

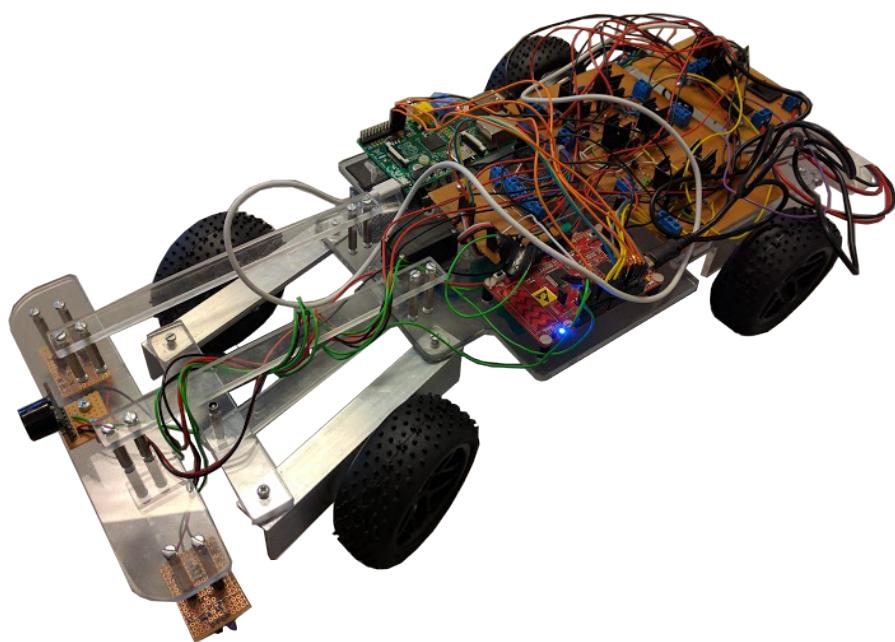
Projekt AutoCar

Dokumentation

Diplomingeniør Elektronik
4. Semesterprojekt forår 2016

Ingeniørhøjskolen Aarhus Universitet
Vejleder: Lars G. Johansen

27. maj 2016



Jonas Baatrup

Jonas Baatrup
Studienr. 201405146

Jesper Kloster

Jesper Kloster
Studienr. 201404571

Troels Brahe

Troels Ringbøl Brahe
Studienr. 20095221

Rasmus Platz

Rasmus Harboe Platz
Studienr. 201408608

Emil Jepsen

Emil Jepsen
Studienr. 20092013

Nicolai F

Nicolai H. Fransen
Studienr. 201404672

Nicolai B

Nicolai K. Bonde
Studienr. 201404519

Ansvarsområder

Tabellen nedenfor viser de primære ansvarsområder for hver af gruppens medlemmer.

Navn	Ansvarsområder
Jonas Baatrup	Motorstyring, Motor-design, Regulering
Nicolai Bonde	RPi kode, Bil-design, Forsyning
Jesper Kloster	Servo, Vejbanesensorer, Tachometer
Nicolai Fransen	Servo, Vejbanesensorer, Tachometer
Troels Brahe	Sonar, Regulering, RPi kode
Emil Jepsen	Motorstyring, Regulering, SPI
Rasmus Platz	TCP, GUI

Tabel 0.1: Tabel over ansvarsfordeling i projektet

Indhold

Indhold	1
1 Introduktion	3
1.1 Projektformulering	3
1.1.1 Motivation	3
1.1.2 Bilen	4
1.2 Systembeskrivelse	4
1.3 Systembetragtning	6
1.3.1 Motor	6
1.3.2 Styring	6
1.3.3 Kommunikation	6
2 Kravspecifikation	7
2.1 Aktørbeskrivelse	9
2.1.1 Aktør: Bruger	9
2.1.2 Aktør: Vejforhold	9
2.1.3 Aktør: Andre bilister	9
2.2 Fully dressed use cases	10
2.2.1 Use case 1 - Start bil	10
2.2.2 Use case 2 - Rapportér driftsstatus	12
2.2.3 Use case 3 - Ændr fart	13
2.2.4 Use case 4 - Følg vejbane	14
2.2.5 Use case 5 - Overhal	15
2.2.6 Use case 6 - Sluk bil	17
2.3 Ikke-funktionelle krav	18
3 Accepttest	19
3.1 Testudstyr	19
3.2 Tests	20
3.2.1 Test af ikke-funktionelle krav	26
4 Systemarkitektur	28
4.1 Hardwarearkitektur	28
4.1.1 Overordnet	28
4.2 Kontrolcenter	30
4.2.1 BDD Kontrolcenter	30
4.2.2 IBD Kontrolcenter	31
4.3 Sensorer	33
4.3.1 BDD Sensorer	33
4.3.2 IBD Sensorer	33
4.4 Strømforsyning	35
4.4.1 BDD Strømforsyning	35

4.4.2	IBD strømforsyning	35
4.5	Styring	37
4.5.1	BDD styring	37
4.5.2	IBD styring	37
4.6	Softwarearkitektur	39
4.6.1	Domæne- og applikationsmodeller	39
4.6.2	Use case 1 - Start bil	41
4.6.3	Use case 2 - Rapportér driftsstatus	41
4.6.4	Use case 3 - Ændr fart	42
4.6.5	Use case 4 - Følg vejbane	42
4.6.6	Use case 5 - Overhal	43
4.6.7	Use case 6 - Sluk bil	43
5	Design, implementering og test	45
5.1	Protokoller	45
5.1.1	SPI-Protokol	45
5.1.2	TCP-Protokol	45
5.1.3	TCP Statuskoder	46
5.2	Hardwaredesign, -implementering, og -test	48
5.2.1	Chassis	48
5.2.2	Hall-effekt sensor	53
5.2.3	Sonar	56
5.2.4	Vejbanesensor	58
5.2.5	ESC	60
5.2.6	Montering af Hall sensor	72
5.2.7	Servomotor	75
5.2.8	Forsyning	77
5.3	Softwaredesign, -implementering og -test	80
5.3.1	PSoC-software	80
5.3.2	Tachometer	87
5.3.3	Sonar	88
5.3.4	Strømbegrænsning	91
5.3.5	ESC	92
5.3.6	Servo	95
5.3.7	Controller	97
5.3.8	SPI	99
5.3.9	Print	101
5.3.10	Raspberry Pi	102
5.3.11	PC-software	112
5.4	Integrationstest	118
5.4.1	Sensorer, Servo og PSoC	118
5.4.2	GUI, Raspberry Pi, PSoC	121
6	Udført accepttest	128
6.1	Test af ikke-funktionelle krav	133

1 Introduktion

Dette dokument indeholder dokumentationen, for udviklingen af AutoCar. Dokumentationen er delt op i tre hoveddele – Kravspecifikation, Systemarkitektur og Design, implementering og test. Det beskriver problemet, der tager udgangspunkt i produktet. Samt de overvejelser og valg der er blevet taget i løbet af projektet.

1.1 Projektformulering

Projektformuleringen er her inddraget for helhedens skyld, og ridser igen op hvad motivationen samt tankerne til projektet er.

1.1.1 Motivation

Trafik i dag bliver mere og mere automatiseret, og ved at gøre biler mere automatiske kan mange af de menneskelige faktorer fjernes, som udgør en risiko i trafikken[1]. Tal viser at googles selvkørende bil, som er en af de førende på markedet, har 27% lavere risiko for at være involveret i et uheld målt over en længere periode.[2]

Ud over den forhøjede sikkerhed i selvkørende biler, kan trafikken optimeres mere generelt. Specielt i forbindelse med motorvejskørsel eller andre steder som skal afvikle store mængder trafik på kort tid. Trafikken kan køre mere optimalt grundet den større tilgængelige mængde information til bilen på et givet tidspunkt. Det vil eksempelvis mindske sikkerhedsafstanden mellem bilerne, da den ikke længere vil være afhængig af den enkelte persons reaktionsevne. En computer reagerer meget hurtigere og begår færre fejl[2].

Til sidst er der de miljømæssige aspekter i at have en bil som kører automatisk, og mere optimeret kørsel, herunder accelerationsminimering, vil mindske energiforbruget under kørsel.

1.1.2 Bilen

I projektet fremstilles en automatiseret bil, "AutoCar", som har en række egenskaber. Hvad skal projektet indeholde:

1. Bil
2. En eller flere sensorer til rumlig opfattelse
3. Indlejret platform
4. Computer til styring og interaktion
5. Trådløs kommunikation
6. Styring til motor og retning

Projektet skal indeholde sensorer til at give bilen en rumlig opfattelse, dvs. at den skal kunne forholde sig til sine omgivelser som forankørende "mini-biler", stillestående objekter i dens bane, vejbanens forløb, og om det er tilladt at overhale på den pågældende strækning. Dette skal foregå via sensorer der mäter afstanden til forankørende biler ved hjælp af sonar, vejbanesensorer til at måle efter "vejbanestriber" ved hjælp af farve, samt et tachometer til at måle farten. Til at kunne måle om det er tilladt at overhale inden for den angivne fartgrænse. Dette skal styres via en Indlejret platform i form af en mikroprocessor.

Bilen skal indeholde en motor til fremdrift, og en motor til at ændre bilens retning hvis den skal lave et sving eller vognbane-skift.

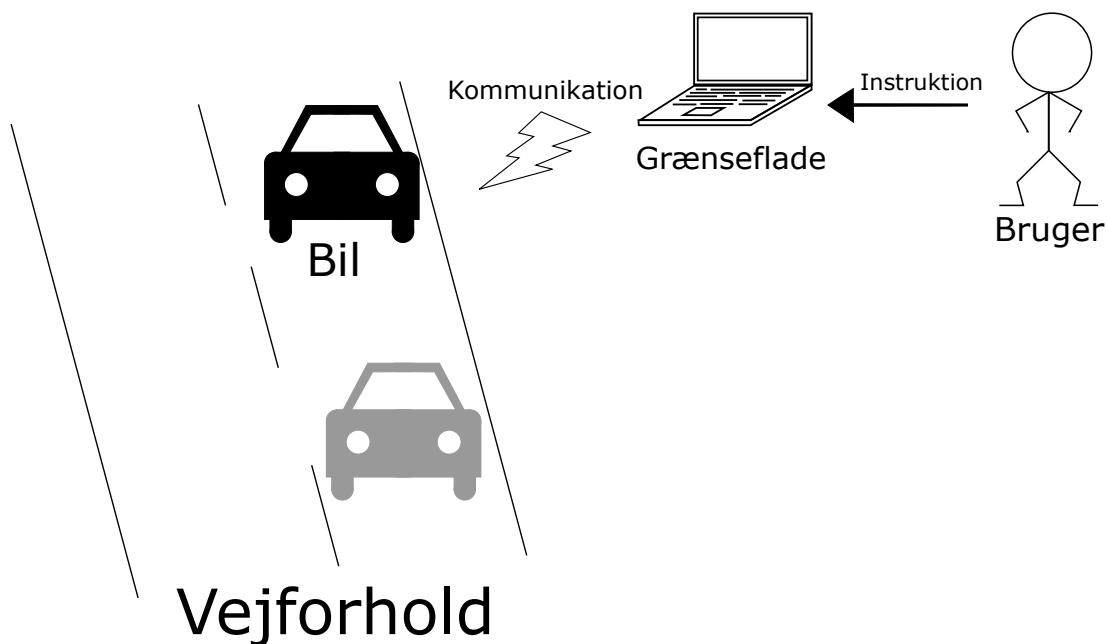
"Bilen" skal have forbindelse til en computer via trådløs kommunikationen. Med denne kommunikation kan bilen styres af en bruger, med kommandoer som f.eks. ændr fart, overhal.

Bilen skal som udgangspunkt kun kunne køre på motorvej, dvs. den ikke skal forholde sig til lyskryds, ændrende hastighedsbegrænsning, modkørende trafik osv.

1.2 Systembeskrivelse

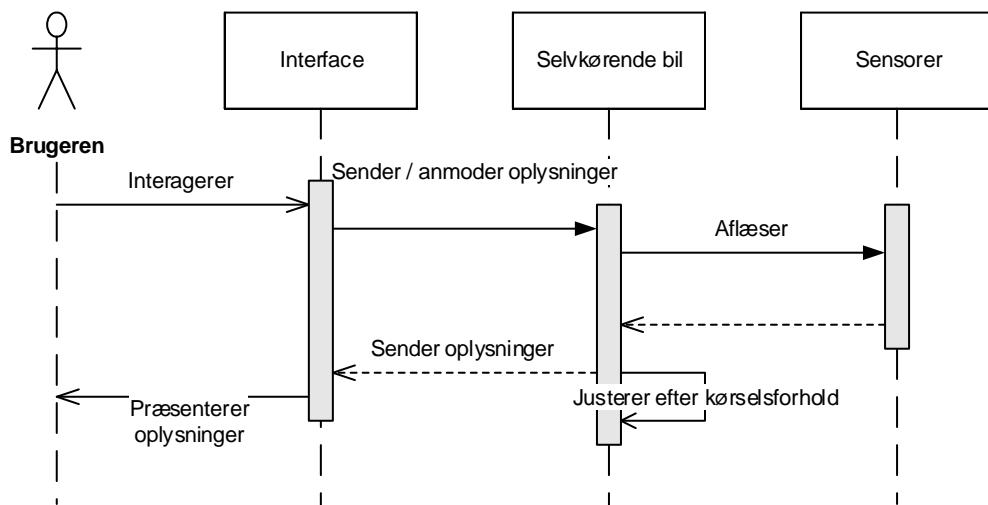
Der ønskes udviklet et system bestående af en selvkørende bil, hvor en bruger kan interagere med bilen med relevante parametre. Figur 1.1 viser et rigbillede af systemet, hvor konteksten er skitseret. Den selvkørende bil udvikles til motorvejskørsel. Det betyder at bilen skal følge trafikken, overhale og bremse. Bilen udvikles ikke til at køre i almindelig trafik. Det udvikles derfor ikke kørsel i kryds og navigation. Bilen implementeres som en model ca. i 1:10 størrelsesforhold.

Brugeren interagerer med en brugergrænseflade, hvorpå der kan sendes instruktioner til bilen. Bilen modtager beskederne fra grænsefladen og indstiller hvordan bilen agerer i forhold til vejforhold. Dette indebærer eksempelvis fart og om hvorvidt bilen skal overhale. Bilen kan rapportere informationer om drift og vejforhold tilbrugeren via brugergrænsefladen.



Figur 1.1: Rigbillede for systemkontekst

Figur 1.2 viser et overordnet sekvensdiagram, hvor flowet mellem bruger, den selv-kørende bil og sensorerne er skitseret grafisk i et hændelsesforløb.



Figur 1.2: Overordnet sekvensdiagram der viser flowet i interaktion mellem bruger og systemet

Interaktionen mellem AutoCar og sensorerne skal forstås som interaktionen til det omkringliggende miljø. For hver interaktion brugeren har med bilen vil bilen svare med et feedback, typisk i form af en besked på grænsefladen.

1.3 Systembetragtning

Der er forud for projektarbejdet lavet betragtninger om hvordan projektet skal udføres. Der er vurderet på hvilke elementer der skal inddrages for at projektet kan realiseres. Hvert element er opdelt i hvilke udfordringer og eventuelle kompetencer der skal tilstræbes. Systembetragtningen er således en opgørelse over hvilke overvejelser der er gået forud for projektet.

1.3.1 Motor

Motoren er en essentiel del af bilen, og er kritisk i forhold til at være dimensioneret til opgaven. De typiske motorer i fjernstyrede biler er almindelige brushed DC-motorer, som er forholdsvis nemme at forsyne. Der er valgt brushless motor i stedet til dette projekt. Valget er begrundet med ønsket om at erhverve erfaring med hvordan en sådan motor virker.

For at undgå mekanisk kompleksitet vil der, så vidt muligt, undgås at geare motoren. Gearing af motoren giver en del muligheder for valget af motor, men er også en udfordring som ligger uden for hvad der følt rimeligt for dette projekt. Motoren skal således kunne give forholdsvis stort kraftmoment ved lav hastighed.

Der er et behov for at motorens hastighed skal kunne måles samt indstilles og holdes forholdsvis konstant. Til dette er tiltænkt at der skal reguleres på motoren. Om muligt skal motoren modelleres og derved skal regulatoren dimensioneres.

1.3.2 Styring

Selve styretøjet for bilen afhænger af den mekaniske base der vælges til bilen. For at holde styringen simpel er det tænkt at bilens retning skal styres af servomotor.

Det er tænkt at i hver side af bilen er en vejbanesensor, der kan detektere "vejbane-striber". Oplagt valg er optiske sensorer.

