Projekt 4 - rapport

Team 2

27. maj 2019

Deltagere:

Stud. nr: 201602094	Navn: Søren Holm Korsgaard
Stud.nr.: 201607563	Navn: Jacob Gustafsson
Stud.nr.: 20084327	Navn: Simon Rasmussen
Stud.nr.: 201704483	Navn: Thomas Dueholm Jensen

Indhold

1 Introduktion (Jacob)

Denne projektrapport sammenfatter vores arbejde, igennem PRO3 og PRO4, med at bygge en Hybrid Drone Power Pack - herefter benævnt HDPP.

Projektet har været at bygge et system, som kan kobles på en drone for at forlænge flyvetiden. Problemstillingen består i, at de fleste moderne droner, har en meget kort flyveradius, da batterierne er meget tunge i forhold til energikapacitet. Den begrænsede flyvetid gør dronerne uegnede til opgaver som pakkelevering, flyvning med blodprøver og andet leveringsarbejde, som man ellers har spået, kunne automatiseres med dronerne.

Da energitætheden i benzin er væsentlig højere, end i et lithiumbatteri, bygges en HDPP der baseres med en brændselsmotor med tilhørende generator. På denne måde har vi mulighed for at lade batterierne mens vi flyver.

Nedenfor følger de krav, som er sat op af kunden, henholdsvis de krav som vi har specificeret for det færdige produkt.

1.1 Kundens (undervisers) krav

1.1.1 Funktionelle

- Udgangsspændingen fra HDPP skal være kompatibel med spændingen på standardbatteripakken
- HDPP skal startes enten elektrisk eller mekanisk med håndkraft
- Kapaciteten på HDPP'en skal være tilstrækkelig til at lande dronen sikkert, hvis forbrændingsmotoren svigter.
- HPP'en skal besidde en basal logning, så man kan udlæse performance.

1.1.2 Designspecifikke

- Komponenter skal udvælges så de på bedste vis er et kompromis mellem lav vægt, pris og performance.
- Brug så vidt muligt tilgængelige mekaniske dele - herunder forbrændingsmotor.
- En microcontroller skal sikre kontrol over systemet.

1.2 Preprojekt

For at synliggøre produktets formåen og berettigelse, har vi valgt at gengive vores storytelling og Rich Picture fra vore preprojekt. Begge har de til formål at give et meget generelt og råt overblik over systemet, på et niveau som alle kan forstå.

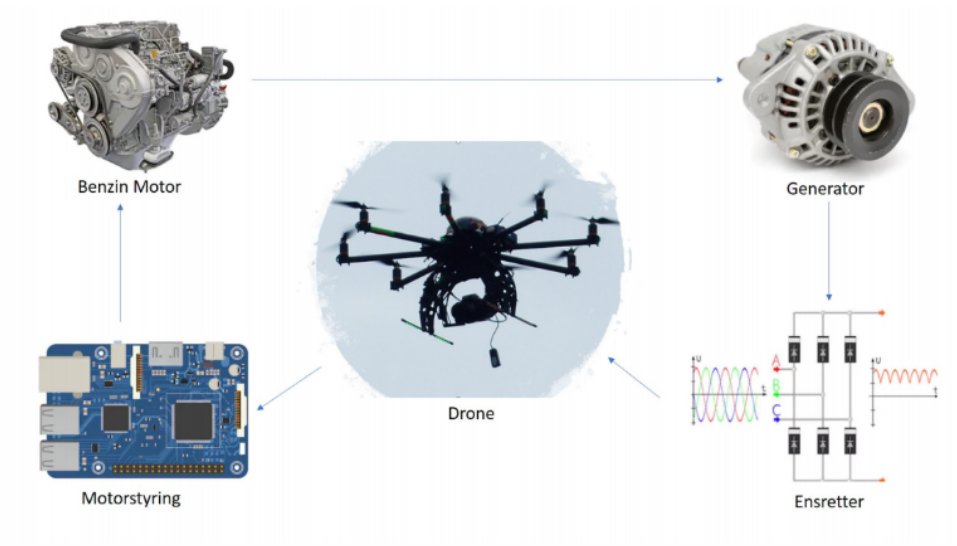
1.2.1 Storytelling

Endelig er det lørdag! Du vågner alt for tidligt i bare spænding, for endelig er det lørdag, og du skal ud og flyve med drone. Dronecertifikatet er endelig i hus, og din store DJI S1000 drone er klar. Med batteriet på 100%, smider du det hele i bilen og kører mod stranden. Der skal tages billeder af morgensolen fra helt nye vinkler. Med det påmonterede kamera, er det ingen sag.

Du ankommer til stranden, kaffen er drukket og du kan mærke hjerterytmen stige - nu skal det være. Du får, let og elegant, dronen i luften og taget nogle gode billeder. Pludselig, og uden varsel, vender dronen tilbage til dig. Du undrer dig meget, og først idet den lander foran dig, kommer du i tanke om den, mildest talt, elendige flyvetid dronen har på batteripakken. Kun omkring 15 minutter! Som du står der og ærgre dig, går de sidste skyer fra solen, og du kunne have fået de perfekte billeder du drømte om. Ovenstående scenarie har vi sat os for at undgå. Dette gøres ved at etablere en forbrændingsmotor og koble til dronens batterisystem gennem en generator. På denne måde lader dronen mens du flyver, og flyvetiden bliver forlænget markant!

1.2.2 Rich Picture

Herunder ses vores Rich Picture, som en konstrueret lige efter opgaven er blevet stillet.



Figur 1: Rich Picture

1.2.3 Kravspecifikation

På baggrund af ovenstående begyndte vi at opstille specifikke, tekniske krav til produktet - blandt andet på baggrund af de af underviserne stillede krav,. I en kravsanalyse specificeres de krav, der er til det færdige produkt eller produktets enkelte komponenter.En udbytterig kravsanalyse indebærer at kunden inddrages.Kravene er nedenfor opstillet efter EARS-princippet (EARS:Easy to Approach Requirements Syntax).

Kravene er inddelt udfra ufravigelige (ubiquitous), begivenhedsorienterede (event-driven), driftsorienterede(state-driven) og fejlorienterede krav (unwanted behaviour). EARS-princippet muliggør således en prioritering af kravene.

1.2.3.1 Ubiquitous

1. Motoren skal kunne startes vha. BLDC-generator.
2. Udstødning skal monteres sådan at varmen ikke påvirker dronen.
3. Generatoren skal levere en middeleffekt på 2450 W ved 22,2 V (110,3 A)
4. Generatoren skal være 3-faset jf. projektbeskrivelsen.
5. HPP må maksimalt veje 5 kg.
6. Ensretteren skal kunne klare at håndtere en effekt af 2,45 kW.
7. Ensretteren skal modtage 3-faset vekselstrøm og levere en jævnstrøm.
8. HPP skal inddæmmes, så den kan modstå vejrforhold, jf. IP56 standard.
9. Ensretteren må maksimalt have ripple på 1V output.

1.2.3.2 Event Driven

1. I tilfælde af nødlanding skal motoren deaktivere.

1.2.3.3 State Driven

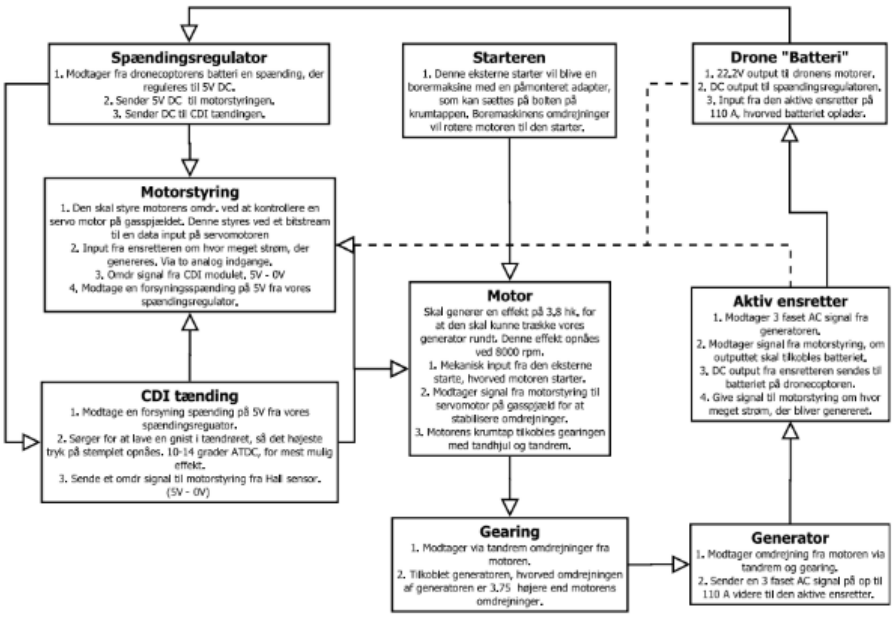
1. HPP skal fungere korrekt før dronecopterens motorer igangsættes ved take-off.
2. Når motoren er aktiv, skal omdrejninger reguleres efter belastning af ladestrømmen.
3. Når generatoren ikke er i fulde omdrejninger, skal dronen ikke kunne lette.
4. Der skal defineres et effekt setpunkt til kontrol af opstart.
5. Når generatoren genererer strøm, skal input og output til/fra ensretteren overvåges og logges.

