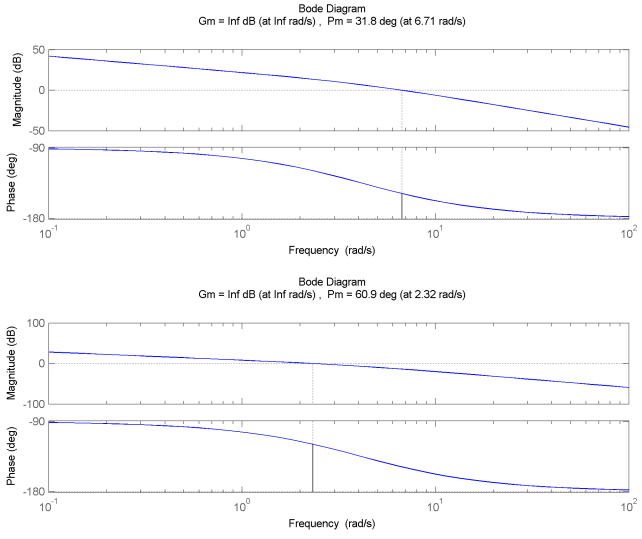


Figure 10: position åben sløjfe bode plot m/u kp regulator

Øverst på figur 11. ses lukket sløjfe bode plot uden kp og som følge er den ringe fasemargin opstår resonans top omkring knækfrekvensen. Denne resonans top er fjernet ved kp regulatoren, som ses i i figur 11. nederst. Det ses at båndbredden er relativ lav og aflæst til ca. 9 rad/sek.

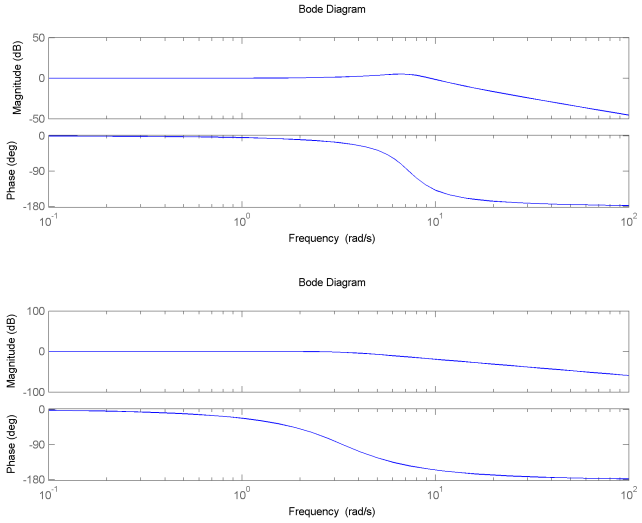


Figure 11: position lukket sløjfe bode plot m/u kp regulator

På figur 12. ses diskret(zoh) åben sløjfe bode plot af model med p-regulatoren. Det ses at fasemargin er stabil på 60 grader.

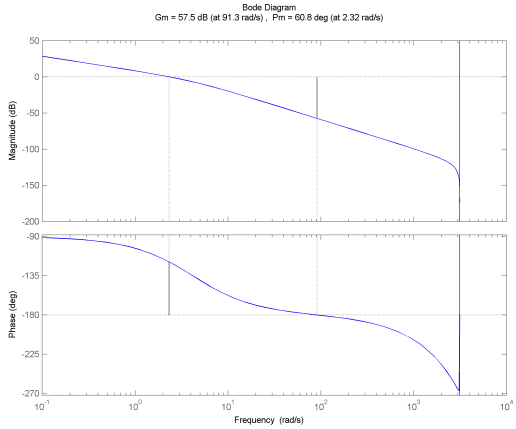


Figure 12: Lineiriseret DC model

kp regulatoren er implementeret i funktionen PositionControl() og i figur 13. ses et step på 50 cm på model og måledata med kp=1 og kp=0.02089 på model og måledata.

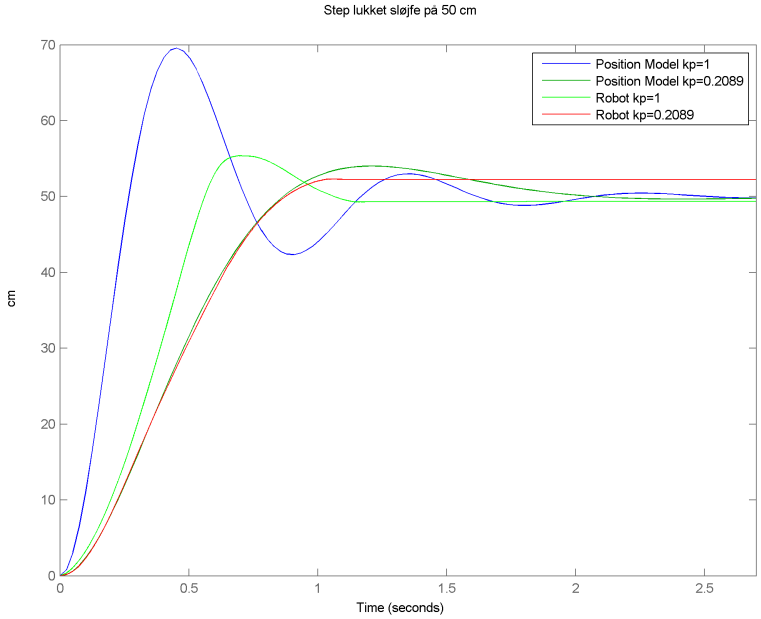


Figure 13: 50 cm step på position model og måledata m. kp=0.2089 og kp=1

Det ses først og fremmest at modellens oscillation tilmed oversving er mindsket ved kp regulatoren. Generelt passer model og måledata ikke sammen her, grunden kunne være at modellen ikke tager højde for begrænsning. Ved aflæsning fås der en steady state fejl på måledata på ca 2.5 cm som følge af kp=0.2089, derimod er modellens betydelig mindre. Umiddelbart ses det, at måltedata med kp=1 ser mest fornuftigt ud en lille steady state fejl, dog forekommer der stadig et oversving. I afsnit 4. vil der kigges på hvorledes hastigheds-, positions og accelerationsregulatorer i sammenspil kan mindske oversving.

3.4 Accelerationregulator

Positions- og hastighedsregulatorens forstærkning vil være størst når forskellen mellem ønsket reference og aktuel måleværdi er størst, hvilket vil resulterer i unødvendige overaccelerationer. For at overkomme dette problem kan der designes en simple ulinær accelerationsregulator som kan erstatte positions- og hastighedsregulatorer når deres forstærkninger er meget store. Regulatoren designes som følgende.

Antaget konstant vinkelacceleration α over en given afstand $\varphi - \varphi_0$ med slut vinkelhastighed $\omega = 0$, kan følgende generelt udtrykkes

$$\omega^2 - \omega_0^2 = 2\alpha(\varphi - \varphi_0) \Rightarrow \alpha = -\frac{\omega_0^2}{2(\varphi - \varphi_0)}, \varphi = \frac{p}{r} \tag{22}$$

Ud fra overstående bestemmes vinkelhastigheden som funktion af nuværende position over en de-accelerations distance i (23). Denne funktion er implementeret i AccelerationControl().

$$\omega(\varphi_{current}) = \sqrt{\omega_0^2 - 2\alpha(\varphi_{current} - (\varphi_{destination} - \varphi_{accstart}))}, \tag{23}$$

$$\varphi_{current}[\varphi_{destination} - \varphi_{accstart} : \varphi_{destination}] \tag{24}$$

Følgende beregninger er fortages ved 50 cm positionsreference, 20 cm de-acceleration ved en opstartshastighed på 15 rad/sek, derved fås følgende beregninger

$$\alpha = -\frac{15\frac{rad}{sek}}{2\frac{20cm}{4.6cm}} = -25.875s^{-1}, \omega(\varphi_{current}) = \sqrt{225 - 2 * 25.875(\varphi_{current} - 30/4.6)} \quad (25)$$

$$\varphi_{current}[30:50] \quad (26)$$

4 Optimeret Styring

4.1 Regulator kombination

For at få en mere optimal og præcis styring kan de 3 regulatorer designet i afsnit 3. kombineres. Frem mod en ønsket destination er det hensigtmæssigt med en kontrolleret acceleration idet, at hastigheds- og positions regulatorer som nævnt vil have stor forstærkning langt fra referencen. I det følgende vil der blive designet en kombinationsstyring.

På figur 14. ses det designede styringssystem indeholdende hastigheds-,positions- og accelerationsregulatorer som alle benyttes i kombination til at opnå en ønsket position bedst muligt.

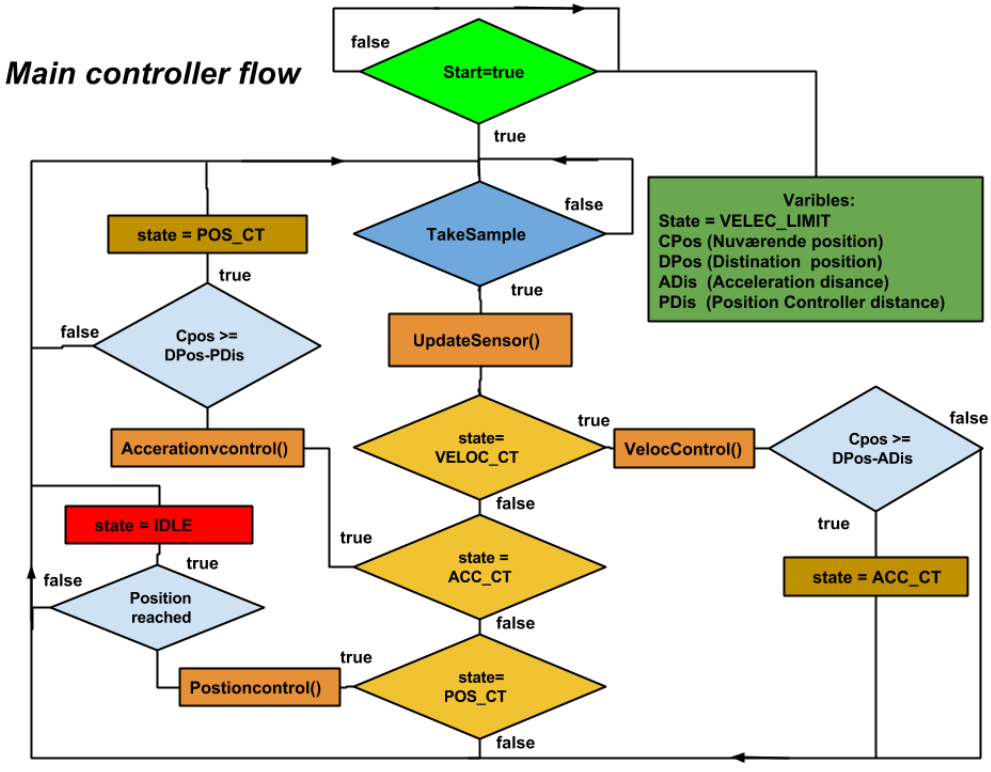


Figure 14: Flow over MainController()

Figuren viser et flow over funktionen MainController(). Ved opstart kunne hastighedsregulatorens forstærkning begrænses i VelocControl() endtil robotten har opnået en hastighed der er tættere på

referenen, således undgås for høj acceleration ved opstart. Ved opnået reference benyttes hastighed regulatoren fortsæt således ,at der bliver konstant ens vinkelhastighed, dette indtil der ønskes de-acceleration bestemt af ADis parametren. Når de-acceleration positionen nås, skiftes der til accelerationsregulatoren fra funktionen Acceleration(). Denne sænker hastigheden med konstant de-acceleration frem mod mål, således at der kan opnås en blød overgang til positionregulatoren hvilken benyttes fra DPos.