1.3.3 Kommunikation

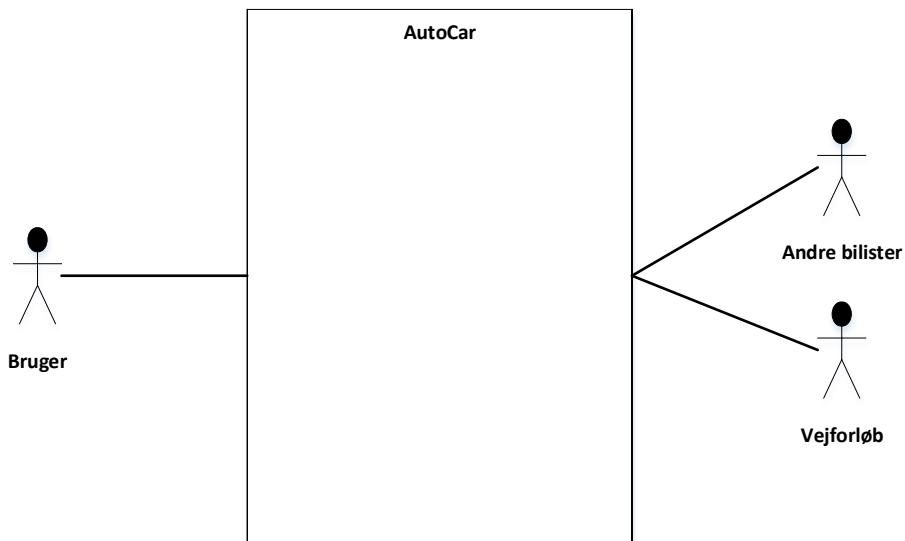
Det er tiltænkt at der kommunikeres trådløst til bilen via WiFi. Dette kan udvides til andre kommunikationsteknologier, eksempelvis GSM, hvilket gør det fleksibelt at udvikle på bilen. Desuden er det tiltænkt at der skal udvikles en kommunikationsprotokol til bilen, som danner grundlag for overførsel af data og kommandoer.

2 Kravspecifikation

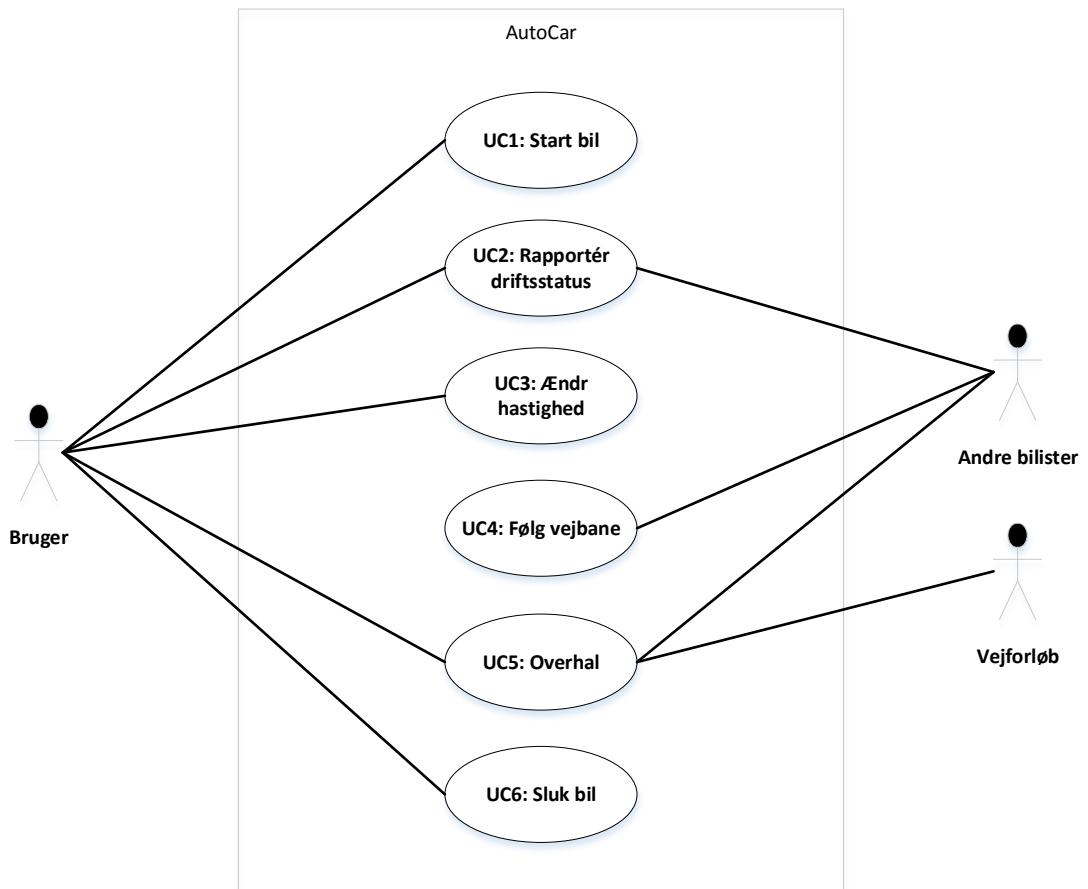
Kravene til produktet er prioriteret ved brug af MoSCoW metoden. Her er kravene for produktet inddelt i fire kategorier, hvor de vigtigste elementer er prioriteret højest. **Must** benævner de krav som er vigtigst at opfylde, og som er absolut nødvendigt for produktet. **Should** er de krav produktet bør opfylde. **Could** er kravene som produktet evt. kunne opfylde, hvis projektets tidsramme tillader det. **Won't** er krav som ikke vil blive opfyldt inden for projektets tidsrammer, men evt. kan tages med i senere iterationer.

Følgende opdeling viser kravene udvalgt for dette projekt:

- | | |
|---------------|---|
| Must | <ul style="list-style-type: none">– Køre fremad og dreje– Holde en fastsat brugerdefineret fart– Holde sig inden for vejstriber– Rapportere hastighed til bruger– Trådløskommunikere med brugeren |
| Should | <ul style="list-style-type: none">– Tilpasse fart efter forankørende– Monitorere og rapportere batteri niveau– Overhale ved brugerkommando– Stoppe ved forhindring på kørebanen |
| Could | <ul style="list-style-type: none">– Blinklys og kørelys– Undvige forhindring– Grafisk brugergrænseflade– Køre baglæns |
| Won't | <ul style="list-style-type: none">– Have navigation– Læse vejskilte– SmartPhone applikation til styring |



Figur 2.1: Aktør-kontekst diagram



Figur 2.2: Use case diagram

2.1 Aktørbeskrivelse

I det følgende afsnit beskrives systemets aktører. Ved hver aktør angives typen, samt en kort beskrivelse af aktørens funktion og/eller hvordan de påvirker systemet.

2.1.1 Aktør: Bruger

Type:

Primær

Beskrivelse:

Brugeren interagerer med systemet via et interface.

Han kan indstille den ønskede fart, samt kontrollere den nuværende fart og spændingen på batterierne

2.1.2 Aktør: Vejforhold

Type:

Sekundær

Beskrivelse:

Dette er de forhold sensorerne mäter på, i forbindelse med vejens forløb. Det er afstand til forankørende og registrering af vejbanestriber

2.1.3 Aktør: Andre bilister

Type:

Sekundær

Beskrivelse:

Dette er de forhold sensorerne mäter på, i forbindelse med de andre bilister. Det registrerer afstanden til forankørende

2.2 Fully dressed use cases

2.2.1 Use case 1 - Start bil

Mål:

Initiere bilen så den er klar til kørsel og er klar til at modtage input

Initiering:

Bruger

Aktører:

Bruger (primær)

Referencer:

Ingen

Samtidige forekomster:

En

Forudsætning:

Bilen er slukket og der er forbindelse fra interface til bil

Resultat:

Bilens sensorer er tændt, motorer er klar, bilen holder stille

Hovedscenarie:

1. Bruger vælger via interface "Start bil"
2. Bilen monitorerer sensorinputs og rapporterer status
3. Bilen udfører motortjek ved at køre bilen lidt frem og derefter tilbage
4. Bilen rapporterer status
5. Bilen tænder for- og baglys, blinker med blinklys hvis status er OK

Extension 1: Status ikke OK

6. Bilen afventer brugerinput

Extensions:

Extension 1: Status ikke OK

1. Bilen rapporterer fejl og forsøger at angive hvilken sensor og/eller motor der fejler

2.2.2 Use case 2 - Rapportér driftsstatus

Mål:

Informerer bruger om batteriniveau, bilens fart og afstand til objekt foran

Initiering:

Bruger

Aktører:

Bruger (primær)

Vejforhold Andre bilister

Referencer:

Ingen

Samtidige forekomster:

En

Forudsætning:

Der er forbindelse fra interface til bil

Resultat:

Brugeren er blevet informeret om sensorstatus

Hovedscenarie:

1. Brugeren anmelder om driftsstatus via interface
2. Bilen aflæser aktuel sensordata

Extension 1: Bilen er slukket

3. Data udskrives til bruger via interface

Extensions:

Extension 1: Bilen er slukket

1. Brugeren informeres om batteriniveau via interface
2. Bilen rapporterer til brugeren at bilen er slukket via interface

2.2.3 Use case 3 - Ændr fart

Mål:

Brugeren indstiller maksimale fart bilen må køre med

Initiering:

Bruger

Aktører:

Bruger (primær), vejforhold

Referencer:**Samtidige forekomster:**

En

Forudsætning:

Use case 1 hovedscenarie - Start bil er fuldført

Resultat:

Bilens maksimale fart er ændret

Hovedscenarie:

1. Brugeren indstiller en fart via interface
2. Bilen justerer sin fart til den ønskede værdi

Extension 1: Ønsket fart er højere end den højest tilladte på 13km/t

Extensions:

Extension 1: Ønsket fart er højere end tilladt

1. Farten indstilles til højest tilladt værdi
2. Advarsel om for høj valgt fart vises på interface

2.2.4 Use case 4 - Følg vejbane

Mål:

Bilen følger vejbanen

Initiering:

Use case 3 - Ændr fart

Aktører:

Vejforhold, andre bilister

Referencer:

Use case 3 - Ændr fart

Use case 5 - Overhal

Samtidige forekomster:

En

Forudsætning:

Use case 3 - Ændr fart - farten er forskellig fra 0

Resultat:

Bilen følger autonomt vejbanen og agerer på forhindringer

Hovedscenarie:

1. Bilen korrigerer retning i forhold til vejbane
2. Bilen bevæger sig med indstillede fart

Extension 1: Forhindring på vejbane

Extensions:

Extension 1: Forhindring på vejbane

1. Bilen detekterer langsomt kørende / stillestående objekt
2. Bilen regulerer til samme fart som forankørende objekt

2.2.5 Use case 5 - Overhal

Mål:

Bilen overhaler objekt

Initiering:

Bruger

Aktører:

Bruger (primær)
Vejforhold
Andre bilister

Referencer:

Use case 4 - Følg vejbane

Samtidige forekomster:

En

Forudsætning:

Use case 4 - Følg vejbane er undervejs

Resultat:

Bilen har passeret forankørende objekt og befinner sig i samme vognbane som før

Hovedscenarie:

1. Brugeren anmoder om overhaling via interface
2. Bilen vurderer om overhaling er mulig

Extension 1: Overhaling ikke mulig

3. Bilen foretager overhaling
4. Bilen ændrer fart til den valgte af brugeren

Extensions:**Extension 1: Overhaling ikke mulig**

1. Bilen vurderer at den forudindstillede fart ikke er tilstrækkelig
2. Bilen informerer brugeren at farten ikke er tilstrækkelig til at overhale

3. Bilen holder samme fart

2.2.6 Use case 6 - Sluk bil

Mål:

Bilens systemer lukkes ned

Initiering:

Bruger

Aktører:

Bruger (primær)

Referencer:**Samtidige forekomster:**

En

Forudsætning:

Use case 1 - Start bil er udført

Resultat:

Bilen er slukket

Hovedscenarie:

1. Bruger vælger "sluk bil" via interface
2. Bilen sætter fart til nul
3. Bilen slukker sensorer, lys og motorer
4. Der udskrives på interface at bilen er slukket

2.3 Ikke-funktionelle krav

I dette afsnit beskrives de ikke-funktionelle krav. Her opstilles f.eks. krav om præcision, brugervenlighed samt produktets dimensioner.

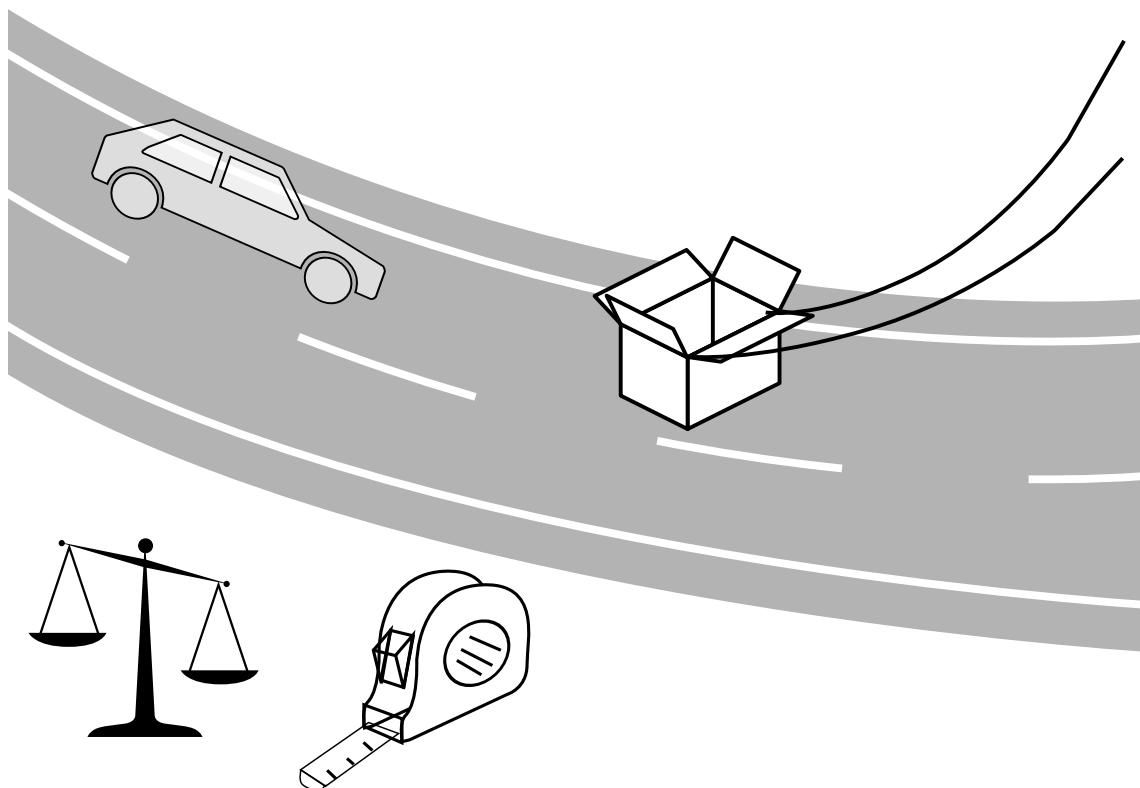
- Bilens længde og bredde må ikke overskride 50cm x 30 cm
- Bruger skal have mulighed for at kommunikere med bilen via tekst-terminal og / eller grafisk brugergrænseflade
- Bilen skal detektere objekter indenfor intervallet 20cm til 2,5m
- Bilen skal kunne køre med en fart på 13 km/t \pm 0,5
- Bilen skal have en maksimal vægt på 2kg
- Bilen skal måle sin fart i intervallet 0-13 km/t \pm 0,5
- Bilen skal kunne indstille sin fart i intervallet 0-13 km/t med en opløsning på 0,5 km/t

3 Accepttest

Accepttesten udføres ved brug af en skaleret tosporet motorvej i samme størrelsesforhold som bilen: 1:10. Den konstrueres vha. hvid tape, som repræsenterer vejbanestriberne. Vejen skal være 35cm bred, og derudover skal der laves et lige vejbanestykke på minimum 20m. Efter dette stykke konstrueres vejen således at sving svarer til forholdene på en dansk motorvej.