1.2.3.4 Unwanted

1. Dronecopterens motorer vil tillade nødlanding ved fejl i moteren.
2. Hvis der opstår fejl i motoren, vil dronen lande når spændingen falder på batteriet.
3. Hvis generatoren ikke opnår fulde omdrejninger inden 10 sek. efter motorstart, afbrydes motoren.
4. Hvis udgangsstrømmen ikke når setpunktet (PID-regulering), skal dronen nødlande.

1.2.4 Blokdiagram

På baggrund af alt det ovenstående, samt flere andre overvejelser, som der kan læses mere om i den vedlagte preprojekt rapport, endte vi op med følgende blokdiagram over de forskellige subsystemer.



Figur 2: Blokdiagram

2 Baggrund

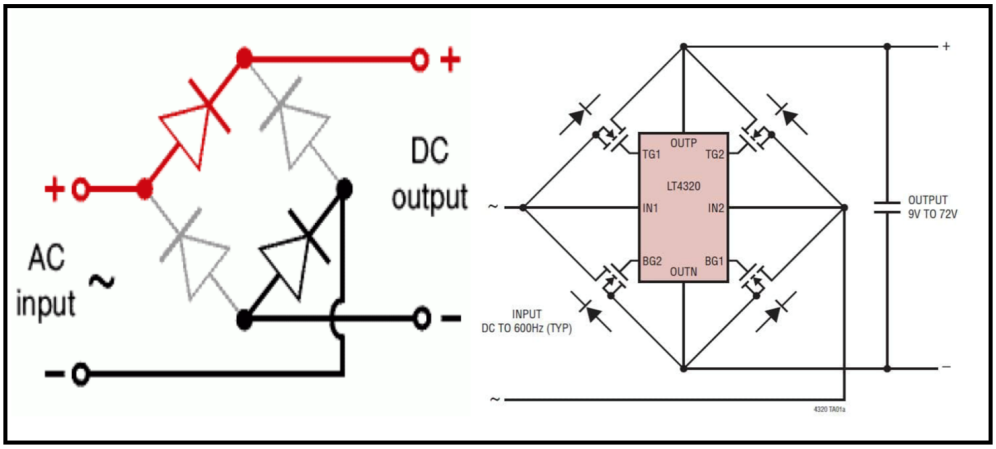
2.1 Aktive ensretter (Søren og Thomas)

Ét af HPP's primære subsystemer er blokken, der skal stå for ensretningen af den tre-fasede strøm/spænding, der genereres fra forbrændingsmotoren og generatoren. Der har i processen været mange essentielle dilemmaer og prioriteter, der skulle afklares for at få den mest optimale løsning. Da der foreligger et grundlæggende ønske om at holde den samlede vægt på HPP nede på et acceptabelt niveau, var det nærliggende at forsøge at realisere ensretningen aktivt. Hvad dette vil sige, berøres i det efterfølgende afsnit.

2.1.1 Passiv vs. aktiv ensretning

I en typisk ensretter (en passiv ensretter) anvendes en diodekonstruktion, som tillader et AC-signals positive cyklus at passere til outputtet, for derefter at invertere den negative del af cyklussen. Resultatet er et udelukkende positivt signal, med en variabel spændingsamplitude. Da vi forventer at lede ca. 110 A gennem systemet, er dioder ikke anvendelige. Spændingsfaldet over dioder ligger typisk i omegnen af 0.3-0.7V, og det vil betyde et effektab, som motoren skal kompensere for via flere omdrejninger.

I stedet anvendes en "Ideal Diode Controller", som har til formål at levere spænding til gate-terminalen på MOSFET transistorer. En MOSFET har i cut-off tilstand en diodelignende opførsel, men når de leder optimalt, er modstanden i transistoren meget lav (helt ned til få mΩ). Parallelkobles transistorer kan modstand sænkes yderligere. IC'en styrer, hvornår transistorerne åbner og lukker, og kontrollerer dermed AC-signalets passage gennem ensretteren, så output-signalet er positivt, ligesom ved en almindelig diode-ensretter. Denne styring gør, at ensretteren defineres som en aktiv ensretter.

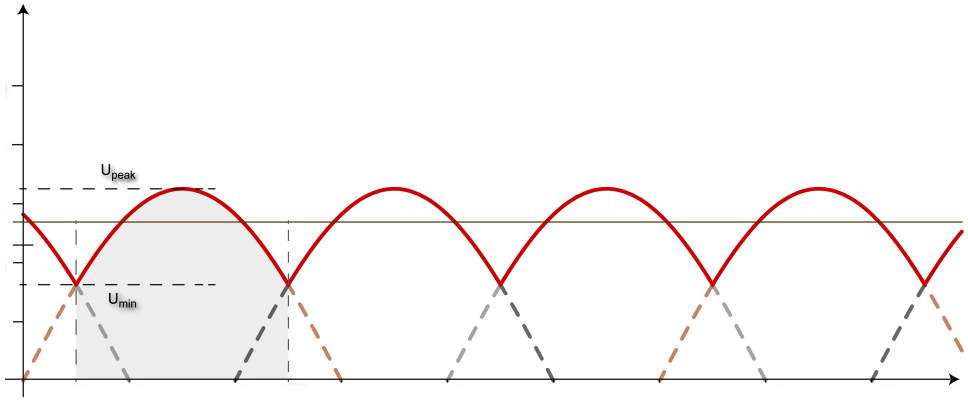


Figur 3: Passiv vs. aktiv ensretning

Den store fordel ved aktiv ensretning kontra passiv er reduceringen af effekttabet gennem kredsløbet. Denne reduktion påvirker direkte, at forbrændingsmotoren skal køre med mindre RPM for at producere behovet til dronen, end hvis ensretningen var passiv. Derudover vil dimensioneringen af heatzinks til kredsløbet ligeledes være væsentlig mindre, da mindre effekttab betyder mindre varmeafledning.

En generel udfordringen ved AC/DC konvertering er *ripples* på output signalet. Ensrettes en enkelt AC-fase, får man et udgangssignal, med stor variation i spænding leveret. Kombineres tre faser med en faseforskydning på 120° mellem sig, som i vores tilfælde, vil summationen af de 3 sinuskurver generere et output signal med mindre *ripple*.

Der vil stadig være *ripple* til stede i signalet, og denne udjævnes til et acceptabelt niveau (for batteriet på drone) vha. kondensatorer. Kondensatoren lader, når $\frac{dv}{dt}$ på udgangssignalet er positivt og aflader, når spændingen igen falder. Den strøm kondensatoren afgiver adderes udgangssignalet, hvorfor $\frac{dv}{dt}$ øges.



Figur 4: Graf over ripple spænding på output signal som funktion af tid. Stiplet linje = ensrettet enkeltfase AC-signal. Rød linje = ripple på kredsløb med kondensator parallelt på load. Kilde: Wikimedia Commons

2.1.2 Valg af IC

I timebox 2 analyserede vi os frem til følgende valg af IC til realiseringen af kredsløbet til den aktive ensretter:

- Linear Technology, LT4320-1, Ideal Diode Bridge Controller

Dette skyldes bl.a., at den er designet specifikt til luftbårne strømforsyningssystemer, som kan håndtere frekvenser op til 600 Hz, 9-72V, og opfylder tidligere defineret krav om optimal effektivitet, og minimalt effekttab.

2.1.3 Kredsløbet

Leverandøren af LT-4320-1, foreslår kredsløbet[2] set på figur, som en yderst energieffektiv tre-faset aktiv ensretter.

Vi valgte at arbejde videre med den opstilling. De næste afsnit indeholder en opsummering af simulering, realiseringen og test af kredsløbet.