4.2 Resultater

Resultater af overstående beskrevet styring vises i figur 15. Der er fortages 5 step responser på 50 cm, hvor kp i positionsregulatoren varieres til hhv 1,0.2089,1.3 og 1.6. Der er også tilføjet en reference (lyserød) hvor der kun er brugt positionsregulator. Accelerationsregulatoren er tages i brug 20 cm før destination og positionsregulatoren 5 cm før destination.

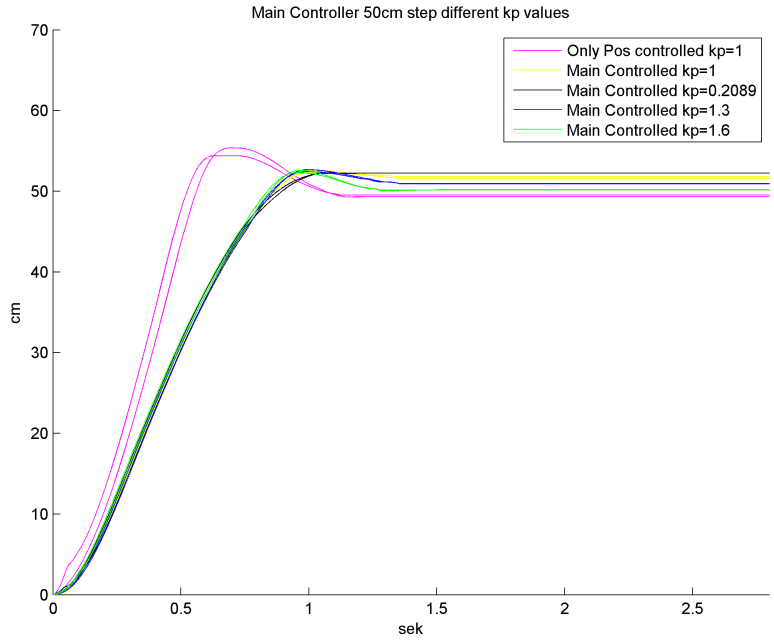


Figure 15: Resultater af regulator kombinationsstyring

I tabel 4. ses de samlet step response parametre somså overshoot og steady state fejl taget udgangspunkt i figur 15.

	$k_{pref} = 1$	$k_p = 1$	$k_p = 0.2089$	$k_p = 1.3$	$k_p = 1.6$
SS fejl cm	0.35-0.55	1.6-1.8	2.22	0.91-0.93	0.12-0.14
Overshoot cm	4.4-5.36	2.28- 2.58	0.01	2.43-2.64	2.28-2.57

Table 4: Model Beregnet parametre

Først og bemærkes at overshoot er reduceret i alle de tilfælde hvor kombinationskontrolleren er brugt. Dette er som følge af, at skifte regulatorer ved passende hastighed eller position således at fejl forstærkning

bliver minimeret. I tilfældet med kp=0.2089 er næsten alt overshoot fjernes som følge af bedre fasemargin, til gengæld fås en større stationær fejl. Ved kp=1 stiger overshoot men til gengæld opnås en mindre steady state fejl. Ved endnu større kp=1.6 vokser over ikke betydere og der er opnået det bedste steady state resultat. Der kunne senere hen forsøges dimensioneret en PI eller PID regulator for bedre resultat. Tilmed kunne der eksperimenteres med forskellige de-accelerationer samt regulator skift afstande.

5 Konklusion

Phidgets motorerne blev valgt på baggrund af deres passende moment på 12 kgcm, top hastighed på 285 rmp samt 20120 puls/rev enkoder opløsning hvilket udgør en fornuftig motor til robocup. En teensy mikrokontroller 3.1 blev valgt på baggrund dens kompakte størrelse, 75MHz clockrate samt floating point. En pololu switchmode konverter blev valgt til 5V forsyning på baggrund af den høje effektivitet og 3.5A strøm levering. L293d 0.6A/motor blev benyttes selvom denne ikke var en optimal driver idet at motorernes stall strøm lå på 1.5A. Tilmed vidste det sig ved beregning at driverens maks moment var 0.18984 kgcm, hvilket lå lang under stall torque samt 25 graders rappe torque. Enkoderen svingede ca. 1-2 μs hvilket gave støj i hastighedsmåliger, denne støj blev mindsket ved midling, men støj var stadig fremtrændende i mindre grad. Et lavpasfilter kunne derfor være nødvendigt. En motor model blev opnået med en lineariseret DC-model og var sammenfaldende med måledata ved step responser set i figur 5. Motorerne her kunne ikke nå sin reference hastighed ureguleret, men ved design af PI hastighedregulator blev en der opnået en stabilt reference samt ens hastighedsforløb, dog lidt oversving (figur 8). En Kp=0.2089 positionsregulator blev designet ved en 60 grader fasemargin resultatet gav i praktik en 2.5 cm steady state. Der var tilmed svag sammenhæng model og måledata her. Et forbedret resultat blev opnået med kp=1.6 fundet ekperimentielt, tilmed alle regulator i kombination således overacceleration blev mindsket (figur 15). Således blev det bedste resultat opnået på 0.12 - 0.14 cm steady state fejl samt oversving på 2.28-2.57 oversving med kp=1.6.