3.1 Testudstyr

- Hvid tape
- Papkasse i samme størrelse som bil (benævnes som testobjekt)
- Målebånd
- Vægt



Figur 3.1: Opstilling for accepttest

3.2 Tests

Use case under test	Use case 1 - Start bil			
Scenarie	Hovedscenarie			
Prækondition	Bilen er slukket, brugeren er forbundet til bilen			
Step	Handling	Forventet	Faktisk	Vurdering
1	Vælg "Start bil" via interface	Statusrapport udskrives. For- og baglys tændes, blinklys blinker		

Tabel 3.1: Test for Use case 1 - Start bil - Hovedscenarie

Use case under test	Use case 1 - Start bil			
Scenarie	Extension 1: Status ikke OK			
Prækondition	Vejbanesensor er disconnected			
Step	Handling	Forventet	Faktisk	Vurdering
1	Vælg "Start bil" via interface	Interfacet melder fejl på vejbane-sensor, og bilen er ikke startet		

Tabel 3.2: Test for Use case 1 - Start bil - Extension 1: Status ikke OK

Use case under test	Use case 2 - Rapporter driftsstatus			
Scenarie	Hovedscenarie			
Prækondition	Brugeren er forbundet til bilen, bilen er tændt			
Step	Handling	Forventet	Faktisk	Vurdering
1	Vælg "driftsstatus" via interface	Data vises på interface		

Tabel 3.3: Test for Use case 2 - Rapporter driftsstatus - Hovedscenarie

Use case under test	Use case 2 - Rapporter driftsstatus			
Scenarie	Extension 1: Bilen er slukket			
Prækondition	Bilen er slukket, brugeren er forbundet via interface			
Step	Handling	Forventet	Faktisk	Vurdering
1	Vælg "driftsstatus" via interface	Batteriniveau udskrives. Bilen rapporterer at den er slukket via interface		

Tabel 3.4: Use case 2 - Rapporter driftsstatus - Extension 1: Bilen er slukket

Use case under test	Use case 3: Ændr fart			
Scenarie	Hovedscenarie			
Prækondition	Use case 1 er fuldført (Bilen er i drift)			
Step	Handling	Forventet	Faktisk	Vurdering
1	Angiv fart via interface og vent	Bilen justerer sin fart		
2	Vælg "Driftsstatus" via interface	Bilen rapporterer fart svarende til den angivne værdi		
3	Angiv fart på 0 km/t via interface	Bilen stopper		

Tabel 3.5: Test for Use case 3 - Ændr fart - hovedscenarie

Use case under test	Use case 3 - Ændr fart			
Scenarie	Extension 1: Ønsket fart er højere end tilladt			
Prækondition	Use case 1 er fuldført (Bilen er i drift)			
Step	Handling	Forventet	Faktisk	Vurdering
1	Angiv en fart på 15 km/t via interface	Bilen advarer om at den angivne fart er for høj. Bilen justerer til højest tilladte fart.		
2	Vælg "Driftsstatus" via interface	Bilen rapporterer fart tilsvarende den højest tilladte værdi		

Tabel 3.6: Test for use case 3 - Ændr fart

Use case under test	Use case 4 - Følg vejbane			
Scenarie	Hovedscenarie			
Prækondition	Bilen er i drift. Brugerne er forbundet via interface			
Step	Handling	Forventet	Faktisk	Vurdering
1	Angiv en fart på 5 km/t	Bilen bevæger sig og holder sig inden for vejbanen		
2	Vælg "Driftsstatus" via interface	Bilen rapporterer fart tilsvarende den angivne værdi		

Tabel 3.7: Test for Use case 4 - Følg vejbane - Hovedscenarie

Use case under test	Use case 4 - Følg vejbane			
Scenarie	Extensions 1: Forhindring på vejbane			
Prækondition	Bilen er i drift og holder stille. Bruger forbundet via interface			
Step	Handling	Forventet	Faktisk	Vurdering
1	Angiv en fart på 5 km/t via interface	Bilen bevæger sig og holder sig inden for vejbanen		
2	Testobjekt bevæges i samme retning foran bilen med lavere fart	Bilen indhenter testobjektet og justerer sin fart ned til testobjektets		
3	Testobjektet standses	Bilen stopper inden testobjektet rammes		

Tabel 3.8: Test for Use case 4 - Følg vejbane - Extension 1: Forhindring på vejbane

Use case under test		Use case 5 - Overhal		
Scenarie		Hovedscenarie		
Prækondition		Bilen er i drift. Brugeren er forbundet via interface		
Step	Handling	Forventet	Faktisk	Vurdering
1	Angiv en fart på 5 km/t via interface	Bilen bevæger sig		
2	Testobjekt bevæges i samme retning foran bilen med lavere fart	Bilen indhenter testobjektet og justerer sin fart ned til testobjektets		
3	Vælg "Overhal" via interface	Bilen trækker ud og accelererer til den er passeret testobjektet. Bilen trækker derefter ind og fortsætter med den angivne fart.		

Tabel 3.9: Test af Use case 5 - Overhal - Hovedscenarie

Use case under test		Use case 5 - Overhal		
Scenarie		Extension 1: Overhaling ikke mulig		
Prækondition		Bilen er i drift. Brugeren er forbundet via interface		
Step	Handling	Forventet	Faktisk	Vurdering
1	Angiv en fart på 5 km/t via interface	Bilen bevæger sig		
2	Testobjekt bevæges i samme retning foran bilen med lidt lavere fart end bilen	Bilen indhenter testobjektet og justerer sin fart ned til testobjektets		
3	Vælg "Overhal" via interface	Bilen informerer brugeren at farten ikke er tilstrækkelig. Bilen fortsætter med samme fart.		

Tabel 3.10: Test af Use case 5 - Overhal - Extension 1: Overhaling ikke mulig

Use case under test	Use case 6 - Sluk bil			
Scenarie	Hovedscenarie			
Prækondition	Bilen er i drift. Brugerens er forbundet via interface			
Step	Handling	Forventet	Faktisk	Vurdering
1	Angiv en fart på 5 km/t via interface	Bilen bevæger sig		
2	Vælg "Sluk bil" via interface	Bilen stopper. Bilens lys slukkes. Der udskrives på interfacet at bilen er slukket.		

Tabel 3.11: Test af Use case 6 - Sluk bil - Hovedscenarie

3.2.1 Test af ikke-funktionelle krav

Krav	Test	Forventet resultat	Resultat	Vurdering
Bilens længde og bredde må ikke overskride 50cm x 30cm	Længde og bredde måles	Længden og bredden overskrider ikke 50cm x 30cm		
Brugeren skal have mulighed for at kommunikere med bilen via tekst-terminal og / eller grafisk brugergrænse-flade				
Bilens skal detektere objekter på en afstand i intervallet 20cm til 2,5m i spring af 5cm	Der placeres et objekt 20cm foran bilen. Driftsstatus vælges og aflæses via interface. Objektet føres ud i en afstand 2,5m i skridt af 5cm og driftsstatus vælges og aflæses via interface.	Detektion i intervallet fra 20cm til 2,5m med oplosning på 5cm		
Bilen skal kunne køre med en fart på mindst 13 km/t \pm 0,5	Der afmåles et vejbanestykke på 20m og et på 10m i forlængelse af hinanden. Der angives en fart på 13 km/t. Idet bilen passerer de første 20m startes et stoppur og stoppes efter bilen har bevæget sig yderligere 10m	Bilen kan køre 13 km/t \pm 0,5 km/t		

Krav	Test	Forventet resultat	Resultat	Vurdering
Bilen skal have en maksimal vægt på 2 kg	Bilen placeres på en vægt	Vægten er under 2 kg		
Bilen skal måle sin fart med en opløsning på 0,5 km/t og bilen skal kunne indstille sin fart med en opløsning på 0,5 km/t	Der afmåles et vejbanestykke på 20m og et på 10m i forlængelse af hinanden. Bilens indstilles til forskellige hastigheder fra 0 km/t til 13 km/t i trin af 0,5 km/t. Idet bilen passerer de første 20m startes et stopur og stoppes idet den har bevæget sig yderligere 10m. Der læses fart ud fra driftsstatus i løbet af de sidste 10m.	Bilen kan aflæse og indstille sin fart med en opløsning på 0,5 km/t		

4 Systemarkitektur

4.1 Hardwarearkitektur

Følgende afsnit indeholder SysML BDD og IBD. BDD bruges til at give overordnet overblik over systemet, samt hvad de enkelt hardware blokke indholder af enheder og forbindelser. IBD viser de interne forbindelser i det overordnede system, samt i de enkelte blokke.

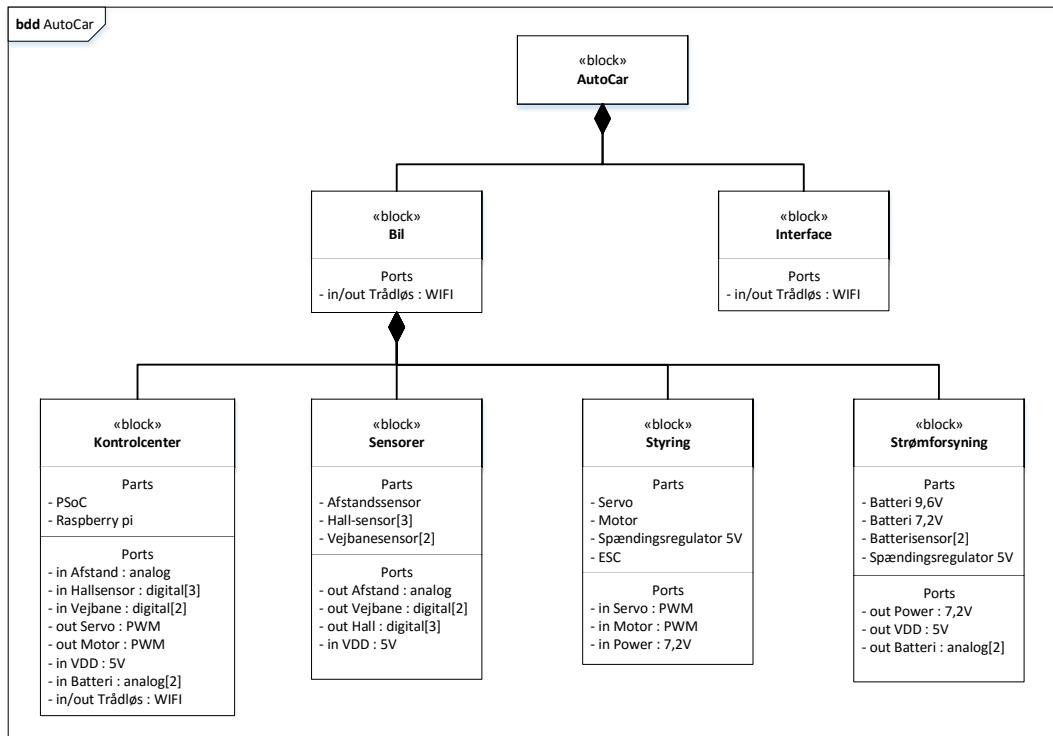
4.1.1 Overordnet

"Overordnet" giver et generelt overblik over systemet, det viser at det meste af vores system ligger i "bil" mens interface kun har en WiFi forbindelse til "Bil". "Bil" er delt ind i fire hoveddele, kontrolcenter som er vores PSoC og Raspberry Pi. Sensor som indeholder vejbanesensor, tachometer samt afstandssensor. Styring som er motor, servo, ESC samt 11V til 5V regulator. Strømforsyning består af to batterier på 7.2V og 9.6V, en 9.6V og 7.2V til 5V spændingsregulator samt to batterisensorer.

Udover det så indeholder systemet et interface, i form af et GUI på en PC. GUI'et er forbundet til systemet via en WiFi forbindelse som sender de nødvendige data.

4.1.1.1 BDD Overordnet

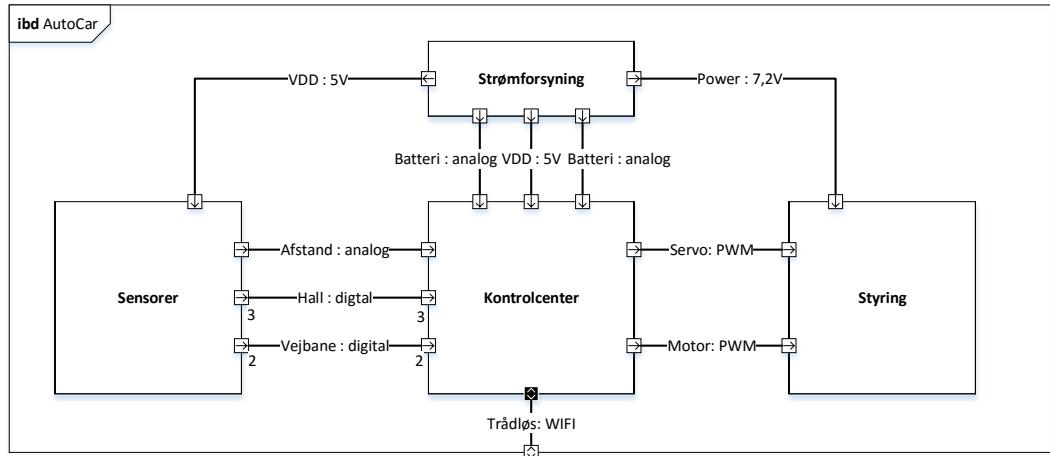
Figur 4.1 viser et Block Definitions Diagram (BDD) over det samlede system. Det er med for, at give det overordnede overblik over systemet – altså hvilke blokke der er en del af systemet. BDD'et er det første springbræt til at lave de fleste af de kommende diagrammer. Ydermere giver BDD'et en overordnet forståelse af systemet.



Figur 4.1: Overordnet BDD

4.1.1.2 IBD Overordnet

Figur 4.2 viser et overordnet IBD over bilen. Her vises de interne forbindelser, som er vigtig for det videre arbejde med hardwaren. Det er brugbart hvis der senere skal dannes et overblik, over forbindelserne mellem de enkelte enheder.



Figur 4.2: Overordnet IBD

4.1.1.3 Signalbeskrivelse

Tabel 4.1 viser signal beskrivelse for det Overordnet system.

Signal type	Navn	Beskrivelse
5V	VDD	5V forsyningsspænding
7,2V	power	7,2V forsyning til motor og servo
0.9V-3.3V	afstand	Analogt signal fra afstandssensor
CMOS	hall	Digitalt signal fra hall-sensorne, CMOS standard
CMOS	vejbane	Digitalt signal fra vejbanesensorne, CMOS standard
PWM	servo	PWM signal til styring af servo
PWM	motor	PWM signal til styring af motor
0V-5V	batteri	Analogt signal fra batterisensor
WiFi	trådløs	Trådløs kommunikation mellem kontrolcenter og interface

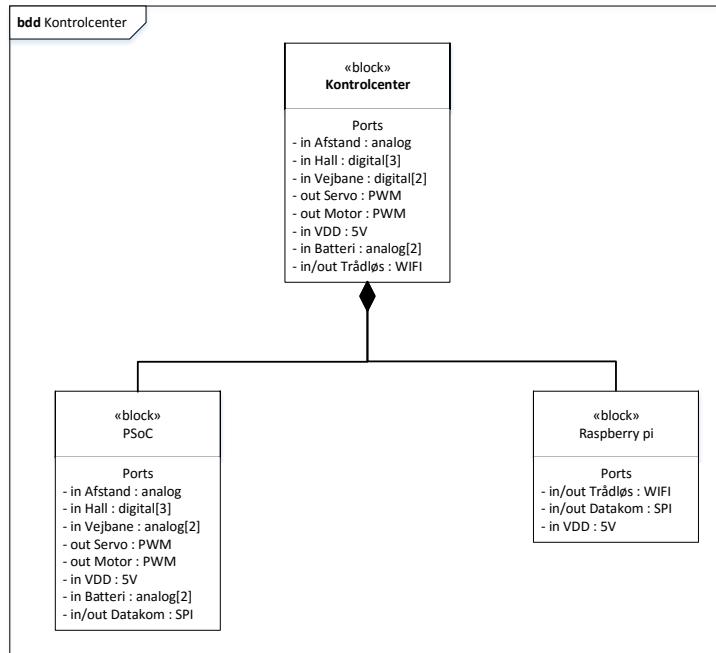
Tabel 4.1: signalbeskrivelse for overordnet IBD

4.2 Kontrolcenter

Kontrolcenter er den centrale styreenhed på bilen. Den indeholder en PSoC, som udgør grænseflade mellem sensorer og Raspberry Pi. Raspberry Pi udgør den overordne styring af bilen. Det er den som modtager data fra interface, og det her er alt behandling af data foregår. Derudover er det her alle kommandoer til resten bilen bliver sendt fra.

4.2.1 BDD Kontrolcenter

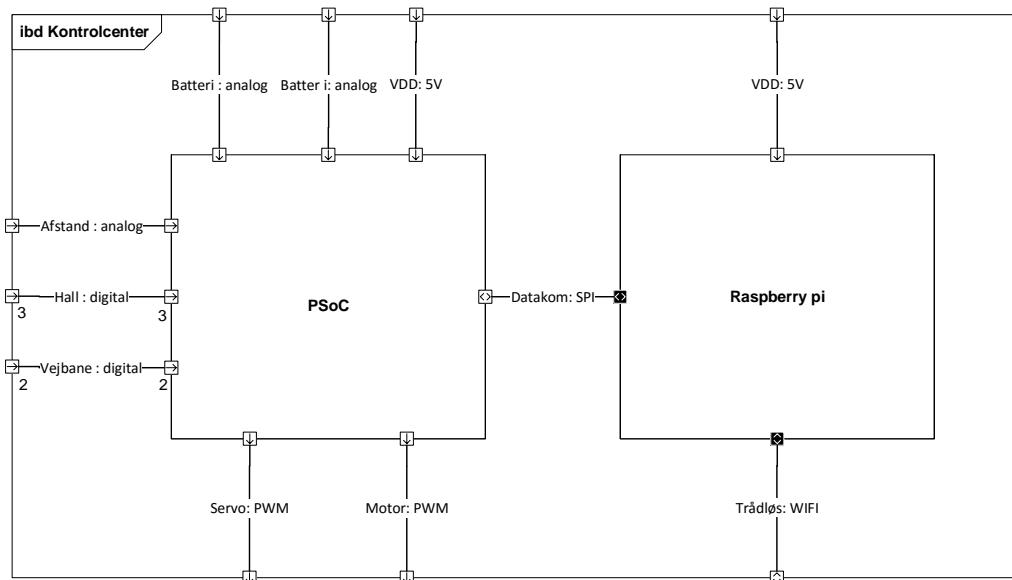
Figur 4.3 viser BDD for kontrolcenter, dette viser hardwaren i forbindelse med kontrolcenteret. Det er med for at give et bedre overblik over hvad Kontrolcenter indeholder.