2.2 Spændingsregulator (Jacob)

2.3 Motorstyring (Simon)

Motorstyringen for systemet vil bestå i præcis justering af forbrændingsmotorens omdrejninger i forhold til dronecopterens strømforbrug. Det valgtes at arbejde med et simplere system for at fokuserer på principperne i motorstyring. Det betød at motorstyringen skulle bestå i udvikling af PID-regulering af forbrændingsmotoren sådan at der skulle foregå korrektion af motorens spjældvinkel i forhold til motorens omdrejninger.

Fjare[3] er en artikel der beskriver PID regulering af en forbrændingsmotor som skal drive en aerocopter. Der er taget inspiration fra denne tilgang inklusiv PID-regulering og PID-koefficienter.

2.3.1 Kravsspecifikation

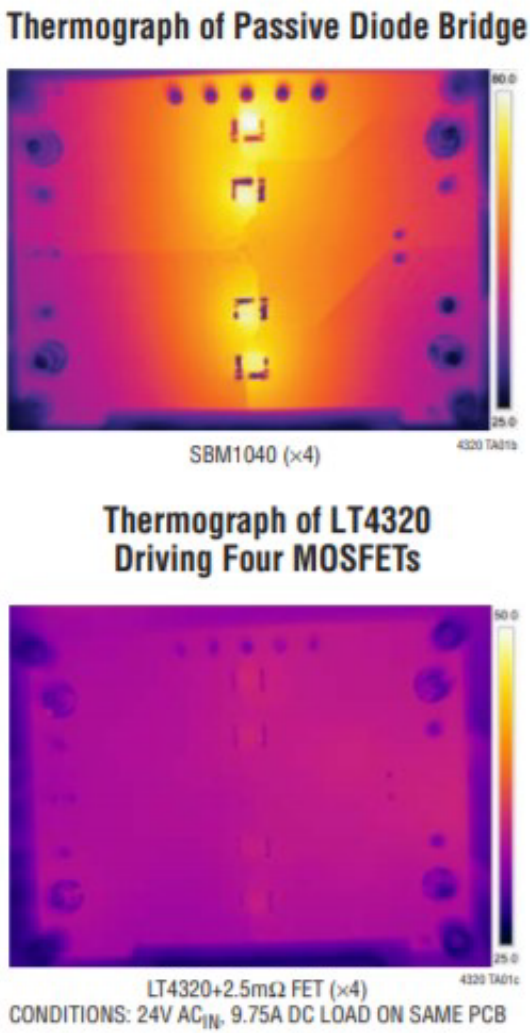
Med baggrund i Fjare[4] sættes følgende krav:

- Overshoot skal ikke være mere en 197 rpm.

[2] urlhttps://www.analog.com/en/products/lt4320.html

[3] Paul D. Fjare. „Feedback Speed Control of a Small Two-Stroke Internal Combustion Engine that Propels an Unmanned Aerial Vehicle“. Speciale. University of Nevada, Las Vegas, jan. 2014.

[4] Ibid.



Temperature Rise

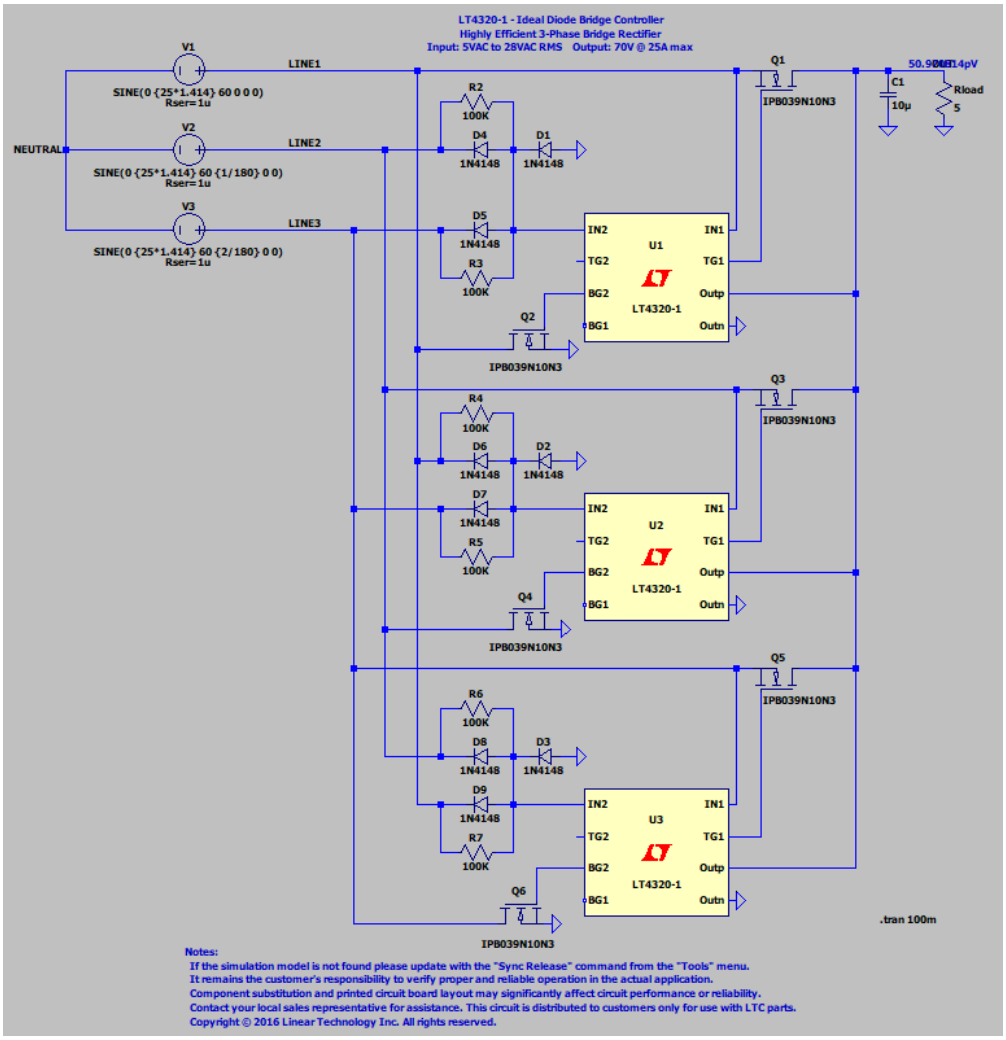
CURRENT	MOSFET 2.5mΩ	DIODE SBM 1040
2A	0.6°C	15°C
4A	3.5°C	32°C
6A	6.7°C	49°C
8A	11°C	66°C
10A	16°C	84°C

DC Input, On Same PCB

Figur 5: Varmeudvikling. Passiv vs. LT4320[1]

[1] Fra databladet til LT4320-1. `https://www.analog.com/media/en/technical-documentation/data-sheets/4320fb.pdf`

- Justeringstiden må max være 8,8 sekunder.
- Ifm. et step respons skal 90 % af målet være opnået i mindre end 3 sekunder.
- Det skal være muligt at vedligeholde en konstant hastighed over længere tid (dvs. mere end 5 minutter).
- Servomotoren skal kunne reguleres.
- Motoren skal kunne startes.
- Omdrejningerne skal kunne måles.



Figur 6: Kredsløb til realiseringen af ensretteren

2.3.2 PID regulering

PID-regulering består i regulering af et output på baggrund af forskellen mellem det tilstræbte og det faktiske - dette kan kaldes fejlen. PID regulering kan beskrives med

$$u(t) = k_P e(t) + k_I \int_0^t e(\tau)\mathrm{d}\tau + k_D \frac{\mathrm{d}e(t)}{\mathrm{d}t} \tag{1}$$

hvor $e(t)$ er fejl som funktion af tiden, $e(\tau)$ er fejl som funktion af den akkumulerede tid og k_P, k_I og k_D er koefficienter svarende til henholdsvis det proportionale, integrede og deriverede forhold. Det proportionale vil sige forholdet mellem tilstræbte og det faktiske for et givent tidspunkt. Det integrede vil sige forskellen mellem tilstræbte og det faktiske akkummuleret over tid. Det betyder at den integrerede koefficient viser en trend for fejlen. Det deriverede er et mål for hvor hurtigt fejlen ændrer sig over tid.

Formålet ved udarbejdelse af PID-regulering er at finde de optimale værdier af k_P, k_i og k_d.

Som udgangspunkt er det tidligere vist([5]) at $K_p = 0,065$ og $K_i = 0,000005$ har været optimale, samt at

[5]Fjare, „Feedback Speed Control of a Small Two-Stroke Internal Combustion Engine that Propels an Unmanned Aerial Vehicle“.