6 Appendix

6.1 Appendix A

6.2 Appendix B

```
1 #include <stdlib.h>
2 #include <usb_serial.h>
3 #include "IntervalTimer.h"
4 #include "serial_com.h"
5 #include "IntervalTimer.h"
6 #include "String.h"
7
8 #define LED 13
9 #define FORWARD 1
10 #define BACKWARD 0
11 #define ENABLED 1
12 #define DISABLED 0
13
14 //Parameters
15 #define PULSES_PR_REV 20160    // 360*14*4
16 #define RADIUS_CM 4.6      //
17 #define BIT_RESOLUTON 256      // can be changed to 4096
18 #define VOLT_MAX 10.84       // Max 10.84 Volt
```

```
19 #define VOLT_MIN 2          // Min 2 Volt
20 #define REVOLUTION_CM 28.87
21 #define Ke 0.3157           // ke*ratio
22 #define POSLOG_SIZE 1600
23 #define LOGSIZE 3000
24
25
26 //---------- Motor 1 ports----------//
27 #define GREEN_M2 20
28 #define BLUE_M2 22
29 #define PUPLE_M2 18
30
31 #define CH_A_M2 17
32 #define CH_B_M2 15
33
34 //---------- Motor 2 ports----------//
35 #define GREEN_M1 9
36 #define BLUE_M1 10
37 #define PUPLE_M1 12
38
39 #define CH_A_M1 6
40 #define CH_B_M1 8
41
42 //----------Encoder----------//
43 typedef struct{
44   uint8_t Interval_ENA = 0;
45   uint16_t IntervalSum_ENA;
46   uint16_t IntervalNum_ENA;
47   uint64_t CurrentTime_ENA;
48   uint64_t LastTime_ENA;
49   uint8_t Cycle_EN_A;
50   uint8_t Cycle_EN_B;
51   signed int PosTicks;
52 }EncoderParams;
53
54 EncoderParams Enco_M1;
55 EncoderParams Enco_M2;
56
57
58 //----------Motor Params----------//
59 typedef struct{
60   uint8_t Direction;
61   uint64_t Position;
62   uint8_t Volt;
63   uint16_t Speed;
64   float ControlOut;
65   uint8_t Enable;
66   bool ReachedPos;
67 }MotorParam;
68
69 MotorParam SetM1;
```

```
70 MotorParam SetM2;
71
72 //--------Encoder calculation--------//
73 typedef struct{
74   uint16_t    Periode_micros;
75   uint16_t    PeriodeSum_micros;
76   uint16_t    PeriodeCount_micros;
77   float   PeriodeAvg_micros;
78   float   RotVelocity_RPM_;
79   float   RotVelocity_RadprS;
80   float   CPosition;
81   double PeriodeCycle;
82   float timeSek;
83 }EncoderCalcOneCycle;
84
85 EncoderCalcOneCycle M1_FBack;
86 EncoderCalcOneCycle M2_FBack;
87
88 //--------Conversion Factors--------//
89 typedef struct{
90   float Rad_pr_cycle;
91   float CM_pr_puls;
92   float RadPS_conv_micros;
93   float RPM_conv_micros;
94 }Factors;
95 Factors CFacts;
96
97 //--------Interface Params----------//
98 typedef struct{
99   uint32_t SampleSize;
100  float SampleTime;
101  uint32_t SampleLength;
102  uint8_t AvgSample1;
103  uint8_t AvgSample2;
104  uint8_t UseLowPassfilter;
105  uint8_t VelocityUnit;
106  uint8_t Mulistep;
107  uint8_t LiveMission;
108  uint8_t DecreaseVal;
109  uint8_t Control;
110  uint8_t VelocRef;
111  uint8_t DirectionCalc;
112  uint16_t CAccDistance;
113  uint16_t CPosDistance;
114  float DPosition;
115  float PosConVoltMax;
116  float PosConVoltMin;
117  float Kp_Pos;
118 }InterfaceParameters;
119 InterfaceParameters UserP;
120
```

```
121 //--------Controller Params----------//
122 typedef struct{
123   float a1;
124   float b0;
125   float b1;
126   float e;
127   float dummy;
128   float u;
129   float RegOut;
130 }ControllerParams;
131
132 ControllerParams M1_CP_veloc;
133 ControllerParams M2_CP_veloc;
134 ControllerParams M1_CP_Pos;
135 ControllerParams M2_CP_Pos;
136
137 typedef struct{
138   float RadPrSek;
139   float Position;
140 }LogParam;
141
142 float error=0;
143
144 LogParam M1_Log[LOGSIZE];
145 LogParam M2_Log[LOGSIZE];
146
147 typedef struct{
148   float alfa;
149   float LastRadprSek;
150   float Sampletime;
151   float CutOff;
152 }LowPassFilterParms;
153
154 LowPassFilterParms M1_LPfilter;
155 LowPassFilterParms M2_LPfilter;
156
157 enum VelocityUnits{
158   RAD_PR_SEK=0,
159   RPM,
160   VOLT,
161   KM_PR_SEK,
162   M_PR_SEK
163 };
164 enum DistanceUnits{
165   CM=0,
166   METERS,
167   KM
168 };
169 enum ProgramStates{
170   IDLE=0,
171   STEPRESPONSE,
```

```
172     MISSION,
173     TEST,
174     STOP
175 };
176 uint8_t state;
177 enum StepResponsesStates{
178     RESET_TIMER,
179     START_STEP,
180     START_INTERFACE_LOG
181 };
182 uint8_t StepState;
183 enum MissionStates{
184     MIS_RESET =0,
185     MIS_RUN,
186 };
187 uint8_t MissionState;
188 uint8_t Missiontype;
189
190 enum ControlStates{
191     VELOCITY_CONTROL_S=0,
192     ACCELERATION_CONTROL_S,
193     POSITION_CONTROL_S
194 };
195 uint8_t ControlState;
196
197 enum InterfaceCommands{
198     NON_SET =0,
199     INTERFACE_SET,
200     INTERFACE_GET,
201     INTERFACE_MISSION,
202     INTERFACE_STEP,
203     STEP_VELOCITY,
204     MISSION_TEST,
205     MISSION_STOP,
206     MISSION_TRESHOLD,
207     MISSION_POSITION,
208     MISSION_POSITION_CONTROLLED,
209     MISSION_DEFAULT,
210     SET_SAMPLE_SIZE,
211     SET_SAMPLE_LENGTH,
212     SET_SAMPLE_AVGENABLE1,
213     SET_SAMPLE_AVGENABLE2,
214     SET_LPFILTER,
215     SET_RESET,
216     SET_CONTROL,
217     SET_VELOC_REF,
218     SET_POSCONVOLT_MIN,
219     SET_POSCONVOLT_MAX,
220     SET_KP_POS,
221     SET_DIR_CALC,
222     STEP_MULTISTEP,
```

```
223   SET_MOTOR_VOLT1,
224   SET_MOTOR_VOLT2,
225   SET_MOTOR_DIR1,
226   SET_MOTOR_DIR2,
227   SET_MOTOR_POS1,
228   SET_MOTOR_POS2,
229   SET_MOTOR_VELOC1,
230   SET_MOTOR_VELOC2,
231   SET_MOTOR_VELOCUNIT,
232   SET_MOTOR_ENABLE1,
233   SET_MOTOR_ENABLE2,
234   SET_MOTOR_DESTINATION
235 };
236
237
238 enum ControllerSwith{
239   NON_CONTROL=0,
240   VELOC_CONTROL,
241   POS_CONTROL,
242   MAIN_CONTROL
243 };
244
245
246
247 //---------------Variables-------------//
248
249 //LOG
250 uint32_t LogNum;
251 float time;
252 //Timing
253 IntervalTimer SampleTimer;
254 IntervalTimer UserInterface;
255 bool GetSample = false;
256 bool DoTresholdMission;
257 bool GetUpdateInterface = false;
258 bool UpdateUserInterface = false;
259 uint64_t tick_micros;
260 uint16_t PosLogCount=0;
261 float alfa=0;
262
263 ////////////////////////////////////////////////////////////////////////
264 //!   Prototyping : se decription in function implementation
265 ////////////////////////////////////////////////////////////////////////
266 void InitConvFactors();
267 void InitVaribles();
268 void SetMotorEnable(bool EN_M1,bool EN_M2);
269 void SetMotorDirection(int DIR_M1,int DIR_M2);
270 void SetMotorVelocity(int V_M1,int V_M2);
271 void SetMotorPosition(float P_M1, float P_M2);
272 void UpdateMotor();
273 void UpdateSensor();
```

```
274 void UpdateLogVeloc();
275 void Test();
276 void Stop();
277 void Treshold();
278 void DoMission();
279 void DoLiveMission();
280 void Reset();
281 void StepReponse();
282 void MainControl();
283 void VelocControl();
284 void PositionControl();
285 void VelocControlInit();
286 void PositonControlInit();
287 void AccelerationControl();
288 void UpdateMotorControl();
289 void UpdateMultiStep(uint8_t interval);
290 void ZeroAvgValues();
291 float LowPass(float);
292 void InitLowPass();
293 void ISR_SAMPLE(void);
294 void ISR_USER_INTERFACE(void);
295 void ISR_CH_A_M1();
296 void ISR_CH_B_M1();
297 void ISR_CH_A_M2();
298 void ISR_CH_B_M2();
299 void Interface();
300 void InterfaceSet(uint8_t action, uint8_t target,long int value);
301 void InterfaceLog();
302 void WriteSensorInterface();
303
304 // Conversions
305 float    AngularVelocity(int CycleTimeMikros,int unit);
306 uint16_t PWMVoltage(float Voltage);
307 float    VoltagePWM(int DutyCycle);
308 float    CyclesDistance_CM(int Cycle);
309 uint64_t DistanceCycles_CM(float distance,int unit);
310 uint16_t UpDatePWM(uint16_t Rad_pr_Sek);
311
312 void initialization() {
313
314   InitConvFactors();
315   InitVaribles();
316
317   //Debug LED
318   pinMode(LED, OUTPUT);
319   digitalWrite(LED, HIGH);
320
321   // Motor 1 ports init
322   pinMode(GREEN_M1, OUTPUT);
323   pinMode(BLUE_M1, OUTPUT);
324   pinMode(PUPLE_M1, OUTPUT);
```

```
325
326   pinMode(CH_A_M1, INPUT);
327   digitalWrite(CH_A_M1, LOW);
328   attachInterrupt(CH_A_M1, ISR_CH_A_M1,CHANGE);
329
330   pinMode(CH_B_M1, INPUT);
331   digitalWrite(CH_B_M1, LOW);
332   attachInterrupt(CH_B_M1, ISR_CH_B_M1,CHANGE);
333
334
335   // Motor 2 ports init
336   pinMode(GREEN_M2, OUTPUT);
337     pinMode(BLUE_M2, OUTPUT);
338     pinMode(PUPLE_M2, OUTPUT);
339
340   pinMode(CH_A_M2, INPUT);
341   digitalWrite(CH_A_M2, LOW);
342   attachInterrupt(CH_A_M2, ISR_CH_A_M2,CHANGE);
343
344   pinMode(CH_B_M2, INPUT);
345   digitalWrite(CH_B_M2, LOW);
346   attachInterrupt(CH_B_M2, ISR_CH_B_M2,CHANGE);
347
348   //analogWriteResolution(12);
349
350   SampleTimer.begin(ISR_SAMPLE, UserP.SampleTime);
351   UserInterface.begin(ISR_USER_INTERFACE,500000);
352   Serial.begin(115200);
353 }
354
355 extern "C" int main(void) {
356
357   initialization();
358
359   while (1) {
360     switch(state){
361       case IDLE:{        // IDLE updating interface
362         Interface();
363         break;
364       }
365       case STEPRESPONSE:{    // State for sampling step response data
366         if(GetSample){     // Controled sample timer
367           StepReponse();
368           GetSample=false;
369         }
370         break;
371       }
372       case MISSION:{
373         if(GetSample){     // Do a misstion
374           DoMission();
375           GetSample=false;
```

```
376           }
377           break;
378         }
379         case TEST:{
380           Test();
381           break;
382         }
383         case STOP:{
384           Stop();
385           break;
386         }
387       }
388       // Main loop delay so processor can keep up
389       delayMicroseconds(1);
390
391     }
392 }
393
394
395
396 //------------------- Function implementation---------------------------//
397
398
399 ////////////////////////////////////////////////////////////////////////
400 //! Purpose: In idle state handling interface communication
401 //! Params  :
402 ////////////////////////////////////////////////////////////////////////
403 void Interface(){