Figur 4.3: Kontrolcenter BDD

4.2.2 IBD Kontrolcenter

Figur 4.4 viser IBD over Kontrolcenter. IBD'et giver et overblik over de interne forbindelser mellem de individuelle hardwaredele, hvilket er essentielt i den kommende hardwaredesignproces.



Figur 4.4: Kontrolcenter IBD

4.2.2.1 Signalbeskrivelse

Tabel 4.2 viser signal beskrivelse for Kontrolcenter.

Signal type	Navn	Beskrivelse
5V	VDD	5V forsyningsspænding
0.9V-3.3V	afstand	Analogt signal fra afstandssensor
CMOS	Hall	Digitalt signal fra Hall effekt-sensorerne, CMOS standard
CMOS	vejbane	Digitalt signal fra vejbanesensorne
PWM	servo	PWM signal til styring af servo
PWM	motor	PWM signal til styring af motor
0V-5V	batteri	Analogt signal fra batterisensor
WIFI	trådløs	Trådløs kommunikation mellem kontrolcenter og interface
SPI	datakom	Datakomunikation mellem Psoc og Raspberry pi

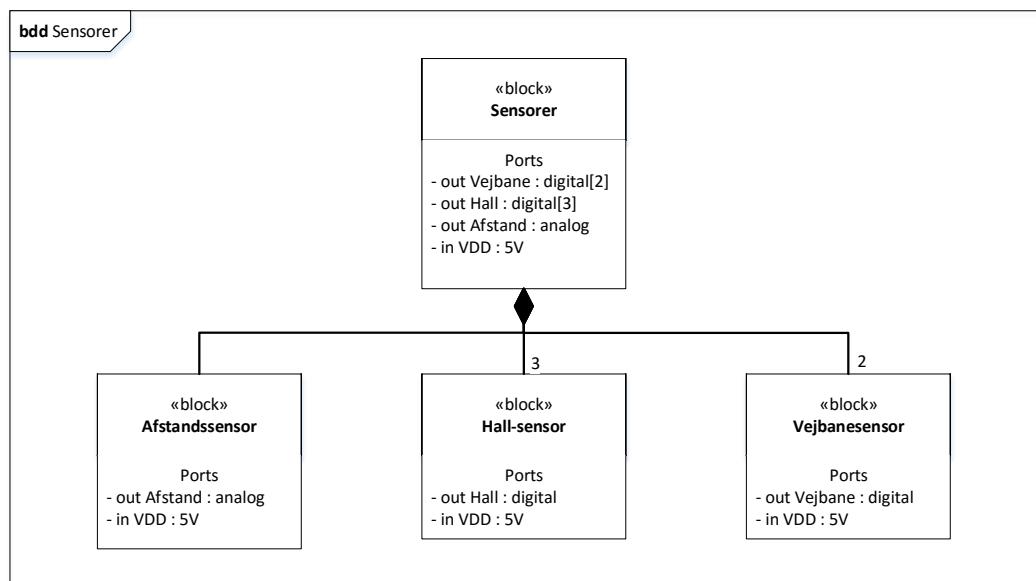
Tabel 4.2: Signalbeskrivelse for Kontrolcenter IBD

4.3 Sensorer

Sensor indeholder alle de hardware-komponenter som mäter på omgivelserne. Afstandssensor fungerer vha. sonar som mäter afstanden til evt. forankørende biler eller objekter på vejbanen. Tachometer bruges til at bestemme omdrejningstallet af hjulet og regulering af motoromdrehning. De to vejbanesensorer er IR-sensorer som bruges til at registrere vejstriberne, for at kontrollere at bilen holder sig inden for vejbanen, samt udførelse af en overhaling.

4.3.1 BDD Sensorer

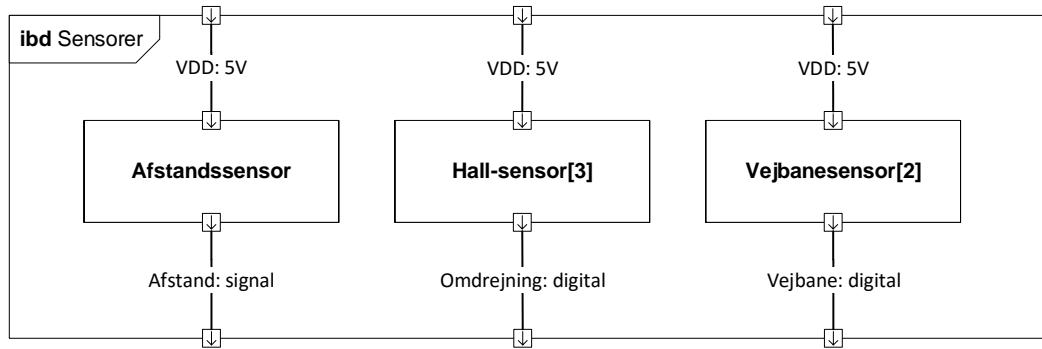
Figur 4.5 viser BDD for Sensorer, dette viser hardwaren i forbindelse med sensorerne. Det er med for at give et bedre overblik over hvad Sensorer indeholder.



Figur 4.5: Sensorer BDD

4.3.2 IBD Sensorer

Figur 4.6 viser IBD over Sensorer. IBD'et giver et overblik over de interne forbindelser mellem de individuelle hardwaredele, hvilket er essentielt i den kommende hardware-designproces.



Figur 4.6: Sensorer IBD

4.3.2.1 Signalbeskrivelse

Tabel 4.3 viser signalbeskrivelse for Sensorer.

Signal type	Navn	Beskrivelse
5V	VDD	5V forsyningsspænding
0.9V-3.3V	afstand	Analogt signal fra afstandssensor
CMOS	Hall	Digitalt signal fra Hall effekt-sensorne, CMOS standard
CMOS	vejbane	Digitalt signal fra vejbanesensorne, CMOS standard

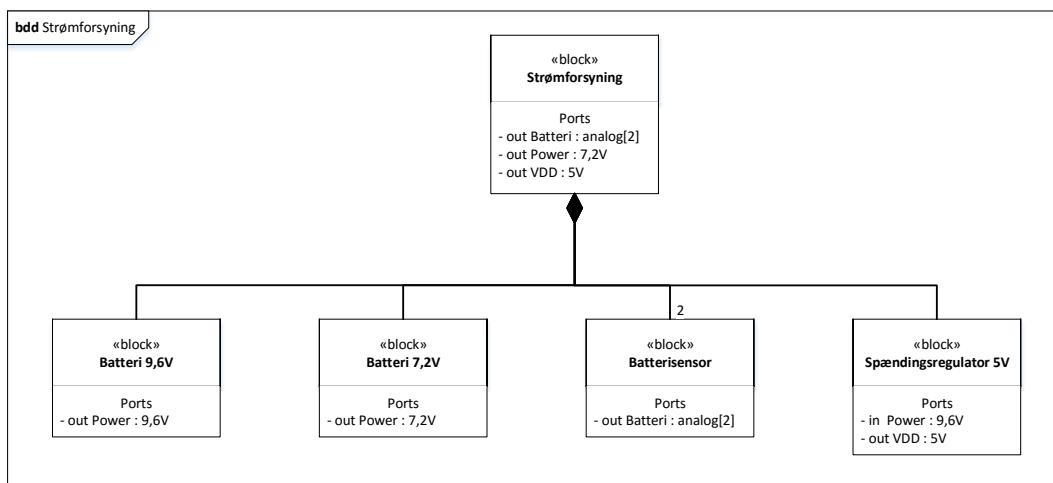
Tabel 4.3: Signalbeskrivelse for Sensorer IBD

4.4 Strømforsyning

Strømforsyning består primært af batterier, og er medtaget på grund af batterisensorerne, som mäter det aktuelle batteriniveau.

4.4.1 BDD Strømforsyning

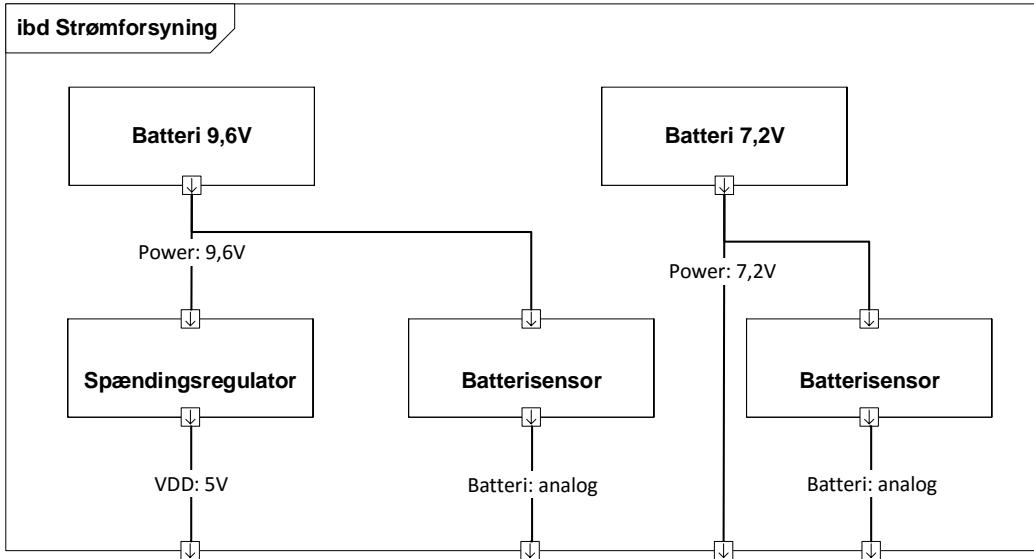
Figur 4.7 viser BDD for Strømforsyning, dette viser hardwaren i forbindelse med Strømforsyning. Det er med for at give et overblik over hvad Strømforsyning indeholder.



Figur 4.7: Strømforsyning BDD

4.4.2 IBD strømforsyning

Figur 4.8 viser IBD over Strømforsyning. IBD'et giver et overblik over de interne forbindelser mellem de individuelle hardwaredele, hvilket er essentielt i den kommende hardwaredesignproces.



Figur 4.8: Strømforsyning IBD

4.4.2.1 Signalbeskrivelse

Tabel 4.4 viser signalbeskrivelse for Strømforsyning.

Signal type	Navn	Beskrivelse
5V	VDD	5V forsyningsspænding
9,6	VCC	9,6V batterispænding
7,2	power	7,2V forsyning til motor og servo
0V-5V	batteri	Analogt signal fra batterisensorerne

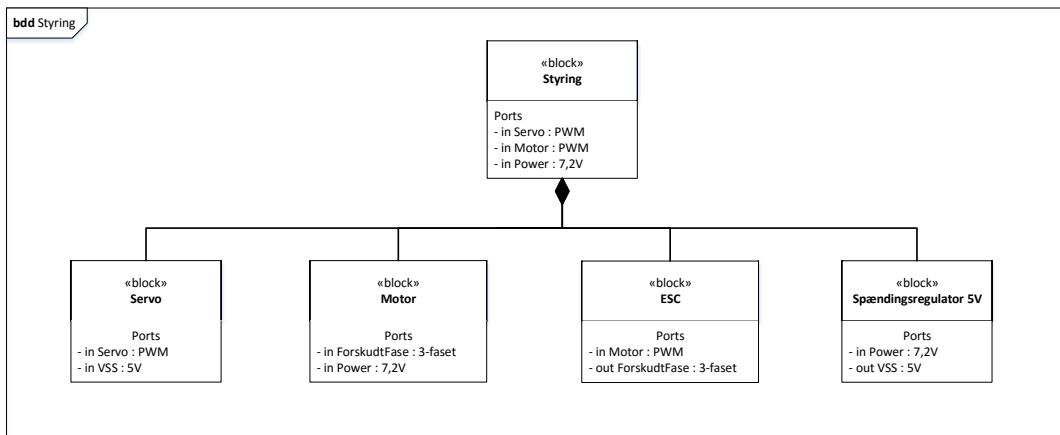
Tabel 4.4: signalbeskrivelse for Strømforsyning IBD

4.5 Styring

Styring indeholder motor til fremdrift af hjulene og servo bruges til at styre styretøjet. En spændingsregulator som regulerer fra 7.2V ned til 5V som servoen skal bruge. Derudover indeholder den en ESC som omdanner et PWM-signal til et 3-faset AC-signal til motoren.

4.5.1 BDD styring

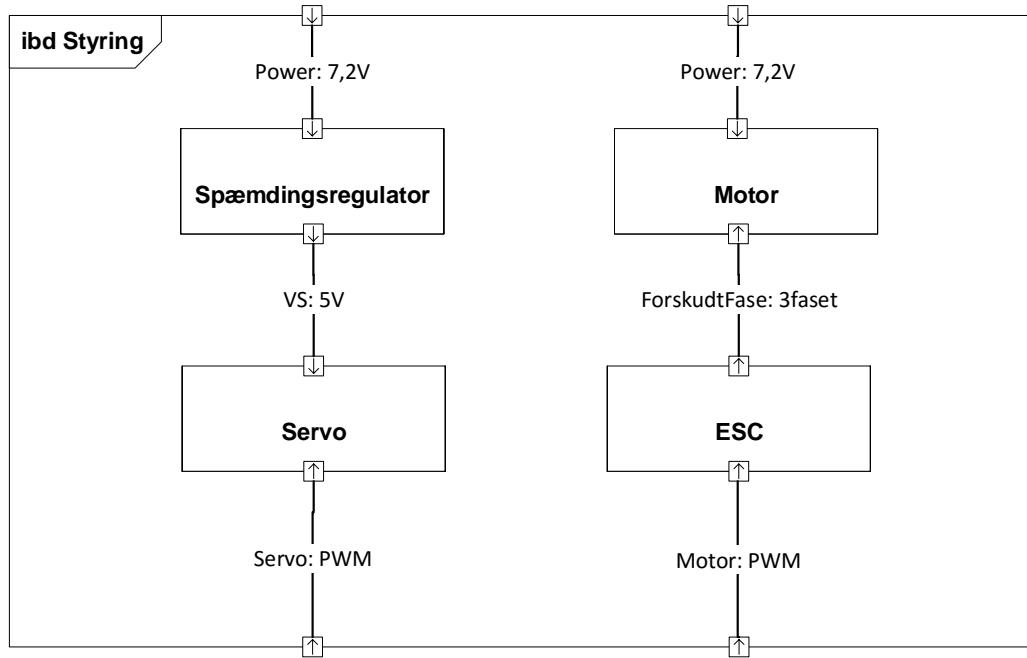
Figur 4.9 viser BDD for Styring. Dette viser hardwaren i forbindelse med Styring. Det er med for at give et bedre overblik over hvad Styring indeholder.



Figur 4.9: Styring BDD

4.5.2 IBD styring

Figur 4.10 viser IBD for Styring. IBD'et giver et overblik over de interne forbindelser mellem de individuelle hardwaredele, hvilket er essentielt i den kommende hardware-designproces.



Figur 4.10: Styring IBD

4.5.2.1 Signalbeskrivelse

Tabel 4.5 viser signalbeskrivelse for Styring.