K_d helt blev droppet eftersom det ikke var en gavnlig parameter.

2.3.3 Software

Fra Fjare[6] kunne et eksempel på et program være skrevet i pseudo C-kode:

```
void velPID ( ) {
  lowpassSpeed = alpha * lastLowpassSpeed + (1-alpha) * measuredSpeed;
  K1 = kp * setpointWeight * (setpointSpeed - lastSetpointSpeed) + kp *
     (lastMeasuredSpeed - lowpassSpeed);
  K2 = ki * (setpointSpeed - lowpassSpeed);
  K3 = kd * (2 * lastMeasuredSpeed - lowpassSpeed -
     lastLastMeasuredSpeed);
  output = lastOutput - K1 - K2 - K3;
  throttlePos = floor(output + 0.5);
  if(throttlePos<throttleopen){
    output = (double) throttleopen;
    throttlePos=throttleopen;
  }
  if(throttlePos>throttlesafe){
    output = ( double )throttlesafe;
    throttlePos=throttlesafe;
  }
  lastLowpassSpeed = lowpassSpeed;
  lastLastMeasuredSpeed = lastMeasuredSpeed;
  lastMeasuredSpeed = lowpassSpeed;
  lastSetpointSpeed = setpointSpeed;
  lastOutput = output;
  throttle.writeMicroseconds(throttlePos);
}
```

[6]Fjare, „Feedback Speed Control of a Small Two-Stroke Internal Combustion Engine that Propels an Unmanned Aerial Vehicle“.

3 Metode og analyser

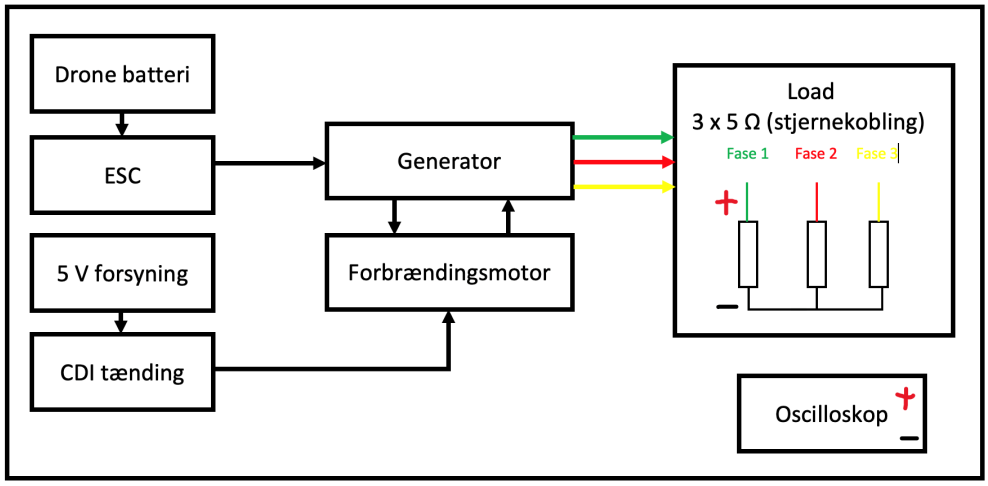
3.1 Aktive ensretter (Søren og Thomas)

3.1.1 Test af generators output spænding

Inden vi anskaffede os IC'erne til kredsløbet, skulle det sikres, at forbrændingsmotor og generator i tomgangsdrift som minimum producerede 9 V (peak). Dette var nødvendigt for, at IC'erne var funktionsdygtige. Der opstilles som følge af ovenstående konstatering et krav til den aktive ensretter.

2.1.3.9 Der skal som minimum være 9 Volt (peak) på udgangen af generatoren, inden denne tilkobles den aktive ensretter

For at verificere dette blev der konstrueret et testsetup, hvor det var muligt at måle output spændingen på én af generatorens tre faser. Et blokdiagram over testsetuppet kan ses i nedenstående figur.



Figur 7: Blokdiagram til test af outout spænding på en fase af generator

Testen viste, at der var en spænding på 11.4 V (peak)[7] på én af generatorens faser med forbrændingsmotoren kørende i tomgang. Ud fra resultatet af denne test var det bekræftet, at der altid ville være et tilstrækkeligt spændingsniveau som input til ensretteren. Derfor indkøbte vi IC'erne med henblik på videre konstruktion af kredsløb.

3.1.2 Simulering

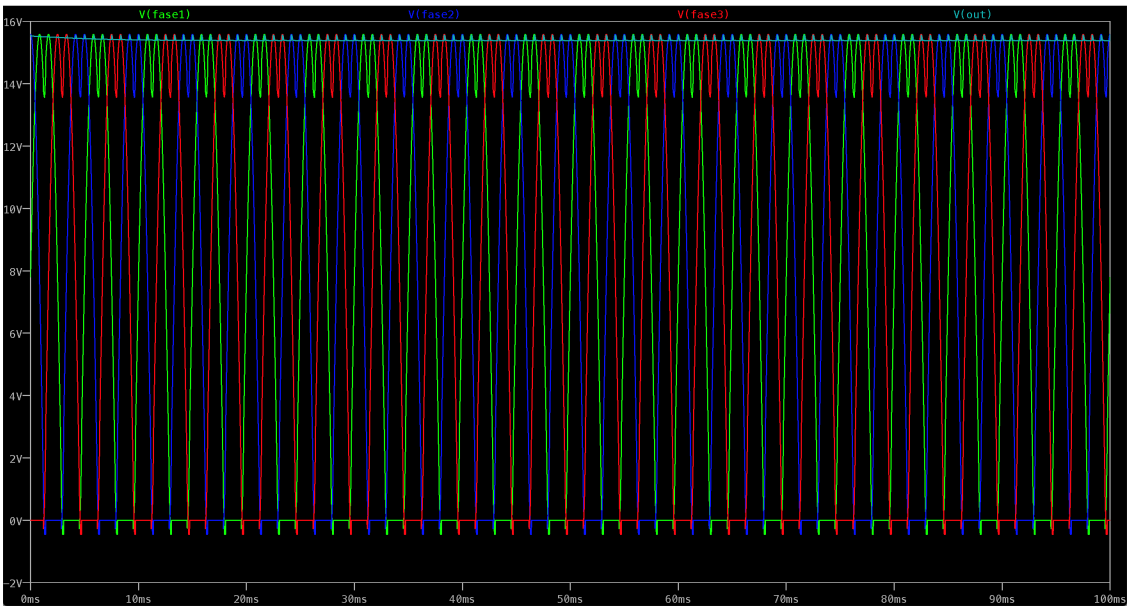
Inden vi påbegyndte opbygningen af den aktive ensretter, blev funktionaliteten simuleret vha. programmet, LT-spice, som er leveret af IC'ens producent, *Linear Technology*.

Figur8 viser et skærmbillede af simuleringen. Det fremgår tydeligt, at outputtet (lyseblå) er ensrettet i forhold til faserne (grøn, rød og blå) til et tilnærmelsesvist DC niveau.

Det skal nævnes, at skaleringen af komponenter i det simulerede kredsløb ikke er identisk med det tiltænkte. Simuleringen er udført for at konstatere kredsløbets teoretiske funktionalitet.

Ud fra resultatet[8] i simuleringen var vi overbevist om, at kredsløbet i teorien kunne leve op til den ønskede funktionalitet. Næste del af processen var derfor at opbygge kredsløbet på et breadboard i en nedskaleret version.

[7] Se bilag, Timebox 6, for yderligere information.
[8] Se bilag, Timebox 7, for yderligere information.



Figur 8: Resultat af simuleringen. V_{out}: Målingen over load modstanden, V_{fase1}: Målingen af fase 1, V_{fase2}: Målingen af fase 2, V_{fase1}: Målingen af fase 1

3.1.3 Realisering på breadboard (nedskaleret)

Kredsløbet blev opbygget på et breadboard i en nedskaleret version for at teste funktionaliteten. Med nedskaleret menes der, at vi benyttede en 1 kΩ's modstand som load, således der ikke blev trukket en stor strøm gennem kredsløbet. Der blev ligeledes påsat en tilpasset kondensator parallelt med load-modstanden for at udjævne eventuelt *ripple* på outputtet.

Figur 9 viser et skærmbillede fra målingen med oscilloskop af den nedskalerede realisering.

Her ses det, at outputtet er jævnt ensrettet i forhold til de tre faser, hvorved vi kunne konstatere, at det nedskalerede kredsløb fungerede efter hensigten.