404   char inChar;
405   uint8_t action =0;
406   uint8_t target =0;
407   long int value =0;
408   String InterfaceResponse="";
409
410   while (Serial.available()) {
411       inChar = (char)Serial.read();
412       if(inChar=='.' || inChar=='\n' || inChar==':'){
413
414         // Checking for prefered state
415         if(InterfaceResponse=="set"){
416           action = INTERFACE_SET;
417         }
418         else if(InterfaceResponse=="mission"){
419           action= INTERFACE_MISSION;
420
421         }
422         else if(InterfaceResponse=="step"){
423           action= INTERFACE_STEP;
424         }
425
426         //According to prefered state
```

```
427         else if(action== INTERFACE_STEP){
428           if(InterfaceResponse  =="velocstep"){
429             target = STEP_VELOCITY;
430             InterfaceSet(action,target,value);
431             break;
432           }
433         }
434         else if(action== INTERFACE_MISSION){
435           if(InterfaceResponse  =="test"){
436              target = MISSION_TEST;
437              InterfaceSet(action,target,value);
438              break;
439           }
440           else if(InterfaceResponse  =="stop"){
441              target = MISSION_STOP;
442              InterfaceSet(action,target,value);
443              break;
444           }
445           else if(InterfaceResponse  =="treshold"){
446              target = MISSION_TRESHOLD;
447              InterfaceSet(action,target,value);
448              break;
449           }
450           else if(InterfaceResponse  =="position"){
451              target = MISSION_POSITION;
452              InterfaceSet(action,target,value);
453              break;
454           }
455           else if(InterfaceResponse  =="positioncontrolled"){
456               target = MISSION_POSITION_CONTROLLED;
457               InterfaceSet(action,target,value);
458               break;
459           }
460         }
461         else if(action== INTERFACE_SET){
462           if(InterfaceResponse  =="samplesize"){
463                target = SET_SAMPLE_SIZE;
464           }
465           else if(InterfaceResponse  =="samplelenght"){
466                target = SET_SAMPLE_LENGTH;
467           }
468           else if(InterfaceResponse  =="Sampleavgenable1"){
469                target = SET_SAMPLE_AVGENABLE1;
470           }
471           else if(InterfaceResponse  =="Sampleavgenable2"){
472                target = SET_SAMPLE_AVGENABLE2;
473           }
474           else if(InterfaceResponse  =="uselpfilter"){
475                target = SET_LPFILTER;
476           }
477           else if(InterfaceResponse  =="reset"){
```

```
478                 target = SET_RESET;
479             }
480             else if(InterfaceResponse ==”destination”){
481                  target = SET_MOTOR_DESTINATION;
482             }
483             else if(InterfaceResponse ==”multistep”){
484                  target = STEP_MULTISTEP;
485             }
486             else if(InterfaceResponse ==”control”){
487                  target = SET_CONTROL;
488             }
489             else if(InterfaceResponse ==”velocref”){
490                  target = SET_VELOC_REF;
491             }
492             else if(InterfaceResponse ==”posconvoltmax”){
493                  target = SET_POSCONVOLT_MAX;
494             }
495             else if(InterfaceResponse ==”posconvoltmin”){
496                  target = SET_POSCONVOLT_MIN;
497             }
498             else if(InterfaceResponse ==”kppos”){
499                  target = SET_KP_POS;
500             }
501             else if(InterfaceResponse ==”dirposcalc”){
502                  target = SET_DIR_CALC;
503             }
504             else if(InterfaceResponse ==”motorvolt1”){
505               target = SET_MOTOR_VOLT1;
506               }
507             else if(InterfaceResponse ==”motorvolt2”){
508                 target = SET_MOTOR_VOLT2;
509               }
510               else if(InterfaceResponse ==”motordir1”){
511               target = SET_MOTOR_DIR1;
512             }
513               else if (InterfaceResponse == ”motordir2”) {
514               target = SET_MOTOR_DIR2;
515             }
516               else if(InterfaceResponse ==”motorpos1”){
517               target = SET_MOTOR_POS1;
518             }
519             else if (InterfaceResponse == ”motorpos2”) {
520               target = SET_MOTOR_POS2;
521           }
522             else if(InterfaceResponse ==”motorveloc1”){
523                 target = SET_MOTOR_VELOC1;
524             }
525             else if (InterfaceResponse == ”motorveloc2”) {
526                  target = SET_MOTOR_VELOC2;
527             }
528             else if (InterfaceResponse == ”motorvelocunit”) {
```

```
529                 target = SET_MOTOR_VELOCUNIT;
530           }
531           else if(InterfaceResponse ==”motorenable1”){
532                 target = SET_MOTOR_ENABLE1;
533           }
534           else if (InterfaceResponse == ”motorenable2”) {
535                 target = SET_MOTOR_ENABLE2;
536           }
537             else if(inChar=='\n'){
538               value = InterfaceResponse.toInt();
539               InterfaceSet(action,target,value);
540           }
541           else if(inChar==':'){
542              action =0;
543              break;
544           }
545         }
546
547         InterfaceResponse="";
548       }
549       else{
550         InterfaceResponse +=inChar;
551       }
552   }
553
554   // If no command requests just send data: This updated every 2 second
555   if(GetUpdateInterface){
556     GetUpdateInterface=false;
557     WriteSensorInterface();
558     DoLiveMission();
559   }
560
561 }
562 ////////////////////////////////////////////////////////////////////////
563 //! Purpose : Setting Requested parameters or starting missiotn
564 //!         interface()
565 //! Params  : Action = what is to be handled
566 //!         Target = what parameter or mission to set
567 //!         Value =  value of parameter
568 ////////////////////////////////////////////////////////////////////////
569 void InterfaceSet(uint8_t action, uint8_t target,long int value){
570   switch(action){
571     case INTERFACE_SET:
572       switch(target){
573         case SET_SAMPLE_SIZE:
574           UserP.SampleSize    = value;
575           Serial.print(”SampleSize:”);
576           Serial.print(UserP.SampleSize);
577           Serial.print(” ”);
578           break;
579         case SET_SAMPLE_LENGTH:
```

```
580           UserP.SampleLength = value;
581           Serial.print("SampleLength:");
582           Serial.print(UserP.SampleLength);
583           Serial.print(" ");
584           break;
585         case SET_SAMPLE_AVGENABLE1:
586           UserP.AvgSample1 = value;
587           Serial.print("AvgSample1:");
588           Serial.print(UserP.AvgSample1);
589           Serial.print(" ");
590           break;
591         case SET_SAMPLE_AVGENABLE2:
592           UserP.AvgSample2 = value;
593           Serial.print("AvgSample2:");
594           Serial.print(UserP.AvgSample2);
595           Serial.print(" ");
596           break;
597         case SET_LPFILTER:
598           UserP.UseLowPassfilter = value;
599           UserP.AvgSample1 = !value;
600           UserP.AvgSample2 = !value;
601           Serial.print("filter and average");
602           Serial.println(UserP.UseLowPassfilter);
603           Serial.println(UserP.AvgSample1);
604           Serial.println(UserP.AvgSample2);
605           break;
606         case SET_RESET:
607           Reset();
608           Serial.print("Reset");
609           break;
610         case SET_MOTOR_DESTINATION:
611           UserP.DPosition = value;
612           Serial.print("destination:");
613           Serial.print(UserP.DPosition);
614           Serial.print(" ");
615           break;
616         case SET_CONTROL:
617           UserP.Control = value;
618           Serial.print("Control:");
619           Serial.print(UserP.Control);
620           Serial.print(" ");
621           break;
622         case SET_VELOC_REF:
623           UserP.VelocRef = value;
624           Serial.print("Velocity Ref:");
625           Serial.print(UserP.VelocRef);
626           Serial.print(" ");
627           break;
628         case SET_POSCONVOLT_MIN:
629           UserP.PosConVoltMin = value;
630           Serial.print("Min:");
```

```
631             Serial.print(UserP.PosConVoltMin);
632             Serial.print(" ");
633             break;
634           case SET_POSCONVOLT_MAX:
635             UserP.PosConVoltMax = value;
636             Serial.print("Max:");
637             Serial.print(UserP.PosConVoltMax);
638             Serial.print(" ");
639             break;
640           case SET_KP_POS:
641             UserP.Kp_Pos = value;
642             Serial.print("Kp Pos:");
643             Serial.print(UserP.Kp_Pos);
644             Serial.print(" ");
645             break;
646           case STEP_MULTISTEP:
647             UserP.Mulistep = value;
648             Serial.print("MultiStep:");
649             Serial.print(UserP.Mulistep);
650             Serial.print(" ");
651             break;
652           case SET_DIR_CALC:
653             UserP.DirectionCalc = value;
654             Serial.print("DirectionCalc:");
655             Serial.print(UserP.DirectionCalc);
656             Serial.print(" ");
657             break;
658           case SET_MOTOR_VOLT1:
659             SetM1.Volt = value;
660             Serial.print(SetM1.Volt);
661             Serial.print(SetM1.Volt);
662             Serial.print(" ");
663             break;
664           case SET_MOTOR_VOLT2:
665             SetM2.Volt = value;
666             Serial.print("Motor2Volt:");
667             Serial.print(SetM2.Volt);
668             Serial.print(" ");
669             break;