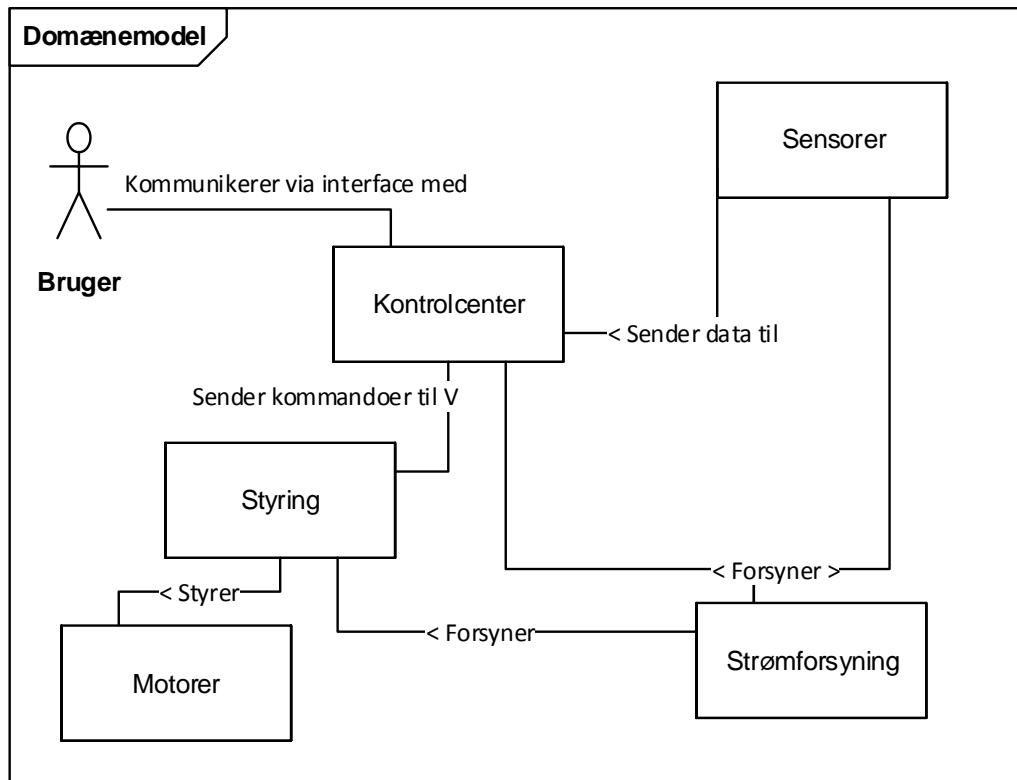
Signal type	Navn	Beskrivelse
7,2	power	7,2V forsyning til motor og servo
PWM	servo	PWM signal til styring af servo
PWM	motor	PWM signal til styring af motor
5V	VSS	5V forsyning til servo
3-faset	forskudtFase	PWM signalet transformert til 3-faset signal til motoren

Tabel 4.5: Signalbeskrivelse for Styring IBD

4.6 Softwarearkitektur

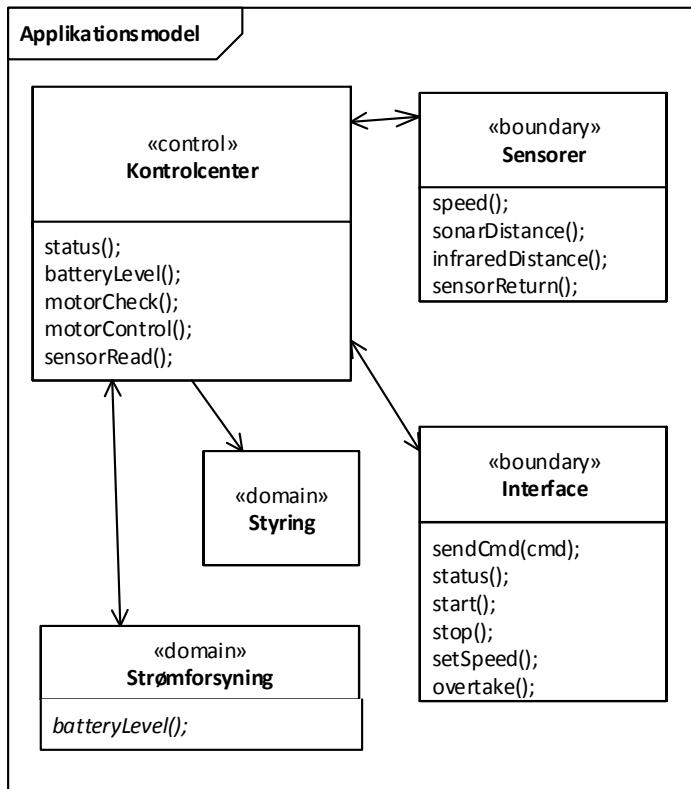
4.6.1 Domæne- og applikationsmodeller

Nedenfor beskrives softwarearkitekturen med en domæne- og applikationsmodel, og en kort gennemgang af hver use case suppleret med et sekvensdiagram.



Figur 4.11: Domænemodel

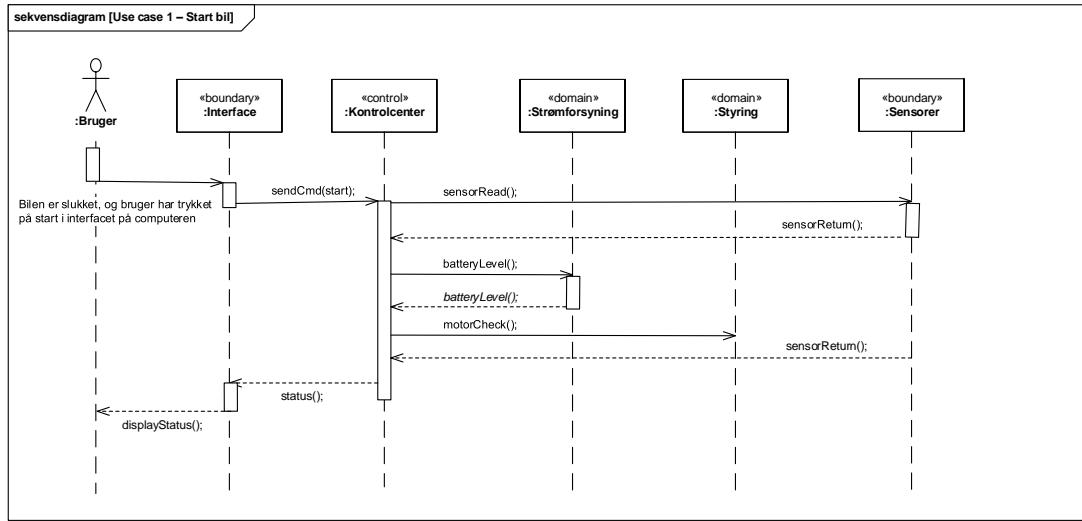
Generelt består systemet af de blokke der defineres i BDD'et, og de kommunikerer sammen som set på domænemodellen ovenfor. Brugeren har adgang til systemet via interfacet, og vejforholdenes ydre påvirkning opsamles af Sensorer. Strømforsyningen er kun inkluderet, da den også har sin egen sensor til at bestemme batteriniveauet. Styringsblokken styrer motorerne; servo og BLDC-motor. Ud fra denne model er det muligt at specificere systemets dele yderligere med en applikationsmodel, som ses på figur 4.12.



Figur 4.12: Applikationsmodel

Applikationsmodellen beskriver entiteternes type og overordnede funktioner. I Software – Design-afsnittet beskrives de konkrete funktioner yderligere, men for arkitekturen er disse generelle beskrivelser af funktionerne tilstrækkeligt. Sensorer- og Interfaceblokke ne er boundary, da det er disse blokke der kommunikerer med bruger og interagerer med omverdenen. I de følgende afsnit beskrives hver af de seks use cases softwaremæssigt. Det er ikke en udtømmende, men en beskrivende behandling der tilsigtes.

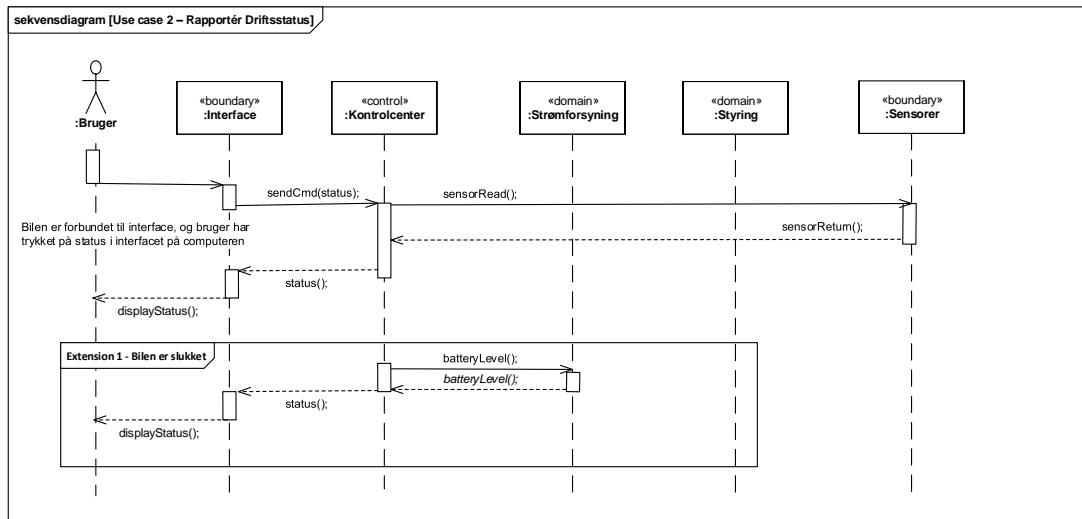
4.6.2 Use case 1 - Start bil



Figur 4.13: Sekvensdiagram - Use case 1 - Start bil

Use case 1 - Start bil, som initieres af brugeren gennem interfacet, har til formål at starte bilen, og checke at sensorinputs og motorer fungerer som de skal, samt checke batterini-veauet. Denne information sendes tilbage til brugeren, og systemet afventer yderligere instruktioner.

4.6.3 Use case 2 - Rapportér driftsstatus

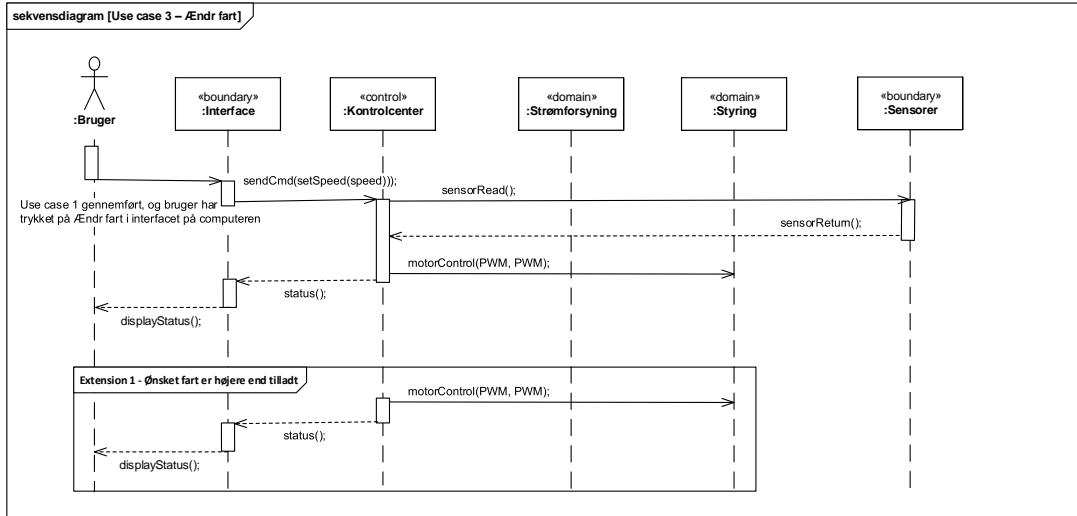


Figur 4.14: Sekvensdiagram - Use case 2 - Rapporter driftsstatus

Use case 2 - Rapporter driftsstatus, som initieres af brugeren gennem interfacet, har til formål at give brugeren en tilbagemelding på sensorernes status. Hvis bilen er slukket

informeres brugeren om batteriniveauet.

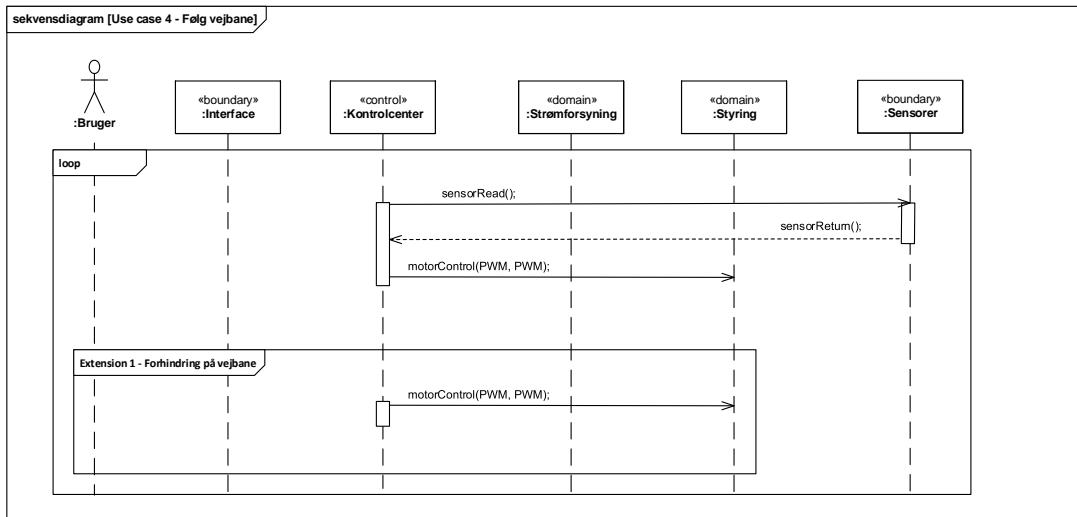
4.6.4 Use case 3 - Ændr fart



Figur 4.15: Sekvensdiagram - Use case 3 - Ændr fart

Use case 3 - Ændr fart, som initieres af brugeren gennem interfacet, har til formål at accelerere bilen til en ny fart. Bilen får kontinuerligt inputs fra Sensorer. Hvis der er et objekt i vejen, eller den angivne hastighed er højere end tilladt, så informeres brugeren herom og bilen fastholder i stedet sin fart.

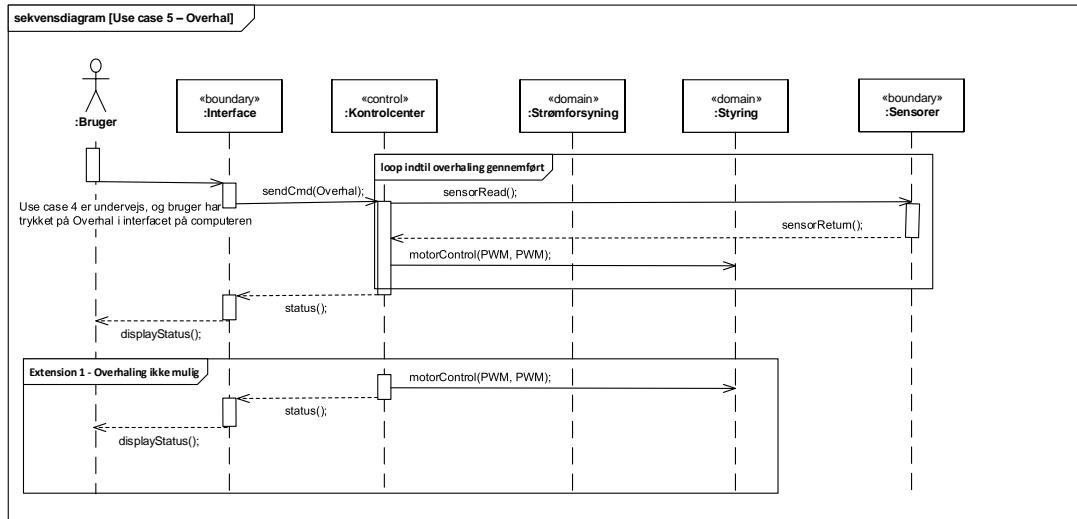
4.6.5 Use case 4 - Følg vejbane



Figur 4.16: Sekvensdiagram - Use case 4 - Følg vejbane

Use case 4 - Følg vejbane, som initieres af vejforholdene, har til formål at holde bilen inde for vejbanestribene. Kommer bilen for tæt på stregerne, justerer Kontrolcenter på Styring, og bilens kurs justeres. Er der en forhindring skal bilen reagere herpå, og kontrolcenteret vil igen sende besked til styringen, denne gang med andre parametre. Denne use case kører kontinuert mens bilen er tændt.

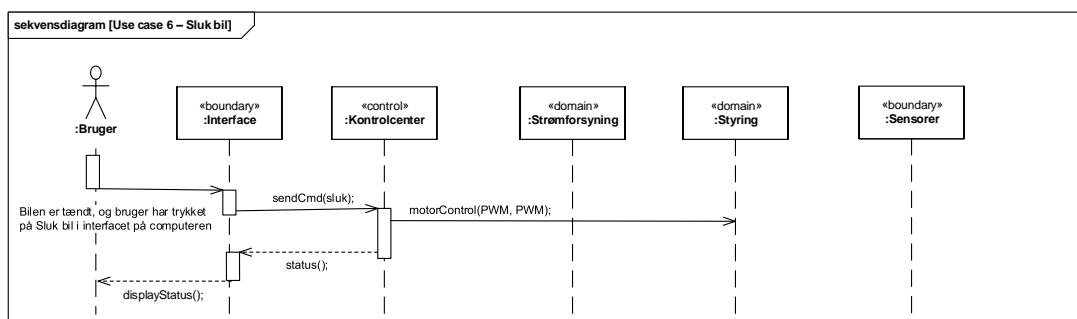
4.6.6 Use case 5 - Overhal



Figur 4.17: Sekvensdiagram - Use case 5 - Overhal

Use case 5 - Overhal, som initieres af brugeren gennem interfacet, har til formål at foretage en overhaling på en forankørende bil. Kontrolcenter vurderer hvor lang tid overhalingen vil tage, baseret på den angivne maksimale fart for bilen, og den forankørende bils fart. Bilen gennemfører overhalingen, hvis det kan gøres inden for en defineret tidsramme. Er det ikke muligt informeres brugeren herom via interfacet.