For yderligere at bevise, at den aktive ensretter var væsentlig mere effektiv end en standard diodebro, foretog vi en måling af spændingen på én af MOSFET transistorernes gate pin i forhold til fasen på transistorens *source* pin.

Figur 10 viser et skærmbillede af resultatet af målingen.

Her ses det, at der er et spændingsfald på 212.5 mV fra gate til source, hvilket er en klar forbedring i forhold til en diodes forward spænding på 700 mV.

Spændingstabet er forårsaget af MOSFET transistorens on resistance, RDS(on), som under denne test var 4Ω. Når kredsløbet skal realiseres uden nedskalering, vil der blive valgt en MOSFET med en langt mindre RDS(on), hvilket teoretisk vil forbedre kredsløbet yderligere.[9]

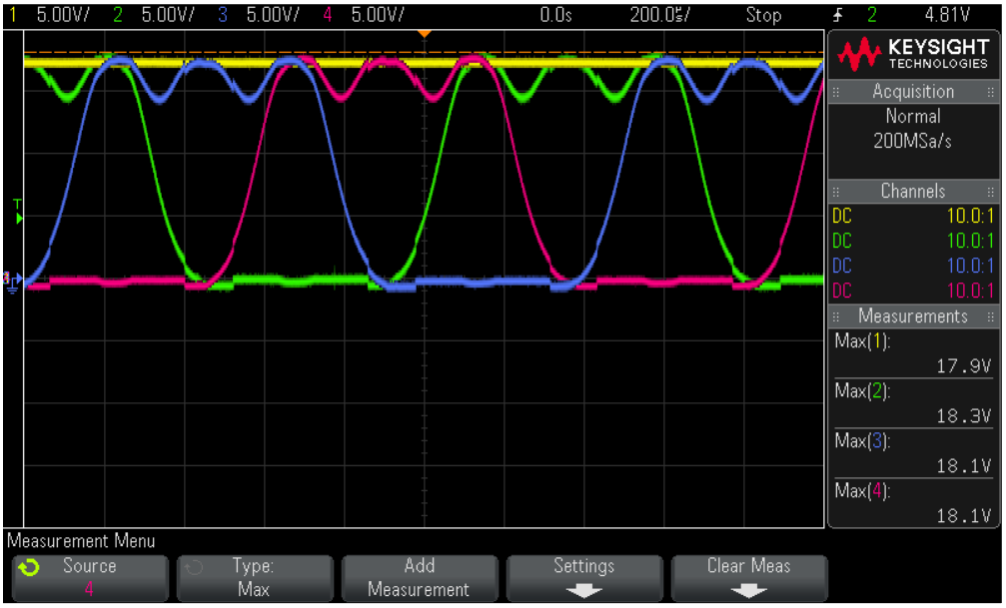
3.2 Spændingsregulator (Jacob)

3.3 Motorstyring

3.3.1 Servo motor og ESC driver (Søren)

For at få vores servo motoren til at fungere, kræves der et signal på 50 Hz og ved at justere PWM mellem 1 ms til 2 ms aktiv høj, hvilket fremgår fra databladet på servo motoren (SG90 micro servo). Kan vi justere positionen på servo motor fra 0 – 90 grader.

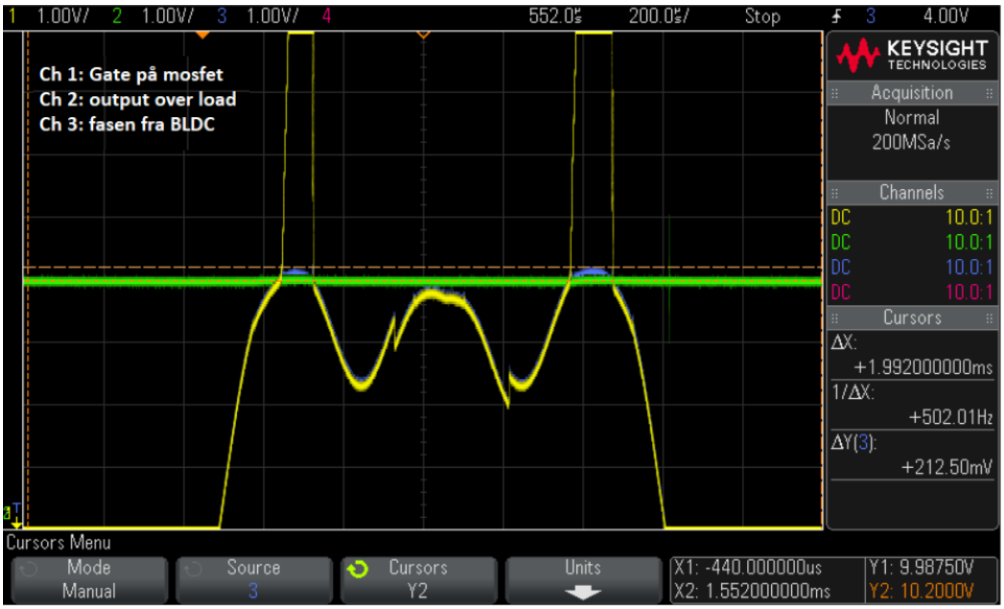
[9]Se bilag, Timebox 7, for yderligere information



Figur 9: Nedskaleret realisering på breadboard. Gul: V_{out}. Grøn, blå og rød: V_{in} (3 faser)

Dette signal bruges også til at justere hastigheden på vores BLDC-motor, via vores ESC. De værdier vi brugte, var 1 ms til stop af BLDC-motor og værdien 1.3 ms, for at kunne starte motoren. Denne værdi 1,3 ms, kom vi frem til via brug af et scope med et wavegen, hvor vi kom frem til at, at ved denne værdi, havde motoren nemt ved at starte. Følgende krav blev opsat, for at vi kunne verificere at det virkede efter hensigten

- 50 Hz signal, 20 ms periode (Krav nr. 2.1.3.8.1)
- Skal kunne justere puls mellem 1 – 2 ms. (Krav nr. 2.1.3.8.2)