670           case SET_MOTOR_DIR1:
671             SetM1.Direction = value;
672             Serial.print("Motor1Direction:");
673             Serial.print(SetM1.Direction);
674             Serial.print(" ");
675             break;
676           case SET_MOTOR_DIR2:
677             SetM2.Direction = value;
678             Serial.print("Motor2Direction:");
679             Serial.print(SetM2.Direction);
680             Serial.print(" ");
681             break;
```

```
682           case SET_MOTOR_POS1:
683             SetM1.Position = value;
684             Serial.print("Motor1PositionCM:");
685             Serial.print((uint32_t)SetM1.Position);
686             Serial.print(" ");
687             break;
688           case SET_MOTOR_POS2:
689             SetM2.Position = value;
690             Serial.print("Motor2PositionCM:");
691             Serial.print((uint32_t)SetM2.Position);
692             Serial.print(" ");
693             break;
694           case SET_MOTOR_VELOC1:
695             SetM1.Speed = value;
696             Serial.print("Motor1Velocity:");
697             Serial.print(SetM1.Speed);
698             Serial.print(" ");
699             break;
700           case SET_MOTOR_VELOC2:
701             SetM2.Speed = value;
702             Serial.print("Motor2Velocity:");
703             Serial.print(SetM2.Speed);
704             Serial.print(" ");
705             break;
706           case SET_MOTOR_VELOCUNIT:
707             UserP.VelocityUnit = value;
708             Serial.print("VelocityUnit:");
709             Serial.print(UserP.VelocityUnit);
710             Serial.print(" ");
711             break;
712           case SET_MOTOR_ENABLE1:
713             SetM1.Enable = value;
714             Serial.print("Motor1Enable:");
715             Serial.print(SetM1.Enable);
716             Serial.print(" ");
717             break;
718           case SET_MOTOR_ENABLE2:
719             SetM2.Enable = value;
720             Serial.print("Motor2Enable:");
721             Serial.print(SetM2.Enable);
722             Serial.print(" ");
723             break;
724         }
725         break;
726       case INTERFACE_MISSION:
727         switch(target){
728           case MISSION_TRESHOLD:
729             Serial.print("mission treshold");
730             state = IDLE;
731             UserP.LiveMission = target;
732             break;
```

```
733           case MISSION_TEST:
734             Serial.print("mission start");
735             state = TEST;
736             break;
737           case MISSION_POSITION:
738             state = MISSION;
739             Missiontype = MISSION_POSITION;
740             break;
741           case MISSION_POSITION_CONTROLLED:
742             state = MISSION;
743             Missiontype = MISSION_POSITION_CONTROLLED;
744             break;
745           case MISSION_STOP:
746             Serial.print("mission stop");
747             state = STOP;
748             break;
749         }
750         break;
751       case INTERFACE_STEP:
752         switch(target){
753           case STEP_VELOCITY:
754             state = STEPRESPONSE;
755             break;
756         }
757         break;
758       default:{
759         //Serial.print("IDLE MODE\n\r");
760         break;
761       }
762     }
763 }
764 ////////////////////////////////////////////////////////////////////////
765 //! Purpose : resetting encoder values
766 //! Params  :
767 ////////////////////////////////////////////////////////////////////////
768 void Reset(){
769   Enco_M1.Cycle_EN_A = 0;
770   Enco_M2.Cycle_EN_A = 0;
771   Enco_M1.Cycle_EN_B = 0;
772   Enco_M2.Cycle_EN_B = 0;
773
774   Enco_M1.PosTicks = 0;
775   Enco_M2.PosTicks = 0;
776
777   Enco_M1.Interval_ENA=0;
778   Enco_M2.Interval_ENA=0;
779
780   Enco_M1.CurrentTime_ENA=0;
781   Enco_M2.CurrentTime_ENA=0;
782
783   Enco_M1.LastTime_ENA=0;
```

```
784   Enco_M2.LastTime_ENA=0;
785
786   Enco_M1.IntervalNum_ENA=0;
787   Enco_M2.IntervalNum_ENA=0;
788 }
789 ////////////////////////////////////////////////////////////////////////
790 //! Purpose : Step response for making and testing controllers
791 //!       Logging and transferring data to interface
792 //! Params  :
793 ////////////////////////////////////////////////////////////////////////
794 void StepReponse(){
795   time = (float)tick_micros*1e-6; // sample time from Interrupt
796   static uint8_t StepCount=0;
797
798   switch(StepState){
799     case RESET_TIMER:{
800       tick_micros=0;
801       time = 0;
802       ZeroAvgValues();
803       SetMotorDirection(FORWARD,FORWARD);
804
805       // If a multistep log is requested from interface
806       if(UserP.Mulistep==ENABLED){
807         StepCount++;
808         SetMotorVelocity(10,10);
809       }
810       // If a controlled log is requested from interface
811       else if(UserP.Control==VELOC_CONTROL)
812         VelocControl();
813       else if(UserP.Control==POS_CONTROL)
814         PositionControl();
815       else if(UserP.Control==MAIN_CONTROL)
816         MainControl();
817       // If Non of the above just do normal step
818       else
819         UpdateMotor(); // Start Motors
820
821       //Take the first sample
822       UpdateSensor();
823       UpdateLogVeloc();
824       StepState = START_STEP;
825       break;
826     }
827     case START_STEP:{
828       UpdateSensor();
829
830       // If a multistep log is requested from interface
831       if(UserP.Mulistep==ENABLED){
832         if(time>=UserP.SampleLength*0.2*StepCount){
833           StepCount++;
834           if(StepCount==5)
```

```
835               StepCount=0;
836             UpdateMultiStep(5);
837             UpdateMotor();
838           }
839         }
840         // If a controller log is requested from interface
841         else if(UserP.Control==VELOC_CONTROL)
842             VelocControl();
843         else if(UserP.Control==POS_CONTROL)
844             PositionControl();
845         else if(UserP.Control==MAIN_CONTROL)
846             MainControl();
847 
848         UpdateLogVeloc();
849         if(time>=UserP.SampleLength){
850           Stop();
851           InterfaceLog();
852           if(UserP.Mulistep==ENABLED){
853             StepCount=0;
854             UpdateMultiStep(0);
855           }
856         }
857         break;
858       }
859     }
860 }
861 ////////////////////////////////////////////////////////////////////////
862 //! Purpose : Used by Step response, to produce multi step responses
863 //!         With different references
864 //! Params  :
865 ////////////////////////////////////////////////////////////////////////
866 void UpdateMultiStep(uint8_t interval){
867   if(interval==0){
868     SetM1.Speed = 10; //
869     SetM2.Speed = 10;
870   }
871   else{
872     SetM1.Speed += interval; // First step will 10 RMP
873     SetM2.Speed += interval; // First step will 10 RMP
874   }
875 
876 }
877 ////////////////////////////////////////////////////////////////////////
878 //! Purpose : reset encoder average varibles
879 //! Params  :
880 ////////////////////////////////////////////////////////////////////////
881 void ZeroAvgValues(){
882   Enco_M1.IntervalSum_ENA = 0;
883   Enco_M1.IntervalNum_ENA = 0;
884   Enco_M2.IntervalSum_ENA = 0;
885   Enco_M2.IntervalNum_ENA = 0;
```

```
886 }
887 ////////////////////////////////////////////////////////////////////////
888 //! Purpose : Logging position and velocity used to stepresponses()
889 //! Params  :
890 ////////////////////////////////////////////////////////////////////////
891 void UpdateLogVeloc (){
892   if(LogNum<=UserP.SampleSize){
893     M1_Log[LogNum].RadPrSek = M1_FBack.RotVelocity_RadprS;
894     M2_Log[LogNum].RadPrSek = M2_FBack.RotVelocity_RadprS;
895     M1_Log[LogNum].Position = M1_FBack.CPosition;
896     M2_Log[LogNum].Position = M2_FBack.CPosition;
897     LogNum++;
898   }
899 }
900 ////////////////////////////////////////////////////////////////////////
901 //! Purpose : not used
902 //! Params  :
903 ////////////////////////////////////////////////////////////////////////
904 void UpdateLogPos(){
905   /*
906   static float PosM1[POSLOG_SIZE];
907   static float PosM2[POSLOG_SIZE];
908   if(PosLogCount<=POSLOG_SIZE){
909     PosM1[PosLogCount] = M1_FBack.CPosition;
910     PosM2[PosLogCount] = M2_FBack.CPosition;
911     PosLogCount++;
912   }
913   if(SetM1.ReachedPos==true||SetM1.ReachedPos==true ||
        PosLogCount<POSLOG_SIZE){
914     char string[100];
915     for(uint32_t i=0; i<PosLogCount;i++){
916       sprintf(string,"%f %f",PosM1[i],PosM2[i]);
917       Serial.print(string);
918       delay(15);
919     }
920     Serial.print("\n"); //Indicating end of log
921     delay(15);
922     PosLogCount=0;
923     MissionState = MIS_RESET;
924     state = IDLE;
925     SetM1.ReachedPos=false;
926     SetM1.ReachedPos=false;
927   }
928   */
929 }
930 ////////////////////////////////////////////////////////////////////////
931 //! Purpose : Tranferring logged data to interface
932 //! Params  :
933 ////////////////////////////////////////////////////////////////////////
934 void InterfaceLog(){
935   char string[400];
```

```
936   float time =0;
937   for(uint32_t i=0; i<LogNum;i++){
938     sprintf(string,"%f %f %f %f
          %f",M1_Log[i].RadPrSek,M2_Log[i].RadPrSek,M1_Log[i].Position,M2_Log[i].Position,tim
939     Serial.print(string);
940     time+=0.001;
941     delay(30);
942   }
943   Serial.print("\n"); //Indicating end of log
944   delay(30);
945   time=0;
946   LogNum=0;
947   ControlState = VELOCITY_CONTROL_S;
948   StepState = RESET_TIMER;
949   state = IDLE;
950 }
951 ////////////////////////////////////////////////////////////////////////
952 //! Purpose : Stop motor
953 //! Params  :
954 ////////////////////////////////////////////////////////////////////////
955 void Stop(){
956   // Directly set speed to zero