4.6.7 Use case 6 - Sluk bil



Figur 4.18: Sekvensdiagram - Use case 6 - Sluk bil

Use case 6 - Sluk bil, som initieres af brugeren gennem interfacet, har til formål at slukke bilen. Først slukkes motorerne, og derefter informeres brugeren om at bilen er slukket.

5 Design, implementering og test

5.1 Protokoller

5.1.1 SPI-Protokol

SPI protokollen er lavet for at have overenstemmelse i hvad der bliver sendt imellem Raspberry Pi og PSoC. Den bruges til alt der styres på PSoC'en og alt hvad der skal vides om de forskellige sensorer. Noget af denne data skal sendes videre over TCP fra Raspberry Pi'en. En anden del af dataene bruges direkte på Raspberry Pi'en, til at regulere på fx. hvor servoen skal pege hen, for at holde sig inden for vejbanestriberne.

Tabel 5.1: SPI protokellen

#	Data	Sendt af Raspberry Pi	Data	Sendt af PSoC	Data	Modtaget på Raspberry Pi
0	-26 til 26	Hastighed	-128 til 127	Tachometer	0	Key
1	-100 til 100	Servo placering	1 eller 0	Vejbanesensor, venstre	-128 til 127	Tachometer
2	1 eller 0	Start	1 eller 0	Vejbanesensor, højre	1 eller 0	Vejbanesensor, venstre
3	-128	Key 1	0-127	Afstandssensor	1 eller 0	Vejbanesensor, højre
4	-2	Key 2	0 til 99	Batterisensor motor	0-127	Afstandssensor
5	-128	Key 3	0 til 99	Batterisensor kredsløb	0 til - 99	Batterisensor motor
6	4	Key 4	0	Key	0 til 99	Batterisensor kredsløb

På tabel 5.1 ses selve protokollen. Det er værd at lægge mærke til, at der sendes 4 Keys fra Raspberry Pi. Dette sikrer at det ikke kommer ud af synkroniseringen. Det er vigtigt at fx. hastigheden ikke bliver sat til 100 eller -100, som servoen gerne må sættes til. Al kommunikation foregår i datatypen `int8_t`, som kan indeholde integers værdier imellem -128 og 127. Denne størrelse er nok til der kan sendes alt det nødvendige.

5.1.2 TCP-Protokol

Alle beskeder i TCP-protokollen starter med et tal, efterfulgt af et kommandonavn. Alle beskeder til bilen, med undtagelse af hastighed, indholder ingen yderligere data. Beskeder til pc'en indeholder en statusbesked, som kan ses i de efterfølgende tabeller. De forskellige dele af beskeden er adskilt af et Q. Alle koder slutter med et W. Bindestregerne i tabellen er ikke en del af beskeden, men indsatt for at gøre det mere læsevenligt.

Kommando	handling	modtager	kommentar
01-Q-status-W	Status	Bil	
02-Q-overhal-W	Overhal	Bil	
03-Q-hastighed-Q- "hastighed"-W	Ændre hastighed	Bil	"Hastighed" er en tal-værdi mellem -26 og +26, hvor den første angiver + eller -
04-Q-start-W	Start	Bil	Startkommando, sætter Run til 1
05-Q-stop-W	Stop	Bil	Stopkommando, sætter Run til 0
06-Q-test-W	Test af forbindelse	Bil	Test om der er forbindelse mellem enhed og pc
07-Q-koer -W	Kør start	Bil	
11-Q-status-Q- {statuskode}-W	Status	Pc	Sender status med status koder
12-Q-overhaling- Q-{statuskode}-W	Overhalingsstatus	Pc	Efter gennemført eller ikke gennemført overhaling sendes, sluttet med status på sensorer
13-Q-hastighed-Q- {statuskode} -W	Ændring af hastighed	Pc	Status om hastighed er ændret eller ej.
14-Q-start-Q- {statuskode} -W	Start status	Pc	Status om start er gennemført eller ej, samt status på sensorer.
15-Q-stop-Q- {statuskode}-W	Stop status	Pc	Status om stop er gennemført eller ej
16-Q-test-W	Test af forbindelse	Pc	Bekræftelse af test af forbindelse.
17-Q-koer-Q- {statuskode} -W	Kør status	PC	Status om kørsel er påbegyndt.

Tabel 5.2: TCP-Protokol

5.1.3 TCP Statuskoder

Her under findes alle status koder som bliver sendt i forbindelse med TCP-kommunikation fra Raspberry pi til PC. Statuskoderne bliver kun brugt i forbindelse med kommunikation den en vej, da de eneste ekstra data som beskederne kan indeholde fra pc'en er hastighed. Herunder er alle status koder for overhaling.

Kommando	Hvad	Kommentar
0001	Overhaling gennemført	
0002	Bilen kører for langsomt til at overhale	
0003	Endnu ikke implementeret	

Tabel 5.3: Statuskoder overhaling

Herunder er alle statuskoder, der bliver sendt under opstart.

Kommando	Hvad	Kommentar
11111zzzz	Alle sensorer fungerer som de skal	Zzz er en talværdi fra 0 til 99, som repræsenterer den samlede % effekt tilbage på batteriet.
1yxxxxxxxx	Højre vejbanesensor	En talværdi på 1 hvis den virker, og 0 hvis den ikke gør
1xyxxxxxx	Venstre vejbanesensor	En talværdi på 1 hvis den virker og 0 hvis den ikke gør
1xxxyxxxx	Tachometerstatus	En talværdi på 1 hvis den virker og 0 hvis den ikke gør
1xxxxyxxxx	Afstandssensor	En talværdi på 1 hvis den virker og 0 hvis den ikke gør
1xxxxyyxx	Batterisensorer motor	Tal mellem 0 og 99 som repræsenterer resterende % effekt på batteriet
1xxxxxxyy	Batterisensorer kredsløb	Tal mellem 0 og 99 som repræsentere resterende % effekt på batteriet

Tabel 5.4: start statuskoder

Herunder er alle statusbeskeder, der kan blive sendt i forbindelse med ønsket forøgelse af hastighed.

Kommando	Hvad	Kommentar
2001	Farten er ændret	
2002	Over maksimal tilladt hastighed, hastigheden sættes til den maksimale hastighed (negativ eller positiv)	
2003	farten er ikke ændret	

Tabel 5.5: Skift hastighed statuskoder

Herunder ses statuskoderne for stop status.

Kommando	Hvad	Kommentar
3001	Bilen er stoppet	
3002	Bilen er ikke stoppet	

Tabel 5.6: stop statuskoder

Herunder ses statuskoder på start kørsel.

Kommando	Hvad	Kommentar
4001	Kørsel påbegyndes	
4002	Kørsel påbegyndes ikke	

Tabel 5.7: Start kørsel statusbeskeder

Herunder er statuskoder, når der anmodes om status. Alle hastigheder der modtages er fordoblet, da det er step af 0,5 og der ikke sendes komma.

Kommando	Hvad	Kommentar
5xxxxxxxxxxxxx	Højre vejbanesensor	1 hvis den kan se en strib, 0 hvis den ikke kan.
5xyxxxxxxxxxx	Venstre vejbanesensor	1 hvis den kan se en strib, 0 hvis den ikke kan.
5xyyyyyyyyyyy	Hastighed	Hastigheden udskrives som et tal mellem 0 og 99. Derudover sendes der en prefix, der enten er + eller -, alt efter om hastigheden er postiv eller negativ
5xxxxxyyyyyyy	Afstandssensor	Sender to tal som udgør en afstand mellem 0 og 999 cm.
5xxxxxxxxyyxx	Batterisensor motor	Zz er en talværdi fra 0 til 99, som repræsenterer den samlede % effekt tilbage på batteriet til motorerne.
5xxxxxxxxxxyy	Batterisensorer kredsløb	Zz er en talværdi fra 0 til 99, som repræsenterer den samlede %, effekt tilbage på batteriet til PSoC, Raspberry PI og sensorerne.

Tabel 5.8: Statuskoder for status

5.2 Hardwaredesign, -implementering, og -test

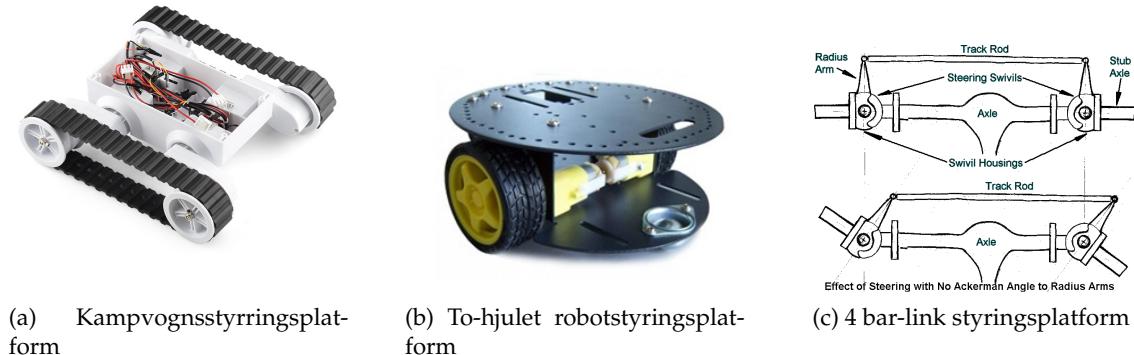
5.2.1 Chassis

I projektet er der valgt selv at designe et chassis til bilen. Dette opstiller en række problemstillinger, som skal løses. Mange af problemstillingerne er hverken software- eller

elektronikorienteret, men mere mekanik eller generelle ingeniørmaessige problemer.

5.2.1.1 Udvælgelse af platform

Der er flere muligheder for den platform, som bilen skal bygges op efter. Der er undersøgt tre typer. En mulighed er et langsomt køretøj som styres af to motorer og larvefødder, en slags kampvognsstyring. En anden mulighed er to hjul, fremfor larvefødder, og coasters foran og bag. Den sidste mulighed er at styre bilen som en traditionel bil, dette kan gøres med en form for 4-bar-link styring. [3], [4], [5]



Figur 5.1: De forskellige former for platforme

Det er valgt at udarbejde bilen, som den tredje løsning. Det er gjort for at bilen skal fungere som en rigtig bil, da det skal være en model for en rigtig bil. Derudover antages det, at dette vil være nemmest at implementere på en full-scale bil.

5.2.1.2 Valg af materialer

Valget af materiale stod imellem flere forskellige muligheder, med forskellige fordele og ulempes:

Træ, har mange fordele, det er nemt at arbejde med, det er holdbart imod dets fibre. Problemet med træ er at det kan være svært at lave små og detaljerede ting.

Plastik, her tænkes på plastik i form af 3D printer materialer som PLA og ABS. Det er kan være nemt at designe, men tager lang tid at fremstille, og det er ikke nødvendigvis holdbart, især som smådele.

Metal, har den fordel at det er meget holdbar i mod brud, dog kan det godt bøje, hvis det bliver for tyndt.

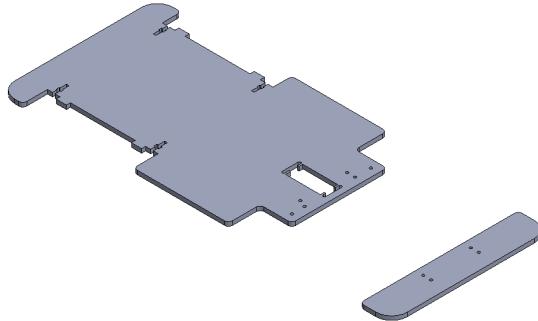
Glasfiber, er noget at det stærkeste der kan bruges, men det er meget komplettest at fremstille og det kræver forskellige kurser.

Akryl, minder meget om metal i fremstillings metoden, men er blødere og så kan det ikke bøje.

Valget faldt på klar akryl med en tykkelse på 5mm. Det kan fremstilles på en CNC-maskine, hvilket der er adgang til på skolen. 5mm blev valgt, da det var den tyndeste tilgængelige plade.

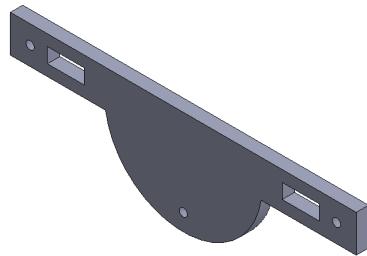
5.2.1.3 Design af chassis

Bilen er udarbejdet til generelt at være $\frac{1}{10}$ størrelse af en bil. Til at dimensionere bilen er der taget udgangspunkt i en Audi A6 Avant [6]. Længden der er valgt, er 40cm og en bredde på 20cm. Ud fra disse mål er der blevet designet en bund til chassiset som kan ses på figur 5.2. Der er på bunden taget højde for at der skal kunne monteres en servo, som skal kunne styre vinklen på forhjulene, for at kunne dreje. Der er også taget højde for at der skal være plads til en motor på det ene baghjul. For at kunne montere hjul er der lavet udsnit så en anden plade kan gå vinkelret i og der er lavet plads til at en møtrik og en skrue kan holde pladen på plads.



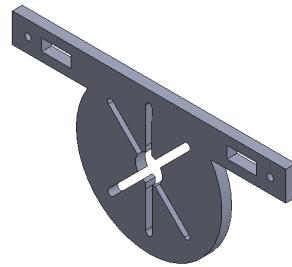
Figur 5.2: Bundstykke af chassiset

Baghjulet hvor der ikke sidder en motor på er monteret på en plade med et hul i denne plade har den rigtige indeksering, så den passer i vinkelret på basen. Den kan ses på figur 5.3



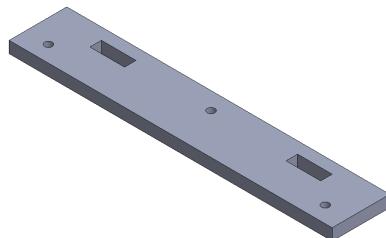
Figur 5.3: Baghjuls monteringsplade

Det andet baghjul er som sagt monteret på motoren. For at sikre den kommer til at passe i de monterings huller der er lavet, er de ikke baseret på placering, men vinkler fra centrum, da det er gjort på samme måde på motoren.

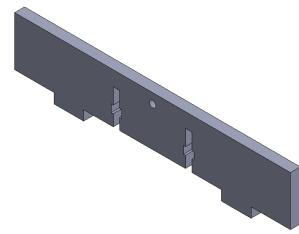


Figur 5.4: Motor monteringsplade

For at forhjulene skal kunne monteres, så de kan bevæge sig, skal de monteres på nogle plader som hænger frit i luften på den ene side. Der er brugt samme princip til at montere disse padder som med baghjulene.