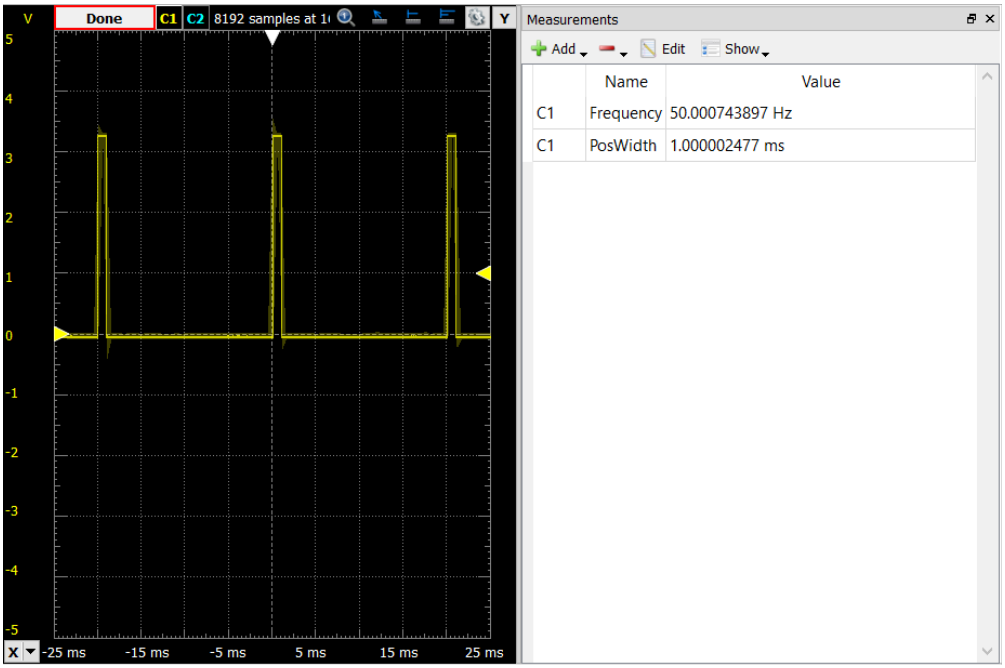


Figur 10: Måling af gate- vs. source spænding

```
void init_pwm() {
        /*
         * Vi ønsker et 50 Hz signal. og vores clock er 48 Mhz.
         * 48Mhz/50Hz = 960000, da vores counter er en 16 bit, dvs max decimal 65536.
         * vores prescaler værdi er = 960000/65536 = 14,648, så vi sætter den til 16.
         * vores count værdi bliver (960000/16)-1 = 59999 for at få en 50Hz frekvens
         */
        SIM->SCGC5 |= SIM_SCGC5_PORTD_MASK;
        SIM->SCGC6 |= SIM_SCGC6_TPM0_MASK;

        /*
         * enable TPM in mux (4UL << 8);
         */

        PORTD->PCR[SERVO_THROTTLE] |= (4UL << 8);
        PORTD->PCR[ESC_SIGNAL] |= (4UL << 8);
        /*Selects the clock source for the TPM counter clock:
         * 01 MCGFLLCLK clock or MCGPLLCLK/2 = (1UL << 24)
         Selects the MCGPLLCLK or MCGFLLCLK clock for various peripheral clocking options:
         1 MCGPLLCLK clock with fixed divide by two = (1UL << 16)
         */

        SIM->SOPT2 |= ((1UL << 24) | (1UL << 16));

        // Set count value in reach for 2^16 = 65536
        TPM0->MOD = PERIOD - 1;

        /*
         * Prescale Factor Selection:
         * 100 Divide by 16 = (1UL << 0)
         */
        TPM0->SC = (4UL << 0);

        /*
         * Debug Mode:
         * 11 LPTPM counter continues in debug mode = (3UL << 6)
         */

        TPM0->CONF |= (3UL << 6);

        /*
         * Channel Mode Select: active HIGH
         * ELSB = (1UL << 5)
         * MSB = (1UL << )
         * check side number 555 in manual for channel options
         */
        TPM0->CONTROLS[PWM_CH_SERVO].CnSC = ((1UL << 3) | (1UL << 5));
        TPM0->CONTROLS[PWM_CH_ESC].CnSC = ((1UL << 3) | (1UL << 5));
        /*
         * set duty cycle to 0
         */
        TPM0->CONTROLS[PWM_CH_SERVO].CnV = MIN_VAL_SERVO;
        TPM0->CONTROLS[PWM_CH_ESC].CnV = MIN_VAL_SERVO;
        /*
         * Clock Mode Selection
         Selects the LPTPM counter clock modes. When disabling the counter, this field remain set until
         acknolwedged in the LPTPM clock domain.

         01 LPTPM counter increments on every LPTPM counter clock =  (1UL << 3)
         */
        TPM0->SC |= (1UL << 3);

}
```

Figur 11: Opsætning af servo og ESC driver

Ved at opsætte analog Discovery til på pin PTD3, vises at vores init funktion virker efter hensigten, med 50 Hz og 1 ms puls. Og hermed lever op til følgende krav 2.1.3.8.1 og 2.1.3.8.2, der er beskrevet på forrige side.



Figur 12: Test af init-funktion

Herefter har vi implementeret en funktion til at styre vores servo motor. Den tager et argument, som er i procent, hvor meget den skal dreje, i forhold til om den skal give fuld gas (100%) eller bare kører i tomgang (0%). Men da vi fik monteret servo motor, fik vi testet at ved 1,4 ms signal til servo, at dette var fuldgas, dette blev derefter tilpasset i koden. Så vores max værdien der kunne sendes ud er 1,4 ms.

3.3.2 Start af motoren

Da den driver vi brugte til at styre servo motoren er den samme, som vi skal bruge for at kunne få vores BLDC-motor i gang, da den fungere som el-starter til motoren. Blev der implementeret en pin ekstra, i driveren, der er vist i koden se figur 16. Herefter lavede vi en start funktion, der skulle bruges til at starte motoren.

Flow for opstart af motor, ved funktionen `start()`:

1. Startknap initialiseres
2. Afventer at brugeren trykker på knappen
3. En timer opsættes, som tæller vores 'i' værdi op, når der er gået 500 ms.
4. Så prøver motoren at starte benzinmotoren, og det vil den gøre så længe start knappen er trykket ned (dog i maks. 15 sek)
5. Efter 15 sek vil opstartsprocessen afbrydes.

```
void angle_throttle(unsigned int procent) {

        if (procent > 0 && procent < 100) {
                int res = (MIN_VAL_SERVO * procent)
                    / 100;
                TPM0->CONTROLS[PWM_CH_SERVO].CnV =
                    MIN_VAL_SERVO + res;
        }
```

Figur 13: Kode til at styre gasspjældet

3.3.3 RPM-detektering

Implementering af koden til at aflæse rpm, der skal bruges til vores PID-regulering. Har vi taget udgangspunkt i at aflæse et tacho signal som kommer fra vores tændspole. Dette signal lave en høj spænding ved hver omdr, og går lav igen.

For at afkode dette signal, bruges der input capture, der er implementeret i TPM periferenheden på kl25. Funktionen input capture, fungere ved at den tælle op til hver 'rising edge' og gemmer denne værdi. Og når vi kender værdien og frekvensen vi tæller op med, kan vi omregne dette til RPM.

Ved formlen: $60\text{sek} \cdot \frac{\text{frekvens}}{\text{input capture v\_rdien}} = \text{RPM}$.

Selve koden er vist her forneden, hvor vi bruger interrupts, til at opfange overflows og når vi får et rising edge, gemmer den værdi i CNV-registeret.

Her initialiseres vores TPM1 modul. Vores frekvens der bruges til at tælle op med, bliver nedskaleret, til 375 kHz. Dvs. imellem hver overflows går der 5,75 Hz, hvilket passer med vores måle område (17 - 167 Hz) vi arbejder i.

3.3.4 Stepinput og overføringsfunktion (Simon)

PID-koefficienterne skal defineres udfra en overføringsfunktion for vores system som skal reguleres. For at udlede overføringsfunktionen blev et forsøg med optagelse af steprespons udarbejdet.

Som udgangspunkt skulle det interessante for dronecopteren være at strømmen fra batteri til dronens propelmotorer er konstant. Tilgengæld skal spjældet til forbrændingsmotoren konstant reguleres. Derfor vil en teststand som udgangspunkt have følgende:

- input: strøm fra batteri til drone
- output: regulering af spjæld via servomotor

Her vil $e(t)$ være et mål for forskellen mellem den ønskede strømstyrke og den faktiske.

Det var ikke muligt at måle strømmen fra batteri til drone. For at lave en teststand som kan tjene som demonstration af PID-regulering samt som skabelon for senere tests fremstilles istedet en teststand som skal justere spjældet i forhold til motorens omdrejninger. Dvs:

```
void start() {
        init_start();
        while (1) {
                if (PTD->PDIR & (1UL << START_BTN)) {
                        // Lave et interrupt for 500 ms
                        initSystickTmr(1500000);
                        //Sætter vores pwm signal med en puls på 1,25ms, dette genere 500 rpm +- 100 rpm
                        TPM0->CONTROLS[PWM_CH_ESC].CnV = START;
                        // Den skal starte så længe knappen er trykket ned og må maks cranke i 15 s
                        while (i != 30 && PTD->PDIR != (0UL << START_BTN))
                                ;
                        __disable_irq();
                        //Sætter vores pwm signal puls til 1 ms, hvilket stopper Starteren
                        TPM0->CONTROLS[PWM_CH_ESC].CnV = MIN_VAL_SERVO;
                        break;

                } else
                        continue;

        }

}
```

Figur 14: Kode til `start()`

- input: motorens omdrejninger
- output: regulering af spjæld via servomotor

I dette tilfælde vil $e(t)$ være et mål for forskellen mellem det ønskede omdrejningstal og det faktiske.

For at få en grov ide om niveauet af koefficienterne kan en simulink simulation være gunstig. I denne skal gøres en række antagelser af forholdet mellem spjældvinkel og omdrejningstallet. Når koefficienterne er tunet i simulink kan der laves en test.

3.3.4.1 Steprespons

Der vil tages udgangspunkt i måling af et steprespons på motorens omdrejninger. Til målinger af omdrejninger anvendtes et oscilloskop som optog data fra Hall-sensoren. Optagelsen blev tricket fra KL25Z.