957   analogWrite(BLUE_M1, 0);
958   analogWrite(GREEN_M1,0);
959   analogWrite(BLUE_M2, 0);
960   analogWrite(GREEN_M2,0);
961   state = IDLE;
962
963   //Reset Live mission
964   UserP.LiveMission = MISSION_DEFAULT;
965 }
966 ////////////////////////////////////////////////////////////////////////
967 //! Purpose : Test purpose
968 //! Params  :
969 ////////////////////////////////////////////////////////////////////////
970 void Test(){
971
972   //SetMotorEnable(true,true);
973   //SetMotorPosition(28,28);
974   //SetMotorDirection(FORWARD,FORWARD);
975   //SetMotorVelocity(60,0);
976
977   UpdateMotor();
978   state = IDLE;
979 }
980 ////////////////////////////////////////////////////////////////////////
981 //! Purpose : Test for lower motor voltage
982 //! Params  :
983 ////////////////////////////////////////////////////////////////////////
984 void Treshold(){
985   static uint16_t RadprSek=30;
```

```
986     SetMotorVelocity(RadprSek,RadprSek);
987     UpdateMotor();
988     RadprSek-=UserP.DecreaseVal;
989     if(RadprSek==3){
990       UpdateMotor();
991       RadprSek=30;
992       SetMotorVelocity(15,15);
993       UserP.LiveMission = MISSION_DEFAULT;
994       //Stop();
995     }
996 }
997 ////////////////////////////////////////////////////////////////////////
998 //! Purpose : Initiate all varibles
999 //! Params  :
1000 ////////////////////////////////////////////////////////////////////////
1001 void InitVaribles(){
1002   state=0;
1003   SetM1.Direction = FORWARD;
1004   SetM1.Enable = ENABLED;
1005   SetM1.Position = 0;
1006   SetM1.Speed = 15;
1007   SetM1.Volt = 6;
1008
1009   SetM2.Direction = FORWARD;
1010   SetM2.Enable = ENABLED;
1011   SetM2.Position = 0;
1012   SetM2.Speed = 15;
1013   SetM2.Volt = 6;
1014
1015   UserP.SampleTime = 1000; // Micro sek
1016   UserP.SampleLength = 3; // seconds
1017   UserP.SampleSize = LOGSIZE;
1018   UserP.AvgSample1 = 1 ;
1019   UserP.AvgSample2 = 1;
1020   UserP.VelocityUnit = RAD_PR_SEK;
1021   UserP.Mulistep = 0;
1022   UserP.LiveMission = MISSION_DEFAULT;
1023   UserP.DecreaseVal = 1;
1024   UserP.DPosition = 50;
1025   UserP.VelocRef = 15;
1026   UserP.Control = 0;  // main controller
1027   UserP.DirectionCalc =1;
1028   UserP.PosConVoltMax = 3;
1029   UserP.PosConVoltMin = 0;
1030   UserP.Kp_Pos =  1.6;
1031   UserP.CAccDistance = 20; //Deacceleration distance before distination
1032   UserP.CPosDistance = 5;
1033
1034   tick_micros=0;
1035   StepState = RESET_TIMER;
1036   MissionState = MIS_RESET;
```

```
1037    ControlState = VELOCITY_CONTROL_S;
1038    LogNum =0;
1039    time =0;
1040
1041    Enco_M1.CurrentTime_ENA =0;
1042    Enco_M1.IntervalNum_ENA =0;
1043    Enco_M1.IntervalSum_ENA =0;
1044    Enco_M1.Interval_ENA=0;
1045    Enco_M1.LastTime_ENA =0;
1046    Enco_M1.Cycle_EN_A =0;
1047    Enco_M1.Cycle_EN_B=0;
1048    Enco_M1.PosTicks =0;
1049
1050    Enco_M2.CurrentTime_ENA =0;
1051    Enco_M2.IntervalNum_ENA =0;
1052    Enco_M2.IntervalSum_ENA =0;
1053    Enco_M2.Interval_ENA=0;
1054    Enco_M2.LastTime_ENA =0;
1055    Enco_M2.Cycle_EN_A =0;
1056    Enco_M2.Cycle_EN_B=0;
1057    Enco_M2.PosTicks =0;
1058
1059    //M1_FBack.CPosition = 60;
1060    //M2_FBack.CPosition = 60;
1061
1062    Missiontype = MISSION_POSITION;
1063
1064    M1_LPfilter.LastRadprSek = 0;
1065    UserP.UseLowPassfilter = 1;
1066    InitLowPass();
1067    VelocControlInit();
1068 }
1069 ////////////////////////////////////////////////////////////////////////
1070 //! Purpose : Init converstion paramters
1071 //! Params  :
1072 ////////////////////////////////////////////////////////////////////////
1073 void InitConvFactors(){
1074    CFacts.Rad_pr_cycle = 2*PI/PULSES_PR_REV;
1075    CFacts.CM_pr_puls = 2*PI*RADIUS_CM/PULSES_PR_REV;
1076    CFacts.RadPS_conv_micros = 623.3318759;
1077    CFacts.RPM_conv_micros = 5952.380954;
1078 }
1079 ////////////////////////////////////////////////////////////////////////
1080 //! Purpose : All all output connected to L293l are updated here
1081 //! Params  :
1082 ////////////////////////////////////////////////////////////////////////
1083 void UpdateMotor(){
1084
1085    // Setting M1 and M2 direction and speed
1086
1087    uint16_t MotorPWMValue = UpDatePWM(SetM1.Speed);
```

```
1088
1089    //M1
1090    if(SetM1.Direction == FORWARD){
1091      analogWrite(BLUE_M1, 0);
1092      //analogWrite(GREEN_M1,MotorPWMValue);
1093      analogWrite(GREEN_M1,MotorPWMValue);
1094
1095    }
1096    else if(SetM1.Direction == BACKWARD){
1097      analogWrite(GREEN_M1, 0);
1098      //analogWrite(BLUE_M1,MotorPWMValue);
1099      analogWrite(BLUE_M1,MotorPWMValue);
1100    }
1101
1102        MotorPWMValue = UpDatePWM(SetM2.Speed);
1103
1104    //M2
1105    if(SetM2.Direction == FORWARD){
1106      analogWrite(BLUE_M2, 0);
1107      //analogWrite(GREEN_M2,MotorPWMValue);
1108      analogWrite(GREEN_M2,MotorPWMValue);
1109    }
1110    else if(SetM2.Direction == BACKWARD){
1111      analogWrite(GREEN_M2,0);
1112      //analogWrite(BLUE_M2,MotorPWMValue);
1113      analogWrite(BLUE_M2,MotorPWMValue);
1114    }
1115
1116    // Setting Enable input of M1 and M2
1117
1118    //M1
1119    if(SetM1.Enable==ENABLED)
1120      digitalWrite(PUPLE_M1,HIGH);
1121    else if(SetM1.Enable==DISABLED)
1122      digitalWrite(PUPLE_M1,LOW);
1123    //M2
1124    if(SetM2.Enable==ENABLED)
1125      digitalWrite(PUPLE_M2,HIGH);
1126    else if(SetM2.Enable==DISABLED)
1127      digitalWrite(PUPLE_M2,LOW);
1128 }
1129 ////////////////////////////////////////////////////////////////////////
1130 //! Purpose : not used
1131 //! Params  :
1132 ////////////////////////////////////////////////////////////////////////
1133 void DoLiveMission(){
1134    switch(UserP.LiveMission){
1135      case MISSION_TRESHOLD:
1136        Treshold();
1137        break;
1138    }
```

```
1139 }
1140 ////////////////////////////////////////////////////////////////////////
1141 //! Purpose : Run a chosen mission from interface
1142 //! Params  :
1143 ////////////////////////////////////////////////////////////////////////
1144 void DoMission(){
1145   switch(MissionState){
1146     case MIS_RESET:{
1147       switch(Missiontype){
1148         case MISSION_POSITION:
1149           UpdateMotor(); // Start Motors
1150           break;
1151         case MISSION_POSITION_CONTROLLED:
1152           PosLogCount=0;
1153           UserP.Control=2;
1154           PositionControl();
1155           break;
1156       }
1157       UpdateSensor();
1158       MissionState = MIS_RUN;
1159       break;
1160     }
1161     case MIS_RUN:{
1162       switch(Missiontype){
1163         case MISSION_POSITION:
1164           if(M1_FBack.CPosition >=UserP.DPosition ||
1165              M2_FBack.CPosition >=UserP.DPosition){
1166             Stop();
1167             MissionState = MIS_RESET;
1168             state = IDLE;
1169           }
1170           break;
1171         case MISSION_POSITION_CONTROLLED:
1172           PositionControl();
1173           //UpdateLogPos();
1174
1175           break;
1176       }
1177       UpdateSensor();
1178       break;
1179     }
1180   }
1181 }
1182 ////////////////////////////////////////////////////////////////////////
1183 //! Purpose : Choose direction of motors
1184 //! Params  : DIR_M1,DIR_M2 : FORWARD or BACKWARD
1185 ////////////////////////////////////////////////////////////////////////
1186 void SetMotorDirection(int DIR_M1,int DIR_M2){
1187    if(DIR_M1)
1188      SetM1.Direction = FORWARD;
1189    else
```

```
1190        SetM1.Direction = BACKWARD;
1191
1192      if (DIR_M2)
1193       SetM2.Direction = FORWARD;
1194      else
1195       SetM2.Direction = BACKWARD;
1196 }
1197 ////////////////////////////////////////////////////////////////////////
1198 //! Purpose : Run a chosen mission from interface
1199 //! Params  : EN_M1,EN_M2 : ENABLE or DISABLE
1200 ////////////////////////////////////////////////////////////////////////
1201 void SetMotorEnable(bool EN_M1,bool EN_M2){
1202   if(EN_M1)
1203     SetM1.Enable = ENABLED;
1204   else
1205     SetM1.Enable = DISABLED;
1206   if(EN_M2)
1207     SetM2.Enable = ENABLED;
1208   else
1209     SetM2.Enable = DISABLED;
1210 }
1211 ////////////////////////////////////////////////////////////////////////
1212 //! Purpose : Helper function to set velecoty
1213 //! Params  :
1214 ////////////////////////////////////////////////////////////////////////
1215 void SetMotorVelocity(int V_M1,int V_M2){
1216   SetM1.Speed = V_M1;
1217   SetM2.Speed = V_M2;
1218 }
1219
1220 ////////////////////////////////////////////////////////////////////////
1221 //! Purpose : Helper function to set position
1222 //! Params  :
1223 ////////////////////////////////////////////////////////////////////////
1224 void SetMotorPosition(float P_M1, float P_M2){
1225   //UserP.DPosition = P_M1;
1226   //UserP.DPosition = P_M2;
1227 }
1228
1229 ////////////////////////////////////////////////////////////////////////
1230 //! Purpose : All encoder calculation are updated here
1231 //! Params  :
1232 ////////////////////////////////////////////////////////////////////////
1233 void UpdateSensor(){
1234       //Getting Encoder values
1235   M1_FBack.Periode_micros = Enco_M1.Interval_ENA;
1236   M2_FBack.Periode_micros = Enco_M2.Interval_ENA;
1237
1238   M1_FBack.PeriodeCycle   = Enco_M1.PosTicks;
1239   M2_FBack.PeriodeCycle   = Enco_M2.PosTicks;
1240
```

```
1241
1242    //Test
1243    //Serial.println(Enco_M2.Interval_ENA);
1244    //M2_FBack.RotVelocity_RadprS =
            AngularVelocity(Interval_ENB_M2,RAD_PR_SEK);
1245    //Serial.println((float)M2_FBack.RotVelocity_RadprS,3);
1246
1247    M1_FBack.timeSek = time ;
1248
1249        //Updating Velocity
1250
1251    // if M1 average is requested from interface
1252
1253    if(UserP.AvgSample1){
1254      M1_FBack.PeriodeSum_micros = Enco_M1.IntervalSum_ENA;