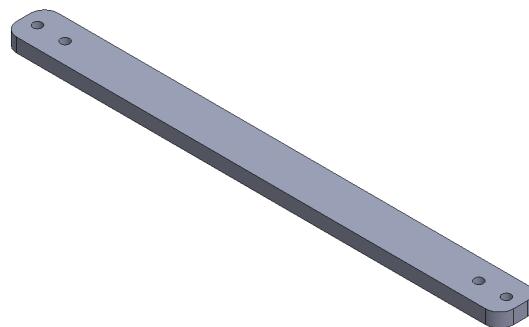
(a) Forhjuls monterings vandrette plade



(b) Forhjuls monterings lodrette plade

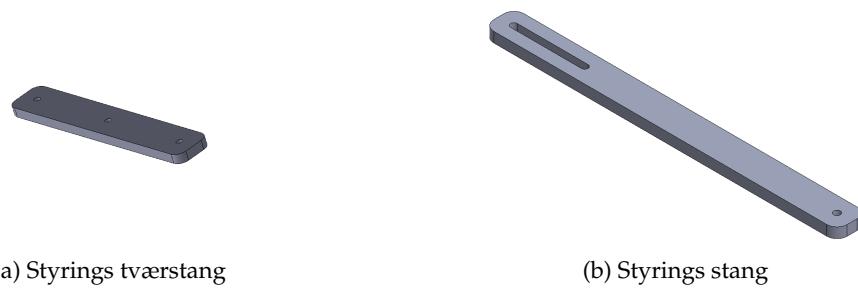
Figur 5.5: Forhjuls montering

For at holde den forreste plade på, er der også to rigide stænger som går fra chassisets base til den forreste plade. Disse er ens og kan ses på figur 5.6



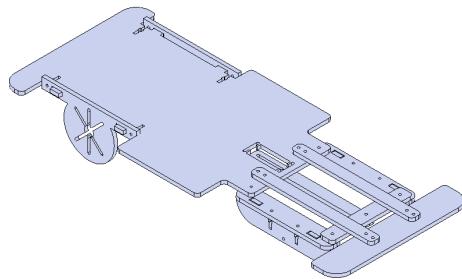
Figur 5.6: Monterings stang til front plade

For at dreje forhjulene skal der være en stang fra servoen ud til forhjulene og derfor er derfor er der forså brug for en tværstang, disse to kan ses på hhv. figur 5.7b og figur 5.7b



Figur 5.7: Forhjulsstyring

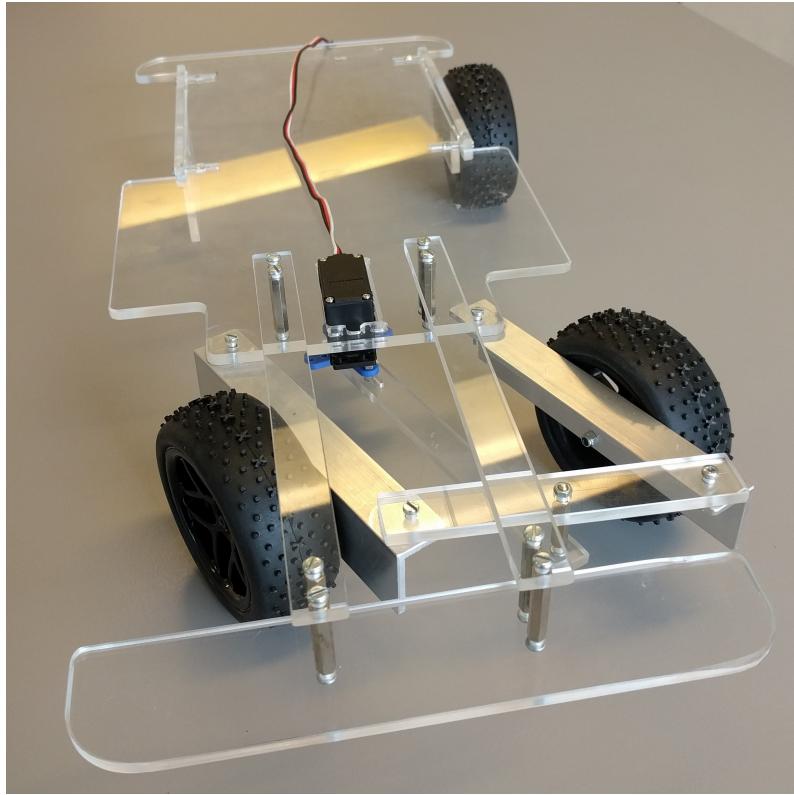
Med disse dele kan hele bilen samles som på figur 5.8. Det ses at der ikke er lavet monteringshuller til de forskellige kredsløb og andet. Det giver mere mening at tilpasse hullerne når de er klar til montering og test. Da der bruges akryl er det nemt at lave nogle huller som stemmer overens med dem der er på kredsløbene.



Figur 5.8: Den samlede bil

5.2.1.4 Det færdige chassis

Efter bilen var blevet fremstillet, er delene blevet samlet til det chassis som ses på figur 5.9. På billedet kan det ses at servo og hjul er monteret. Desuden kan det også ses at forhjuls monteringen er lavet i metal, da det var nemmere under fremstilling, at lave dem af to stykker vinkel metal. Alle ting er monteret med M3 maskinskruer og møtrikker samt PCB stand-offs.



Figur 5.9: Chassiset samlet

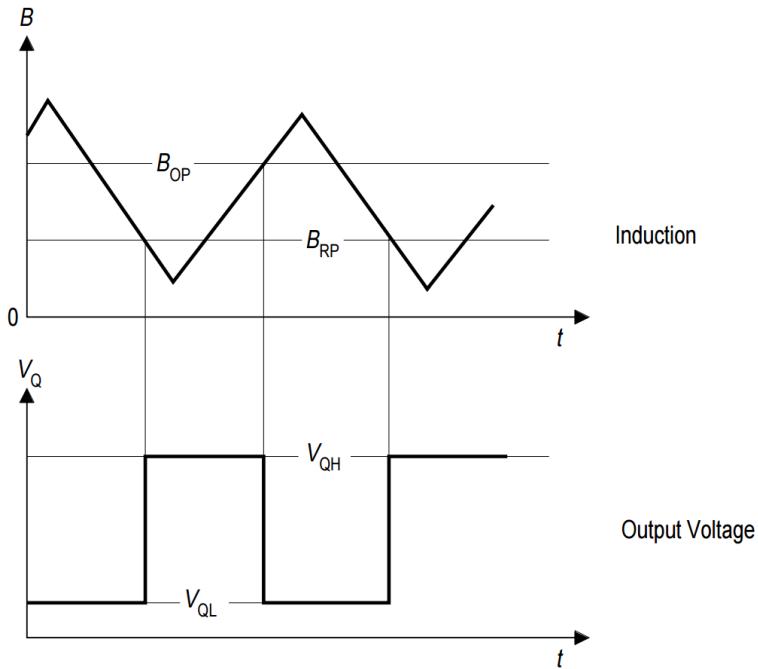
5.2.2 Hall-effekt sensor

En Hall effekt-sensor er en transducer, der ændrer outputsignalet alt efter tilstedeværelsen af et magnetisk felt. Sensoren har, ligesom en magnet, to polariteter, så den både kan detektere nord- og sydpoler. Sensorerne er placeret således, de detekterer nordpol. Dette er dog irrelevant, da det blot forskyder perioden af outputtet en halv periode. Output signalet er digitalt, hvor høj svarer til forsyningsspændingen og lav er stel. Dette bruges til at generere et firkant-signal, hvor perioden ændres af motorens omdrejningshastighed.

På figur 5.10 vises principippet i Hall-effekt sensoren. Hvis induktionen fra magneten bliver større end B_{OP} går outputtet lavt, og bliver den lavere end B_{RP} går outputtet højt. Dvs. at sensoren er aktiv høj, og der skal bruges en pull-up modstand fra signal benet til forsyningsspændingen.

Der er to overordnede Hall-effekt sensorer - bipolar og unipolar. Ved en bipolær sensor bruges der en latch til at holde outputtet lavt, indtil et magnetfelt med modsat polaritet detekteres. Mens outputtet ved en unipolær sensor går højt, med det samme magneten er uden for rækkevidde. I dette projekt bruges en TLE4905L, som er en unipolær Hall-effekt sensor. Der kunne dog bruges både en bi- og unipolær sensor da magneterne i motoren skiftevis er nord- og sydpol.

Til dette projekt bruges tre stk. Hall effekt-sensor TLE4905[7]. Den er valgt fordi den har et digitalt output, der er beskyttet mod elektriske forstyrrelser.



Figur 5.10: Princip af hall effect sensor [7]

Til beregning af farten bruges to formler; først beregnes omdrejningstallet, RPM, og ud fra det beregnes farten. RPM udregnes ved, at bruge følgende formel, hvor 't' er tiden mellem hver puls fra sensoren. Dvs. periodens frekvens ganget med 60, for at få det i minutter, og divideret med 7, da den får 7 pulser per omgang.

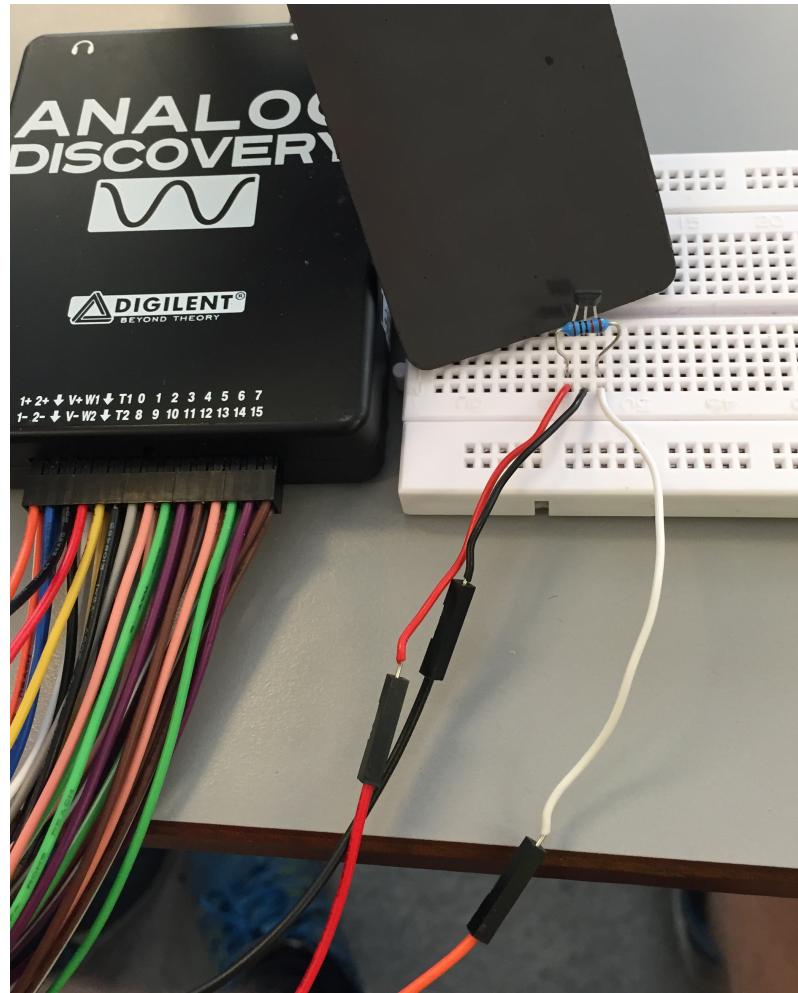
$$\text{RPM} = \frac{\frac{1}{t} \cdot 60}{7}$$

Mens farten udregnes ved nedenstående formel. 'R' er motorens radius i meter, som er 0,043 m.

$$\text{fart} = \frac{2 \cdot \pi \cdot R \cdot \text{RPM}}{1000} \cdot 60$$

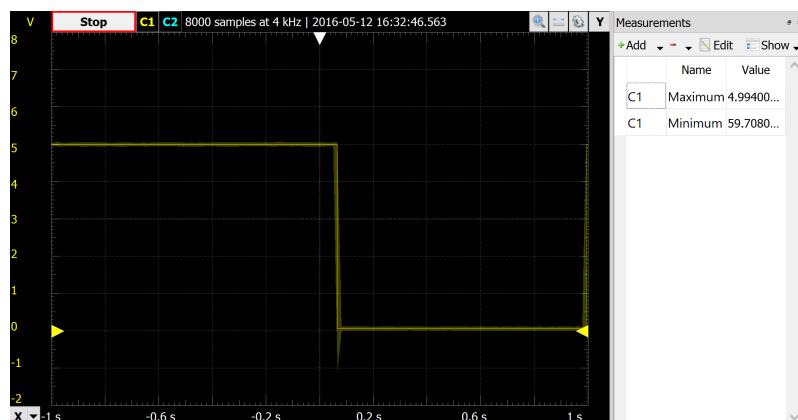
5.2.2.1 Modultest

Tachometeret testes ved at bruge opstillingen på figur 5.11. Her er den røde ledning VCC på 5V, den sorte er GND, og den hvide er det digitale output-signal. Pull-up modstanden forbindes mellem output-benet og VCC. Den sorte blok er magneten, der bruges til at aktivere hall-effect sensoren.



Figur 5.11: Test opstilling hall-effect sensor

På figur 5.12 ses resultatet for testen af Hall-effekt sensoren. Ved tilstedeværelsen af magneten ses det tydelige skifte mellem deaktivert og aktiveret. Derudover ses det på maximum- og minimumsværdierne, at signalet er klart inden for CMOS-standarten for digitalt 1(3,5V) og 0(1,5V), som bruges i PSOC'en.



Figur 5.12: Test af Hall-effect sensor

5.2.3 Sonar

5.2.3.1 Frekvensvurdering

Til afstandsbestemmelse for systemet er der benyttet en Sonar, nærmere præcist LV-MaxSonar® -EZ™ MB1010[8]. Et Lidar- eller Radar-system er det der almindeligvis bruges af selvkørende biler lavet af firmaer som Google (Lidar) og Tesla (Radar)[9], men budget-begrænsninger betød at valget faldt på en sonar. En sonar anvender lyd til at ramme objekter, og måler tiden det tager før det reflekterede signal måles. Lydens hastighed i luft kan bestemmes med formlen

$$v_{lyd} = \sqrt{\frac{\gamma RT}{M}}$$

, hvor γ er materialets adiabatiske konstant, R er Rydberg's konstant, T er temperaturen, og M er materialets gennemsnitlige molekulærmasse. $\gamma = 1.4$ for luft ved jordoverfladen, $R = 8.314 \frac{J}{mol \cdot K}$, $T = 293.149K$ (20 Celsius), og molar-massen for luft kan findes ud fra følgende sum:

$$\frac{1}{\bar{M}} = \sum_i \frac{w_i}{M_i}$$

, hvor M_i er komponentens molarmasse, w_i er dens masse-fraktion, og \bar{M} er den gennemsnitlige molarmasse af gassens komponenter. Denne bliver altså [10] [11]

$$\begin{aligned} \frac{1}{\bar{M}} &\approx \frac{w_{\text{Nitrogen}}}{M_{\text{Nitrogen}}} + \frac{w_{\text{Oxygen}}}{M_{\text{Oxygen}}} + \frac{w_{\text{Argon}}}{M_{\text{Argon}}} + \frac{w_{\text{CO}_2}}{M_{\text{CO}_2}} + \frac{w_{\text{Neon}}}{M_{\text{Neon}}} \\ &= \frac{78.08}{28.0134 \frac{g}{mol}} + \frac{20.95\%}{31.9988 \frac{g}{mol}} + \frac{0.934\%}{39.948 \frac{g}{mol}} + \frac{0.03978\%}{44.01 \frac{g}{mol}} + \frac{0.001818\%}{20.179 \frac{g}{mol}} \\ &= 0.03466 \frac{mol}{g} = 28.85 \frac{g}{mol} \end{aligned} \quad (5.1)$$

, når de fem mest almindelige gasser i atmosfæren, i alt 99.997%, inkluderes. Dermed fås

$$v_{lyd} = \sqrt{\frac{\gamma RT}{M}} \approx 343.9 \frac{m}{s}$$

Sonaren kan ifølge databladet detekttere objekter op til 6.45m, hvilket med denne fart tager

$$\Delta t = \frac{6.45m \cdot 2}{343.9 \frac{m}{s}} = 37.5ms$$

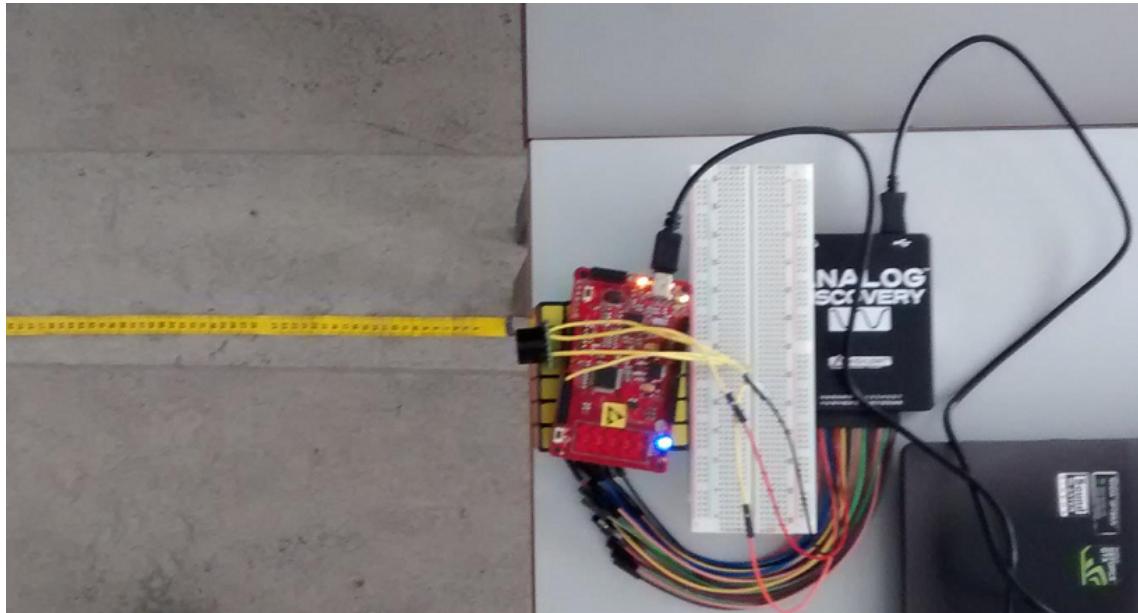
Databladet angiver en opdateringsfrekvens på 20Hz dvs. 20ms mellem hver måling. Dette virker meget rimeligt, og 20Hz er nok til at kunne reagere på objekter i bilens bane. Desuden nævnes det i databladet, at sonaren helst skal være 40 cm fra alle objekter når den tændes. Det skyldes, at den første read-cycle bruges til, at kalibrere sonaren internt. Hvis der placerer noget for tæt på, vil sonaren give større fejl.