Data fra oscilloskopet bestod i 50000 målinger som bestod af Hall-sensorens spænding, spændingen fra servomotoren, der styrer spjældet, samt et tidsmål. Nedenfor ses et udsnit af de 50000 målinger:

second	Volt	Volt1
-1	0.215206000000000	4.98492460000000
-0.999900000000000	0.215206000000000	4.90452260000000
-0.999800000000000	0.134804000000000	4.90452260000000
-0.999700000000000	0.215206000000000	4.98492460000000
-0.999600000000000	0.215206000000000	4.90452260000000
-0.999500000000000	0.215206000000000	4.90452260000000
-0.999400000000000	0.215206000000000	4.90452260000000
-0.999300000000000	0.215206000000000	4.82412060000000
-0.999200000000000	0.215206000000000	4.90452260000000

Tabel 1

Spændingen fra Hall-sensoren, Volt1, anvendes. Når data plottes for et kort tidsrum ses støj omkring 5 volt

Data processeres nu med følgende Matlab-script:

```
close all
data = stepdata5;
t = data.second;
v = data.Volt1;
border1 = 4.5;
border2 = 0.1;
v2 = data.Volt;
b = 0;
stop = 0;
tp = [];
tpt = [];
vpt0 = [];
i = 1;
nyv = [];
while (i <= length(t))
    if (v(i)>border1)
        vn = 5;
    else
        vn = 0;
    end
    nyv = [nyv vn];
    i = i + 1;
end
i = 1;


while (i < length(t))
    if ((nyv(i+1)>border1 & nyv(i)<border2) & b == 0)
        t1 = t(i+1);
        stop = 1;
        b = 1;
    end
    i = i + 1;
    vpt0 = [vpt0 0];
    while (stop ~= 2 & b == 1 & i < length(t))
        if (stop == 1)
            while (stop ~= 2 & i < length(t))
                if (nyv(i+1)>border1 & nyv(i)<border2)
                    t2 = t(i+1);
                    stop = 2;
                end
                i = i + 1;
                vpt0 = [vpt0 0];
            end
        end
        if (stop == 2)
            period = t2-t1;
            tpt = [tpt t2];
            vpt0(i) = nyv(i);
            tp = [tp period];
            t1 = t2;
```

```
            stop = 1;
        end
    end
end
```

I scriptet fortolkes spænding over 4,5 V som 5 V. Markering af periodegrænser ved 5 Volt kan ses her (de røde prikker øverst):

Herudover udregnes tiden for hver omdrejningsperiode og gemmes i vektoren "tp". Ved plot at tp givet ved rpm overfor tid fås:

3.3.4.2 Overføringsfunktion

Udfra stepresponset skal følgende identificeres:

- Rise time, t_r
- Settling time, t_s
- Overshoot, M_p
- Peak time, t_p

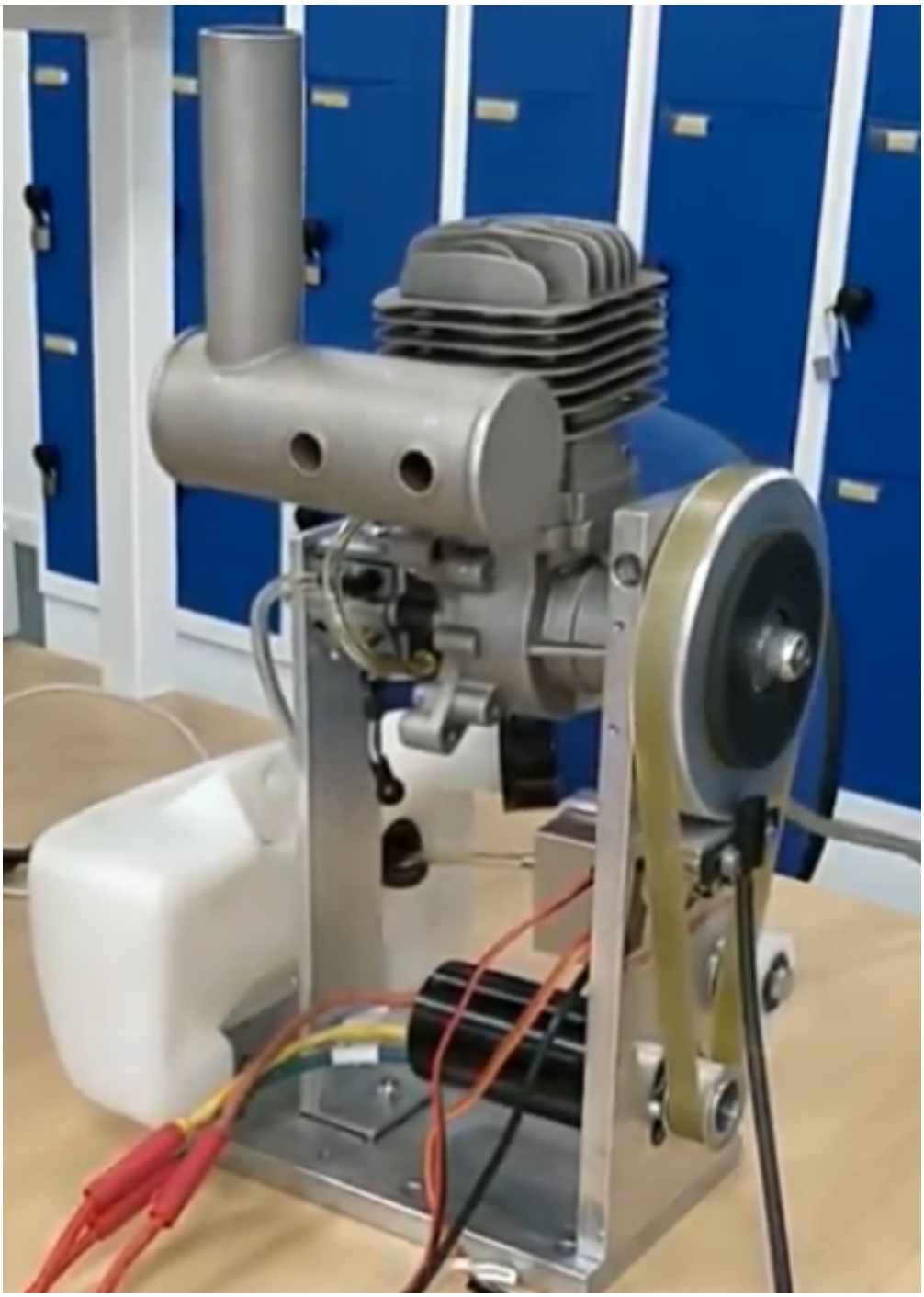
Scopet der har målt data gemmer også data op til stepresponset. Stepresponset ses ved tiden 0. Ved at applicere et midlingsfilter på 200 fås et indtryk af et steady-state niveau ved 8900, med start fra 7600, dvs 1300.

Ved at applicere et midlingsfilter på 10 fås et indtryk af en 1. grads overføringsfunktion.

Det måles at steady-state er opnået ved 0,19 sekunder. Dvs. $0.81 = 5\tau \Leftrightarrow \frac{0,19}{5} = \tau = 0,04$.

Hermed kan overføringsfunktionen sættes som

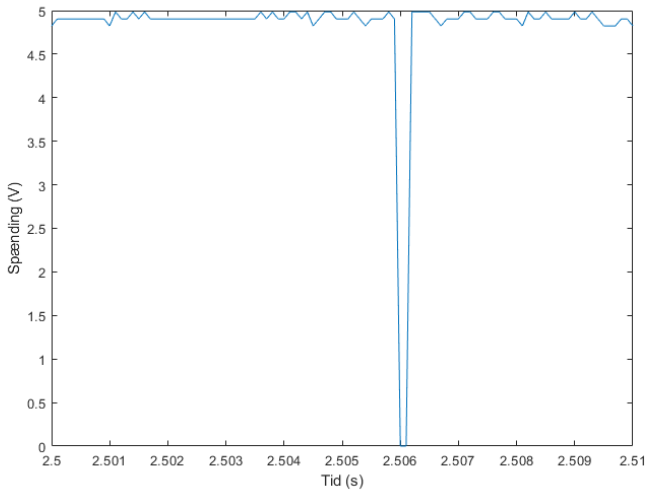
$$G(s) = \frac{1300}{0,04s + 1} \tag{2}$$