1255      M1_FBack.PeriodeCount_micros = Enco_M1.IntervalNum_ENA;
1256      Enco_M1.IntervalSum_ENA = 0;
1257      Enco_M1.IntervalNum_ENA = 0;
1258      M1_FBack.PeriodeAvg_micros =
            (float)(M1_FBack.PeriodeSum_micros/M1_FBack.PeriodeCount_micros);
1259      //M1_FBack.RotVelocity_RPM_    =
            AngularVelocity(M1_FBack.PeriodeAvg_micros,RPM);
1260      M1_FBack.RotVelocity_RadprS =
            AngularVelocity(M1_FBack.PeriodeAvg_micros,RAD_PR_SEK);
1261
1262      /*
1263      float RprS =  AngularVelocity(M1_FBack.Periode_micros,RAD_PR_SEK);
1264      M1_FBack.RotVelocity_RadprS  = M1_LPfilter.alfa*RprS + (1-
            M1_LPfilter.alfa)*M1_LPfilter.LastRadprSek;
1265      M1_LPfilter.LastRadprSek =M1_FBack.RotVelocity_RadprS;
1266      */
1267    }
1268    else if(UserP.UseLowPassfilter){
1269      float RprS =  AngularVelocity(M1_FBack.Periode_micros,RAD_PR_SEK);
1270      M1_FBack.RotVelocity_RadprS  = M1_LPfilter.alfa*RprS + (1-
            M1_LPfilter.alfa)*M1_LPfilter.LastRadprSek;
1271      M1_LPfilter.LastRadprSek =M1_FBack.RotVelocity_RadprS;
1272    }
1273    else{
1274      //M1_FBack.RotVelocity_RPM_    =
            AngularVelocity(M1_FBack.Periode_micros,RPM);
1275      M1_FBack.RotVelocity_RadprS =
            AngularVelocity(M1_FBack.Periode_micros,RAD_PR_SEK);
1276      // If they not used zero them
1277      Enco_M1.IntervalSum_ENA = 0;
1278      Enco_M1.IntervalNum_ENA = 0;
1279    }
1280
1281
1282    // if M2 average is requested from interface
1283    if(UserP.AvgSample2){
```

```
1284      M2_FBack.PeriodeSum_micros = Enco_M2.IntervalSum_ENA;
1285      M2_FBack.PeriodeCount_micros = Enco_M2.IntervalNum_ENA;
1286      Enco_M2.IntervalSum_ENA = 0;
1287      Enco_M2.IntervalNum_ENA = 0;
1288      M2_FBack.PeriodeAvg_micros =
              (float)(M2_FBack.PeriodeSum_micros/M2_FBack.PeriodeCount_micros);
1289      //M1_FBack.RotVelocity_RPM_    =
              AngularVelocity(M2_FBack.PeriodeAvg_micros,RPM);
1290      M2_FBack.RotVelocity_RadprS =
              AngularVelocity(M2_FBack.PeriodeAvg_micros,RAD_PR_SEK);
1291      /*
1292      float RprS =  AngularVelocity(M2_FBack.Periode_micros,RAD_PR_SEK);
1293      M2_FBack.RotVelocity_RadprS  = M2_LPfilter.alfa*RprS + (1-
              M2_LPfilter.alfa)*M2_LPfilter.LastRadprSek;
1294      M2_LPfilter.LastRadprSek =M2_FBack.RotVelocity_RadprS;
1295      */
1296
1297    }
1298    else if(UserP.UseLowPassfilter){
1299      float RprS =  AngularVelocity(M2_FBack.Periode_micros,RAD_PR_SEK);
1300      M2_FBack.RotVelocity_RadprS  = M2_LPfilter.alfa*RprS + (1-
              M2_LPfilter.alfa)*M2_LPfilter.LastRadprSek;
1301      M2_LPfilter.LastRadprSek =M2_FBack.RotVelocity_RadprS;
1302    }
1303    else{
1304
1305      //M2_FBack.RotVelocity_RPM_    =
              AngularVelocity(Enco_M2.Interval_ENA,RPM);
1306      M2_FBack.RotVelocity_RadprS =
              AngularVelocity(Enco_M2.Interval_ENA,RAD_PR_SEK);
1307      // If they not used zero them
1308      Enco_M2.IntervalSum_ENA = 0;
1309      Enco_M2.IntervalNum_ENA = 0;
1310    }
1311    //Updating Current Position
1312    M1_FBack.CPosition = CyclesDistance_CM(M1_FBack.PeriodeCycle);
1313    M2_FBack.CPosition = CyclesDistance_CM(M2_FBack.PeriodeCycle);
1314  }
1315
1316  ////////////////////////////////////////////////////////////////////
1317  //! Purpose : Same as UpdateSensor just used for interface
1318  //! Params  :
1319  ////////////////////////////////////////////////////////////////////
1320  void WriteSensorInterface(){
1321    char string[400];
1322    M1_FBack.Periode_micros = Enco_M1.Interval_ENA;
1323    M2_FBack.Periode_micros = Enco_M2.Interval_ENA;
1324    M1_FBack.PeriodeCycle   = Enco_M1.PosTicks;
1325    M2_FBack.PeriodeCycle   = Enco_M2.PosTicks;
1326
1327    //Updating Velocity
```

```
1328    //M1_FBack.RotVelocity_RPM_    =  AngularVelocity(Enco_M1.Interval_ENA,RPM);
1329    M1_FBack.RotVelocity_RadprS =
            AngularVelocity(M1_FBack.Periode_micros,RAD_PR_SEK);
1330    //M2_FBack.RotVelocity_RPM_    =  AngularVelocity(Enco_M2.Interval_ENA,RPM);
1331    M2_FBack.RotVelocity_RadprS =
            AngularVelocity(M2_FBack.Periode_micros,RAD_PR_SEK);
1332
1333    //Updating Current Position
1334    M1_FBack.CPosition = CyclesDistance_CM(M1_FBack.PeriodeCycle);
1335    M2_FBack.CPosition = CyclesDistance_CM(M2_FBack.PeriodeCycle);
1336
1337    sprintf(string,"U%f-%f-%d-%d-%d-%d-%f-%f\n",
1338        M1_FBack.CPosition,
1339        M2_FBack.CPosition,
1340        SetM1.Enable,
1341        SetM2.Enable,
1342        SetM1.Direction,
1343        SetM2.Direction,
1344        M2_FBack.RotVelocity_RadprS,
1345        M2_FBack.RotVelocity_RadprS
1346
1347        );
1348    Serial.println(string);
1349 }
1350
1351 ////////////////////////////////////////////////////////////////////////
1352 //! Purpose : Velocity Controller - canonical PI contoller
1353 //! Params  :
1354 ////////////////////////////////////////////////////////////////////////
1355 void VelocControl(){
1356     //Controlling M1
1357     M1_CP_veloc.e = UserP.VelocRef - M1_FBack.RotVelocity_RadprS;
1358     M1_CP_veloc.u = M1_CP_veloc.b0*M1_CP_veloc.e+M1_CP_veloc.dummy;
1359
1360     if(M1_CP_veloc.u > VOLT_MAX)
1361       M1_CP_veloc.u = VOLT_MAX;
1362     else if(M1_CP_veloc.u <VOLT_MIN)
1363       M1_CP_veloc.u =VOLT_MIN;
1364
1365     M1_CP_veloc.RegOut     = M1_CP_veloc.u;
1366     M1_CP_veloc.dummy    = M1_CP_veloc.b1*M1_CP_veloc.e - M1_CP_veloc.a1*
            M1_CP_veloc.u;
1367     SetM1.ControlOut = M1_CP_veloc.RegOut;
1368
1369     //Controlling M2
1370     M2_CP_veloc.e = UserP.VelocRef - M2_FBack.RotVelocity_RadprS;
1371     M2_CP_veloc.u = M2_CP_veloc.b0*M2_CP_veloc.e+M2_CP_veloc.dummy;
1372
1373     if(M2_CP_veloc.u > VOLT_MAX)
1374       M2_CP_veloc.u = VOLT_MAX;
1375     else if(M2_CP_veloc.u <VOLT_MIN)
```

```
1376        M2_CP_veloc.u =VOLT_MIN;
1377
1378      M2_CP_veloc.RegOut = M2_CP_veloc.u;
1379      M2_CP_veloc.dummy  = M2_CP_veloc.b1*M2_CP_veloc.e - M2_CP_veloc.a1*
              M2_CP_veloc.u;
1380      SetM2.ControlOut = M2_CP_veloc.RegOut;
1381
1382      UpdateMotorControl();
1383  }
1384  ////////////////////////////////////////////////////////////////////////
1385  //! Purpose : Init Velocity Controller
1386  //! Params  :
1387  ////////////////////////////////////////////////////////////////////////
1388  void VelocControlInit(){
1389
1390    M1_CP_veloc.e = 0;
1391    M1_CP_veloc.a1 = -1;
1392    M1_CP_veloc.b0 = 0.4952;    //0.347
1393    M1_CP_veloc.b1 = -0.4855;
1394    M1_CP_veloc.dummy = 0;
1395    M1_CP_veloc.u =0;
1396
1397    M2_CP_veloc.e = 0;
1398    M2_CP_veloc.a1 = -1;
1399    M2_CP_veloc.b0 = 0.4952;    //0.347
1400    M2_CP_veloc.b1 = -0.4855;
1401    M2_CP_veloc.dummy = 0;
1402    M2_CP_veloc.u =0;
1403  }
1404  ////////////////////////////////////////////////////////////////////////
1405  //! Purpose : not used
1406  //! Params  :
1407  ////////////////////////////////////////////////////////////////////////
1408  void UpdateMotorControl(){
1409      uint16_t MotorPWMValue=0;
1410
1411      /*
1412      if(UserP.Control==VELOC_CONTROL)
1413        MotorPWMValue =SetM1.SpeedReg*BIT_RESOLUTON/VOLT_MAX;
1414      else if(UserP.Control==POS_CONTROL)
1415        MotorPWMValue =SetM1.PosReg*BIT_RESOLUTON/VOLT_MAX;
1416      */
1417
1418      MotorPWMValue = SetM1.ControlOut*BIT_RESOLUTON/VOLT_MAX;
1419
1420      // Setting M1 and M2 direction and speed
1421      //M1
1422      if(SetM1.Direction == FORWARD){
1423        analogWrite(BLUE_M1, 0);
1424        //analogWrite(GREEN_M1,MotorPWMValue);
1425        analogWrite(GREEN_M1,MotorPWMValue);
```

```
1426
1427        }
1428        else if(SetM1.Direction == BACKWARD){
1429          analogWrite(GREEN_M1, 0);
1430          //analogWrite(BLUE_M1,MotorPWMValue);
1431          analogWrite(BLUE_M1,MotorPWMValue);
1432        }
1433
1434
1435        /*
1436        if(UserP.Control==VELOC_CONTROL)
1437          MotorPWMValue =SetM2.SpeedReg*BIT_RESOLUTON/VOLT_MAX;
1438        else if(UserP.Control==POS_CONTROL)
1439          MotorPWMValue =SetM2.PosReg*BIT_RESOLUTON/VOLT_MAX;
1440        */
1441
1442        MotorPWMValue = SetM2.ControlOut*BIT_RESOLUTON/VOLT_MAX;
1443
1444        //M2
1445        if(SetM2.Direction == FORWARD){
1446          analogWrite(BLUE_M2, 0);
1447          //analogWrite(GREEN_M2,MotorPWMValue);
1448          analogWrite(GREEN_M2,MotorPWMValue);
1449        }
1450        else if(SetM2.Direction == BACKWARD){
1451          analogWrite(GREEN_M2,0);
1452          //analogWrite(BLUE_M2,MotorPWMValue);
1453          analogWrite(BLUE_M2,MotorPWMValue);
1454        }
1455
1456        // Setting Enable input of M1 and M2
1457
1458        //M1
1459        if(SetM1.Enable==ENABLED)
1460          digitalWrite(PUPLE_M1,HIGH);
1461        else if(SetM1.Enable==DISABLED)
1462          digitalWrite(PUPLE_M1,LOW);
1463        //M2
1464        if(SetM2.Enable==ENABLED)
1465          digitalWrite(PUPLE_M2,HIGH);
1466        else if(SetM2.Enable==DISABLED)
1467          digitalWrite(PUPLE_M2,LOW);
1468 }
1469 ////////////////////////////////////////////////////////////////////////
1470 //! Purpose : Postion Controller - P controller
1471 //! Params  :
1472 ////////////////////////////////////////////////////////////////////////
1473 void PositionControl(){
1474   //Controlling M1
1475   M1_CP_Pos.e = UserP.DPosition - M1_FBack.CPosition;
1476
```

```
1477    //-------------------------P-------------------------------
1478    M1_CP_Pos.u = UserP.Kp_Pos*M1_CP_Pos.e;
1479    //-------------------------Pi------------------------------
1480    //M1_CP_Pos.u = M1_CP_Pos.b0*M1_CP_Pos.e+M1_CP_Pos.dummy;
1481
1482
1483    if(M1_CP_Pos.u<0){
1484      SetM1.Direction = BACKWARD;
1485      M1_CP_Pos.u = abs(M1_CP_Pos.u);
1486    }
1487    else
1488      SetM1.Direction = FORWARD;
1489
1490    /*
1491    if(M1_CP_Pos.u > UserP.PosConVoltMax)
1492      M1_CP_Pos.u = UserP.PosConVoltMax;
1493    */
1494
1495    //--------------------P-------------------------------
1496    SetM1.ControlOut = M1_CP_Pos.u;
1497    //--------------------Pi------------------------------
1498    //SetM1.ControlOut     = M1_CP_Pos.u;
1499    //M1_CP_Pos.dummy    = M1_CP_Pos.b1*M1_CP_Pos.e - M1_CP_Pos.a1* M1_CP_Pos.u;
1500
1501
1502    //Controlling M2
1503    M2_CP_Pos.e = UserP.DPosition - M2_FBack.CPosition;
1504
1505    //-----------------------P-----------------------------
1506    M2_CP_Pos.u = UserP.Kp_Pos*M2_CP_Pos.e;
1507
1508    //-----------------------Pi----------------------------
1509    //M2_CP_Pos.u = M2_CP_Pos.b0*M2_CP_Pos.e+M2_CP_Pos.dummy;
1510
1511    if(M2_CP_Pos.u<0){
1512      SetM2.Direction = BACKWARD;
1513      M2_CP_Pos.u = abs(M2_CP_Pos.u);
1514    }
1515    else
1516      SetM2.Direction = FORWARD;
1517
1518    /*
1519    if(M2_CP_Pos.u > UserP.PosConVoltMax)
1520      M2_CP_Pos.u = UserP.PosConVoltMax;
1521    */
1522
1523    //-----------------P-----------------
1524    SetM2.ControlOut = M2_CP_Pos.u;
1525
1526    //-----------------Pi----------------
1527    //SetM1.ControlOut     = M1_CP_Pos.u;
```

```
1528   //M2_CP_Pos.dummy    = M2_CP_Pos.b1*M2_CP_Pos.e - M2_CP_Pos.a1* M2_CP_Pos.u;
1529
1530   UpdateMotorControl();
1531 }
1532 ////////////////////////////////////////////////////////////////////////
1533 //! Purpose : Init Postion Controller
1534 //! Params  :
1535 ////////////////////////////////////////////////////////////////////////
1536 void PositonControlInit(){
1537   M1_CP_Pos.e = 0;
1538   M1_CP_Pos.a1 = -1;
1539   M1_CP_Pos.b0 = 0.209;    //0.347
1540   M1_CP_Pos.b1 = -0.2089;
1541   M1_CP_Pos.dummy = 0;
1542   M1_CP_Pos.u =0;
1543
1544   M2_CP_Pos.e = 0;
1545   M2_CP_Pos.a1 = -1;
1546   M2_CP_Pos.b0 = 0.209;    //0.347
1547   M2_CP_Pos.b1 = -0.2089;
1548   M2_CP_Pos.dummy = 0;
1549   M2_CP_Pos.u =0;
1550 }
1551 ////////////////////////////////////////////////////////////////////////
1552 //! Purpose : Acceleration Controller -
1553 //! Params  :
1554 ////////////////////////////////////////////////////////////////////////
1555 void AccelerationControl(){
1556   float posdiff =  ( M1_FBack.CPosition- (UserP.DPosition-UserP.CAccDistance)
       )/RADIUS_CM;
1557
1558   SetM1.ControlOut  = sqrt((UserP.VelocRef*UserP.VelocRef) -2*alfa*posdiff);
1559
1560   posdiff = (M2_FBack.CPosition- (UserP.DPosition-UserP.CAccDistance)
       )/RADIUS_CM;
1561
1562   SetM2.ControlOut  = sqrt((UserP.VelocRef*UserP.VelocRef) -2*alfa*posdiff);
1563
1564   SetM1.ControlOut = SetM1.ControlOut*Ke;
1565   SetM2.ControlOut = SetM2.ControlOut*Ke;
1566
1567   UpdateMotorControl();
1568 }
1569
1570 ////////////////////////////////////////////////////////////////////////
1571 //! Purpose : combination of the three controllers
1572 //! Params  :
1573 ////////////////////////////////////////////////////////////////////////
1574
1575 void MainControl(){
1576   switch(ControlState){
```

```
1577      case VELOCITY_CONTROL_S:
1578        VelocControl();
1579        if(M1_FBack.CPosition>=(UserP.DPosition-UserP.CAccDistance) ){
1580          ControlState= ACCELERATION_CONTROL_S;
1581          alfa =
                (UserP.VelocRef*UserP.VelocRef)/(2*UserP.CAccDistance/RADIUS_CM);
                // omega^2/(2*distance)
1582        }
1583        break;
1584      case ACCELERATION_CONTROL_S:
1585        AccelerationControl();
1586        if(M1_FBack.CPosition>=(UserP.DPosition-UserP.CPosDistance) ){
1587          ControlState= POSITION_CONTROL_S;
1588        }
1589        break;
1590      case POSITION_CONTROL_S:
1591        PositionControl();
1592        break;
1593    }
1594  }
1595  ////////////////////////////////////////////////////////////////////////
1596  //! Purpose : Lowpass filter velocity
1597  //! Params  : input - RadprSek
1598  ////////////////////////////////////////////////////////////////////////
1599  float LowPass(float RadprSek){
1600    M1_LPfilter.LastRadprSek= M1_LPfilter.alfa*RadprSek+(1-
          M1_LPfilter.alfa)*M1_LPfilter.LastRadprSek;
1601    return M1_LPfilter.LastRadprSek;
1602  }
1603  void InitLowPass(){
1604    M1_LPfilter.CutOff = 1/(2*PI*27);
1605    M1_LPfilter.Sampletime = 0.001;
1606    M1_LPfilter.alfa =
          M1_LPfilter.Sampletime/(M1_LPfilter.Sampletime+M1_LPfilter.CutOff);
1607
1608    M2_LPfilter.CutOff = 1/(2*PI*27);
1609    M2_LPfilter.Sampletime = 0.001;
1610    M2_LPfilter.alfa =
          M2_LPfilter.Sampletime/(M2_LPfilter.Sampletime+M2_LPfilter.CutOff);
1611  }
1612  ////////////////////////////////////////////////////////////////////////
1613  //! Purpose : Calculate Velocity of microseconds in chosen unit
1614  //! Params  : CycleTimeMikros - encoder periode, unit = se enum
1615  ////////////////////////////////////////////////////////////////////////
1616  float     AngularVelocity(int CycleTimeMikros,int unit){
1617    if(unit==RPM){
1618      if(CycleTimeMikros<=0)
1619        return 0;
1620      return (float)(CFacts.RPM_conv_micros/CycleTimeMikros);
1621    }
1622    else if(unit==RAD_PR_SEK){
```

```
1623      if(CycleTimeMikros<=0)
1624        return 0;
1625      return (float)(CFacts.RadPS_conv_micros/CycleTimeMikros);
1626    }
1627    return -1;
1628  }
1629  ////////////////////////////////////////////////////////////////////////
1630  //! Purpose : Calculate PWM value 0-256 or 0-4096 from voltage
1631  //! Params  : Voltage
1632  ////////////////////////////////////////////////////////////////////////
1633  uint16_t PWMVoltage(float Voltage){
1634      return (uint16_t)(BIT_RESOLUTON/VOLT_MAX*Voltage);
1635  }
1636  ////////////////////////////////////////////////////////////////////////
1637  //! Purpose : Calculate Voltage from PWM value 0-256 or 0-4096
1638  //! Params  : PWM value
1639  ////////////////////////////////////////////////////////////////////////
1640  float     VoltagePWM(int DutyCycle){
1641    return (float)(VOLT_MAX/BIT_RESOLUTON*DutyCycle);
1642  }
1643  ////////////////////////////////////////////////////////////////////////
1644  //! Purpose : Calculate Voltage from PWM value 0-256 or 0-4096
1645  //! Params  : PWM value
1646  ////////////////////////////////////////////////////////////////////////
1647  float     CyclesDistance_CM(int Cycle){
1648    return (float)(Cycle*CFacts.CM_pr_puls);
1649  }
1650  ////////////////////////////////////////////////////////////////////////
1651  //! Purpose : not used
1652  //! Params  :
1653  ////////////////////////////////////////////////////////////////////////
1654  uint64_t DistanceCycles_CM(float distance,int unit){
1655    if(unit==METERS){
1656      distance = distance*1000;
1657    }
1658    return (uint64_t)(distance/CFacts.CM_pr_puls);
1659  }
1660  ////////////////////////////////////////////////////////////////////////
1661  //! Purpose : not used
1662  //! Params  :
1663  ////////////////////////////////////////////////////////////////////////
1664  uint16_t UpDatePWM(uint16_t Rad_pr_Sek){
1665    if(UserP.VelocityUnit == RAD_PR_SEK){
1666
1667      return (uint16_t)(Ke*Rad_pr_Sek*BIT_RESOLUTON/VOLT_MAX);
1668    }
1669    return 100;
1670  }
1671  ////////////////////////////////////////////////////////////////////////
1672  //! Purpose : IRS Interface update timer for live update called
1673  //!           every 2 seconds
```

```
1674 //! Params   :
1675 ////////////////////////////////////////////////////////////////////////
1676
1677 void ISR_USER_INTERFACE(void){
1678   GetUpdateInterface = true;
1679 }
1680 ////////////////////////////////////////////////////////////////////////
1681 //! Purpose : Sampling timer called trigger stepresponse every 1 ms
1682 //! Params   :
1683 ////////////////////////////////////////////////////////////////////////
1684 void ISR_SAMPLE(void){
1685   tick_micros+=UserP.SampleTime;
1686   GetSample=true;
1687 }
1688 ////////////////////////////////////////////////////////////////////////
1689 //! Purpose : Encoder tick Interrupts
1690 //! Params   :
1691 ////////////////////////////////////////////////////////////////////////
1692 void ISR_CH_A_M2() {
1693   Enco_M2.CurrentTime_ENA = micros();
1694   Enco_M2.Interval_ENA = Enco_M2.CurrentTime_ENA - Enco_M2.LastTime_ENA;
1695   if(UserP.AvgSample2){  // if Avg sample is requested from interface
1696     Enco_M2.IntervalSum_ENA+=Enco_M2.Interval_ENA;
1697       Enco_M2.IntervalNum_ENA++;
1698   }
1699   uint8_t  A = digitalRead(CH_A_M2);
1700   uint8_t  B = digitalRead(CH_B_M2);
1701   if((A&&B)||(!A&&!B))
1702     Enco_M2.PosTicks++;
1703   else
1704     Enco_M2.PosTicks--;
1705
1706   Enco_M2.LastTime_ENA =  Enco_M2.CurrentTime_ENA;
1707 }
1708 void ISR_CH_B_M2() {
1709   uint8_t B = digitalRead(CH_B_M2);
1710   uint8_t A = digitalRead(CH_A_M2);
1711   if((B&&A)||(!B&&!A))
1712     Enco_M2.PosTicks--;
1713   else
1714     Enco_M2.PosTicks++;
1715 }
1716 void ISR_CH_A_M1() {
1717
1718   Enco_M1.CurrentTime_ENA = micros();
1719   Enco_M1.Interval_ENA = Enco_M1.CurrentTime_ENA - Enco_M1.LastTime_ENA;
1720   if(UserP.AvgSample1){  // if Avg sample is requested from interface
1721     Enco_M1.IntervalSum_ENA+=Enco_M1.Interval_ENA;
1722       Enco_M1.IntervalNum_ENA++;
1723   }
1724
```

```
1725    uint8_t   A = digitalRead(CH_A_M1);
1726    uint8_t   B = digitalRead(CH_B_M1);
1727    if((A&&B)||(!A&&!B))
1728      Enco_M1.PosTicks--;
1729    else
1730      Enco_M1.PosTicks++;
1731
1732    Enco_M1.LastTime_ENA =   Enco_M1.CurrentTime_ENA;
1733  }
1734  void ISR_CH_B_M1() {
1735    uint8_t B = digitalRead(CH_B_M1);
1736    uint8_t A = digitalRead(CH_A_M1);
1737    if((B&&A)||(!B&&!A))
1738      Enco_M1.PosTicks++;
1739    else
1740      Enco_M1.PosTicks--;
1741  }
```

main.cpp