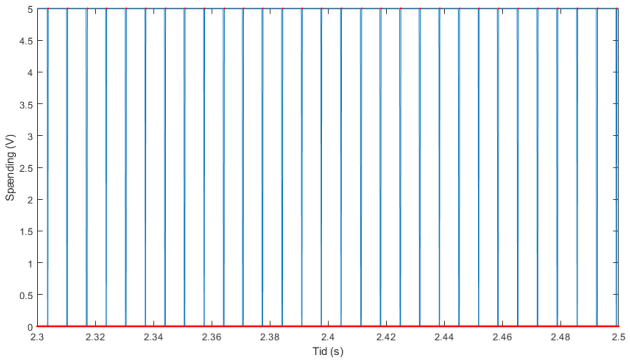
Figur 15: BLDC-motor forbundet til motoren med en tandrem

```
void init_read_rpm() {
        //set clock
        SIM->SCGC6 |= SIM_SCGC6_TPM1_MASK;
        SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;
        //set clock source to TPM
        SIM->SOPT2 |= (SIM_SOPT2_TPMSRC(1) | SIM_SOPT2_PLLFLLSEL_MASK);
        //load counter to max value 2^16
        TPM1->MOD = 0xffff;
        //set channel to input capture and enable intterupt
        TPM1->CONTROLS[TPM_CH].CnSC = TPM_CnSC_ELSA_MASK | TPM_CnSC_CHIE_MASK;
        //set pin to timer TPM
        PORTE->PCR[RPM_PIN] |= (3UL << 8);
        // Enable0 interrupts, with 128 prescaler. so count freq is 48MHz /128 = 375 kHz
        TPM1->SC = TPM_SC_CMOD(1) | TPM_SC_PS(7) | TPM_SC_TOIE_MASK;

        NVIC_SetPriority(TPM1_IRQn, 3);
        NVIC_ClearPendingIRQ(TPM1_IRQn);
        NVIC_EnableIRQ(TPM1_IRQn);

}
```

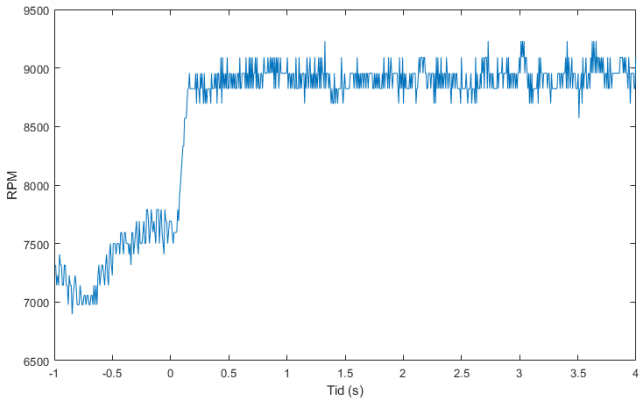
Figur 16: Kode til RPM-detektering



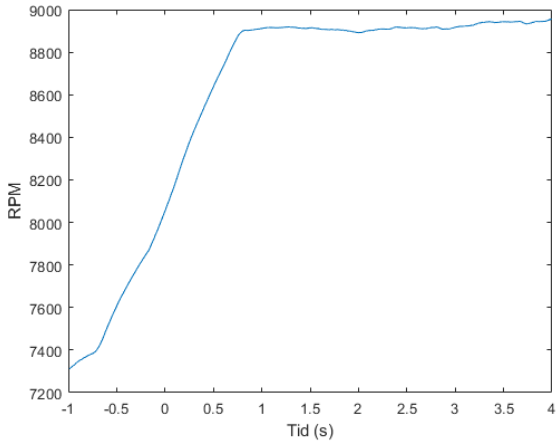
Figur 17



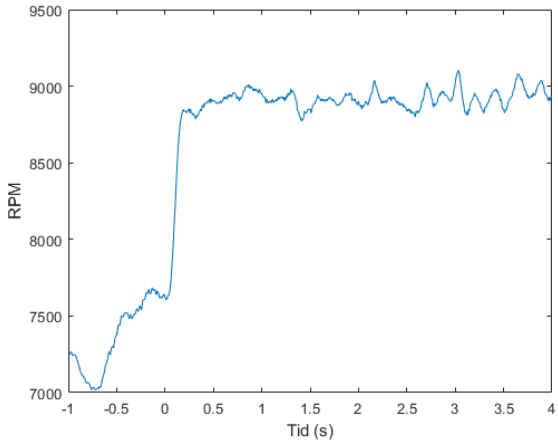
Figur 18



Figur 19

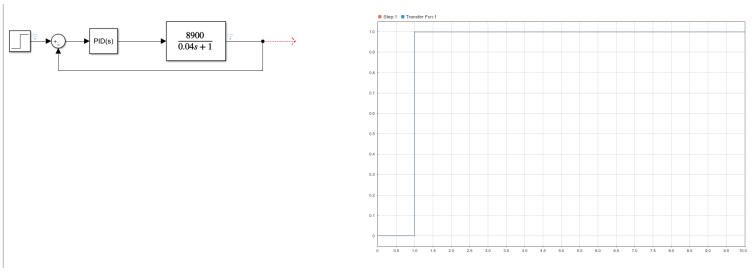


Figur 20



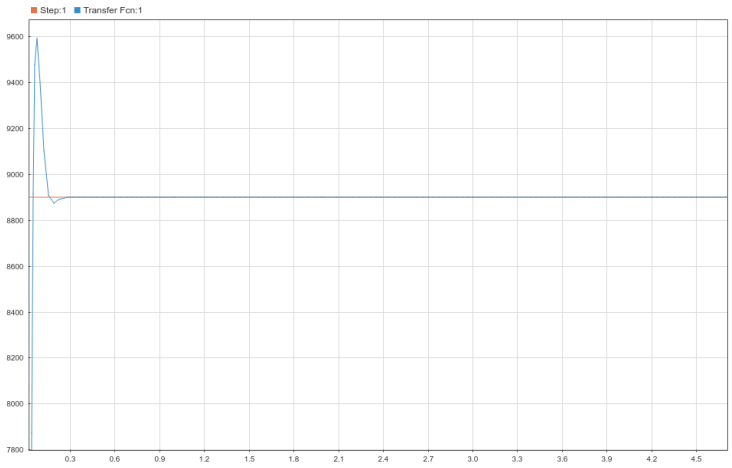
Figur 21

I Simulink oprettes følgende:



Figur 22: Simulink - diagram

Herefter laves autotuning i PID-modulet. Der findes koefficienter svarende til $P = 0,0010$, $I = 0,0514$, $D = -1,4946$ og følgende respons:



Figur 23: Simulink - diagram 2

3.3.5 Software (Simon)

Der udarbejdes nu et udkast til PID-kontrol software som også benytter sig af software til kontrol af servomotor samt aflæsning af motorens omdrejninger (se afsnit XXX).

```
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "rpm-detect.h"
#include "servo_driver.h"

// Filtrering
double alpha = 0.2;
```

```
double measuredSpeed = 0 ;

// Koefficienter
double kp = 0.001;
double ki = 0.0514;
double kd = -1.4946;
double K1;
double K2;
double K3;

// Vægtning af koefficienter
double setpointWeight = 0.2;
double lowpassSpeed;
double setpointSpeed;

// Diverse
double output;
double throttleopen = 1.0;
float throttlePos;

int rpmch;
int MAX_RPM = 9000;

// Initialiering
double lastSetpointSpeed = 0;
double lastMeasuredSpeed = 0;
double lastLowpassSpeed = 0;
double lastOutput = 0;
double lastLastMeasuredSpeed = 0;

void velPID (int setpointSpeed, int measuredSpeed) {
    lowpassSpeed = alpha * lastLowpassSpeed + (1-alpha ) * measuredSpeed;
    K1 = kp * setpointWeight * (setpointSpeed-lastSetpointSpeed) + kp * (
       lastMeasuredSpeed-lowpassSpeed);
    K2 = ki * (setpointSpeed-lowpassSpeed);
    K3 = kd * (2 * lastMeasuredSpeed-lowpassSpeed-lastLastMeasuredSpeed);
    output = lastOutput-K1-K2-K3;
    if (output < 0) {
      output = 0;
    }
    throttlePos = output/MAX_RPM;
    lastLowpassSpeed = lowpassSpeed;
    lastLastMeasuredSpeed = lastMeasuredSpeed;
    lastMeasuredSpeed = lowpassSpeed;
    lastSetpointSpeed = setpointSpeed;
    lastOutput = output;
    angle_throttle(throttlePos);
}

init_read_rpm()
```

```
int main(void) {
  /* Init board hardware. */
  BOARD_InitBootPins();
  BOARD_InitBootClocks();
  BOARD_InitBootPeripherals();
  /* Init FSL debug console. */
  BOARD_InitDebugConsole();
  rpmch = 7000;
  init_pwm();
  start();
  velPID(rpmch, measuredSpeed);
  return 0 ;
}
```

4 Resultater

4.1 Aktiv ensretter

OBS** Den røde tråd

- Max strøm dronebehov

4.2 Motorstyring

Der blev lavet implementering af PID-koden. Resultatet var desværre at der ved øgning af mindskning af omdrejningstal blev lavet en stigning i omdrejningstal og omvendt.