5.2.3.2 Modultest

Sonarens analoge output er benyttet. Outputtet giver en spænding svarende til $\frac{V_{cc}}{512}$ per tomme. Med en forsyning på 5V bør dette være

$$\frac{5V}{512 \cdot 2.54cm} \approx 3.86 \frac{mV}{cm}$$

Til at teste det tilsluttes sonaren PSoC'en og et Oscilloskop. Et målebånd blev rullet ud foran den, som vist på billedet nedenfor.



Figur 5.13: Testopstilling for sonar

Tabellen nedenfor viser testresultaterne af sonaren

Afstand (cm)	Oscilloskop (mV)	ADC (mV)	Oscilloskop (cm)	ADC (cm)	Justeret ADC (cm)
20	69	84.92	17.88	22.0	19.86
30	107	124.68	27.72	32.3	30.36
40	147	149.00	38.08	38.6	36.78
50	176	182.96	45.60	47.4	45.75
70	247	248.97	63.99	64.5	63.18
90	318	325.01	82.38	84.2	83.26
120	421	454.32	109.07	117.1	117.41
140	490	514.92	126.94	133.4	133.41
160	561	602.93	145.34	156.2	156.65
180	630	675.89	163.21	175.1	175.92
200	697	773.93	180.57	200.0	201.80

Den første søjle er de målte afstande med tommestok, og de to næste er resultaterne fra hhv. Digilent-oscilloskop og PSoC'ens SAR ADC. De to efterfølgende kolonner angiver den omregnede afstand fra disse to måleapparater, ved at benytte de $3.86 \frac{\text{mV}}{\text{cm}}$ fra før. Det ses, at ADC'ens output ikke stemmer overens med den målte afstand, og heller ikke oscilloskopets resultat er helt præcist. Ud fra regressionsanalyse kan det findes, at den bedste sammenhæng mellem det analoge output og afstanden faktisk er

$$V_{\text{Oscilloskop}} = 3.61 \frac{\text{mV}}{\text{cm}} \cdot d - 1.6\text{mV}$$

Der er et offset på 1.6mV Derudover er der en anden a-værdi end angivet, hvilket nok skyldes fælles ground med PSoC. Disse afvigelser opstår ikke ellers. Den mere interessante sammenhæng er mellem ADC'ens output og den faktiske afstand. Det er denne

sammenhæng der betyder noget for koden. Ingen anvendes lineær matrix-regression, og der fås

$$V_{ADC} = 3.787 \frac{mV}{cm} \cdot d + -2.561 mV$$

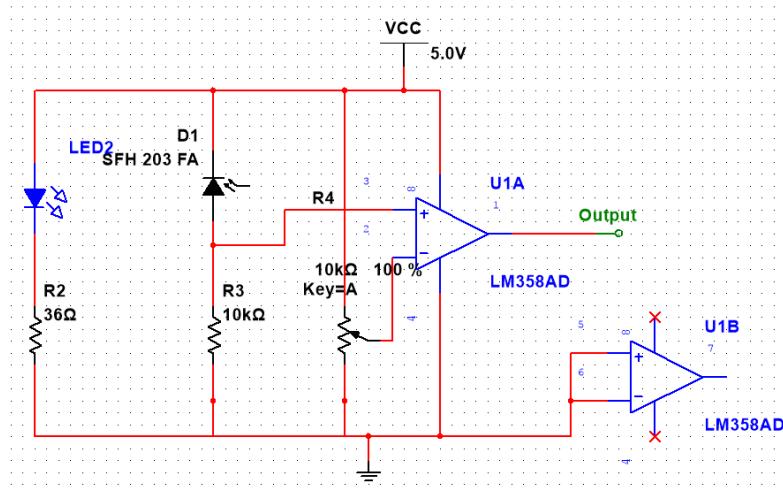
For at finde afstanden i koden skal følgende justering indsættes:

$$V_{ADC} = 3.93 \frac{mV}{cm} \cdot d + 15.8 mV \implies d \cong \frac{V_{ADC}}{3.787 \frac{mV}{cm}} - 2.561 mV$$

Den sidste kolonne i tabellen viser ADC'ens output efter denne justering i koden, og det ses at den passer meget præcist.

5.2.4 Vejbanesensor

Til at detektere vejbanestribene er der benyttet en IR-sensor. Formålet med denne er, at dens signal vil skifte, når den møder en vejstribe. Det er gjort ved, at bruge en IR LED samt en fotodiode. IR LED'en fungerer ved, at den udsender et infrarødt lys, som bliver reflekteret hvis det rammer noget lyst, og absorberet hvis det rammer noget mørkt. Her er valgt en SFH485, fordi den har en optik der koncentrerer strålen.[12] Hvis LED'en og fotodioden bliver placeret tæt op af hinanden, vil lyset fra IR LED'en blive reflekteret direkte mod fotodioden. På denne måde kan fotodioden altså detektere lyset hvis bilen kører over et reflekterende underlag. Som fotodiode er brugt en SFH203FA, da den skærmer for sollys.[13] På figur 5.14 ses vores designopstilling.



Figur 5.14: Vejbanesensor design

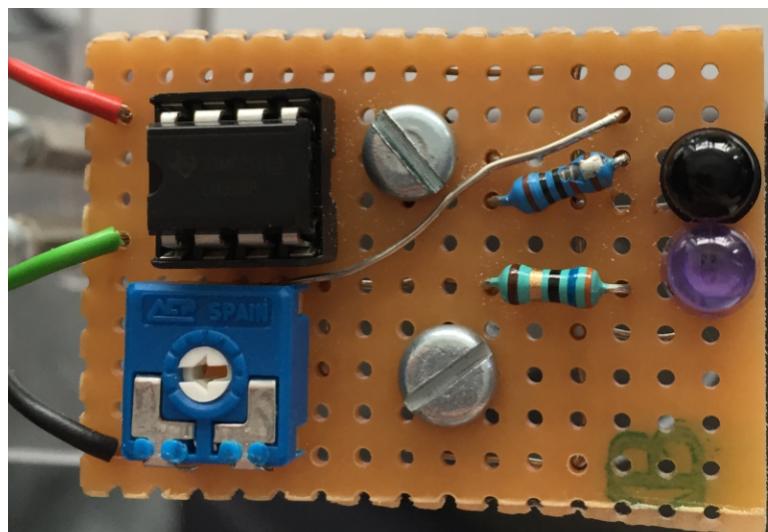
Da PSoC'en skal bruge et højt signal ved en vejstribe og ellers et lavt signal, er signalet fra sensorens output lavet som et digitalt signal. Dette er gjort ved at indsætte OPAMP'en LM358M.[14] Forstærkerdelen i OPAMP'en bruges dog ikke. Det eneste OPAMP'en bruges til her, er at sammenligne to spændinger. Hvoraf den ene fastsættes af os, og den anden varierer alt efter underlaget. Så hvis spændingen fra kredsløbet bliver mindre end

den fastsatte spænding, så vil der komme ét output og et andet output hvis spændingen er over den fastsatte.[15]

Når fotodioden ikke detekterer noget lys, fungerer den som en meget høj modstand. Når den detekterer lys, vil den have en lavere modstand og strømmen igennem den vil øges. Spændingen herfra bliver sendt ind på det ene ben på OPAMP'en, og det er den spænding der sammenlignes med vores fastsatte. Modstanden på de 36Ω er fastlagt ved, at se på databladsværdien "forward current" for IR LED'en, som i vores tilfælde er på 100mA. Det er den maksimale strøm LED'en kan tåle. Der er valgt en modstand der sørger for, at komme tæt på de 100mA, men samtidig sikre, at den ikke kommer over. Spændingsfaldet over IR LED'en er omkring 1.9V, så der vil være 3.1V over modstanden. Strømmen igennem IR LED med de 36Ω er derfor:

$$\frac{3.1V}{36\Omega} = 86.1mA$$

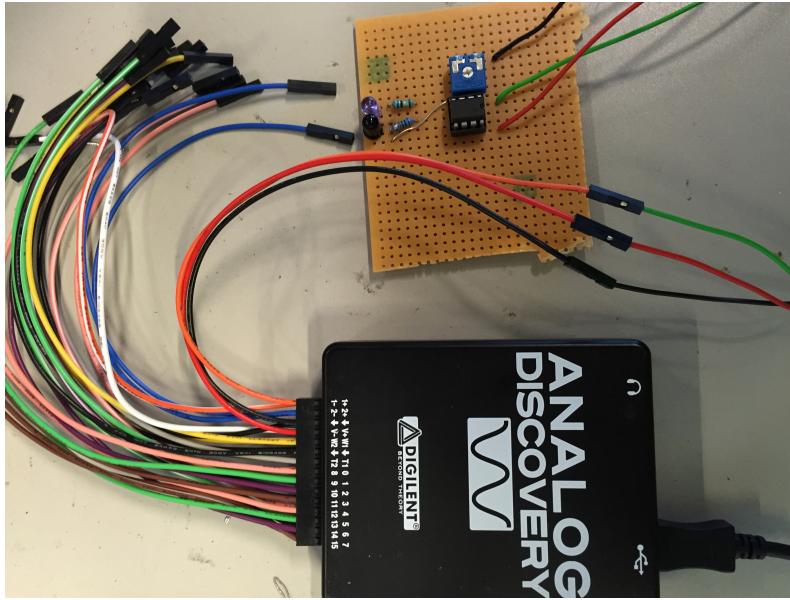
Sammenligningsværdien sættes med et $10k\Omega$ potentiometer. Potentiometret gør, at sensoren kan kalibreres, og derfor indstilles til at virke på forskellige underlag. Den færdige sensor ses på figur 5.15



Figur 5.15: Implementering af vejbanesensor

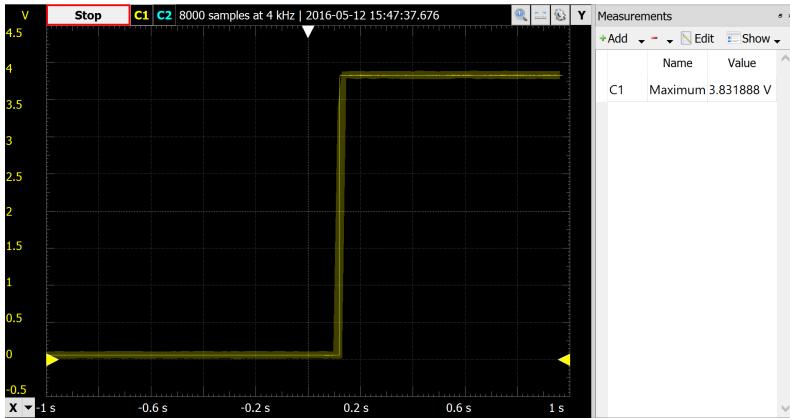
5.2.4.1 Modultest

Til at teste vejbanesensoren bruges analog discovery og en refleksbrik. Printet forsynes fra analog med 5V (rød ledning) og GND (sort ledning). Oscilloskopet (grøn ledning) sættes på sensorens output. Testopstillingen ses på figur 5.16



Figur 5.16: Testopstilling vejbanesensor

På figur 5.17 ses oscilloskop billedet af testen. Det ses, at uden refleksbrikken er outputtet 0V. Når refleksbrikken sættes over sensoren stiger spændingen til 3.8V. CMOS standarden for et højt signal er 3,5V, hvilket vil sige, at PSoC'en vil betragte de 3,8V som et højt signal og selvfølgelig de 0V som et lavt.



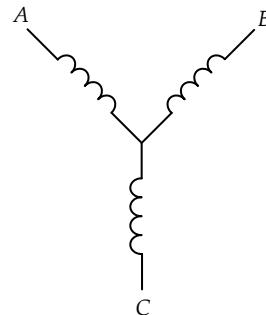
Figur 5.17: Test af vejbanesensor

5.2.5 ESC

Motorstyringen udviklet er en såkaldt ESC - Electronic Speed Control[16]. Modulet skal levere den nødvendige effekt til motoren samt regulere motorens hastighed til et ønsket niveau. Modulet består af en reguleringsløkke samt et effekttrin. Kredsløbet beskrevet i dette er kun effekttrinnet. Reguleringsløkken er implementeret i software på PSoC og effekttrinnet er implementeret i hardware med diskrete komponenter.

Der anvendes en brushless motor som opererer ud fra 3-faser. Figur 5.18 viser principippet for hvordan en brushless motor er opbygget. Motorens 3 faser er forbundet i en

stjerneforbindelse. Motorens fysiske spoler er monteret radialt og er omgivet af permanente magneter i den ydre ring.



Figur 5.18: Brushless motor princip. Figur tilpasset fra [17, FIGURE 8]



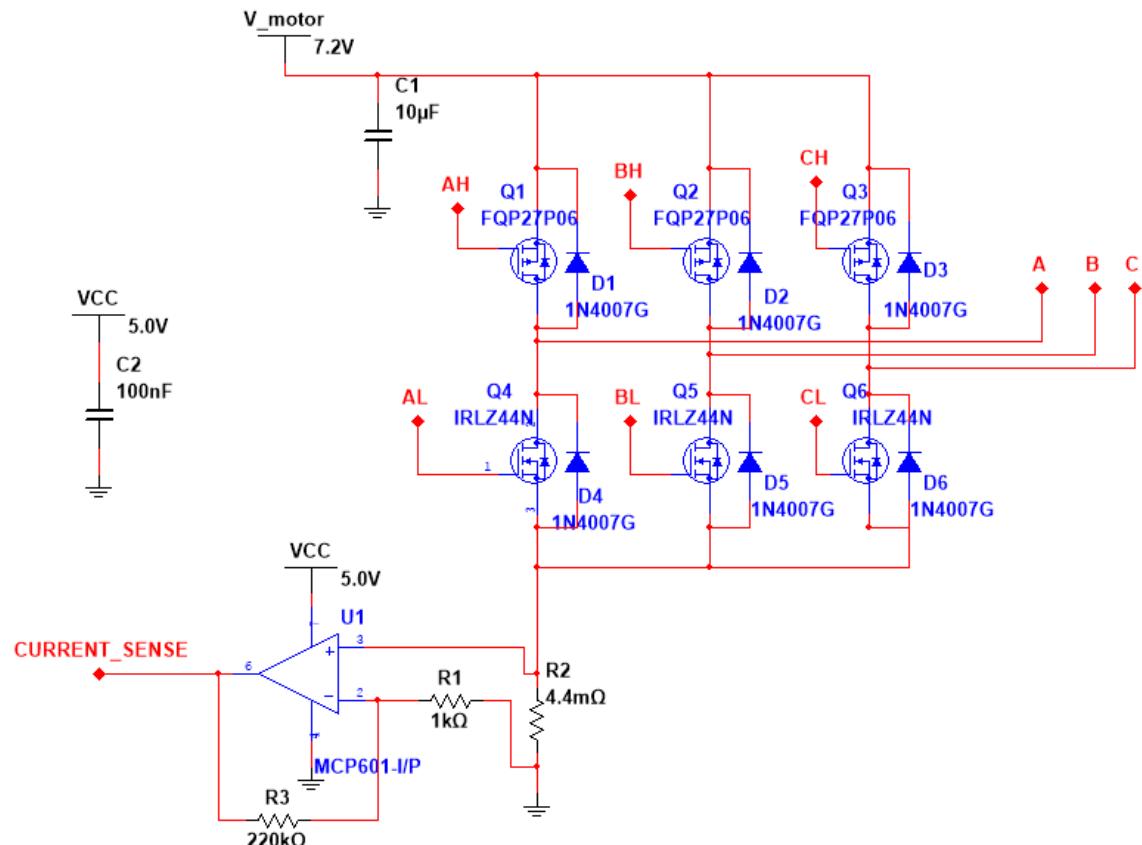
Figur 5.19: Motor fra hobbyking.com [18]

Figur 5.19 viser motoren der anvendes. Motoren der anvendes er en Quanum MT Series 5208 Brushless Multirotor Motor[18] indkøbt fra Hobby-King. Motoren anvendes typisk til droner, men kan anvendes til bilen pga. det lave omdrejningstal og den lave pris. Motoren har 12 spoler fordelt over 3 faser og udgør stator. 14 permanente magneter (number of poles) er placeret i en ring rundt om spolerne og udgør rotor, altså den bevægende dels. Effektkredsløbet sammen med reguleringen skal udgøre kommutatoren, som skaber den vekslende strøm krævet for at drive motoren.

5.2.5.1 Effekt-trin

For at kunne skifte polariteten over motorens spoler er der udviklet et effekttrin som kan leverer den nødvendige strøm til motoren - en såkaldt 3-faset inverter.

I udgangstrinnet anvendes N-kanal og P-kanal MOSFET i en push-pull konfiguration. Disse skal dimensioneres således de kan klare strømmen der leveres til motoren. I et worst-case scenario løber der maksimal strøm igennem transistoren, men i størstedelen af tiden vil udgangstrinnet til hver af de 3 faser skiftes til at lede. Worst-case sker hvis motoren tvinges til at holde stille i en position – eksempelvis hvis den er kørt fast eller prøver at køre op ad bakke men er for hårdt belastet.



Figur 5.20: Effektkredsløb til motoren. Gate driver kredsløb ikke vist.

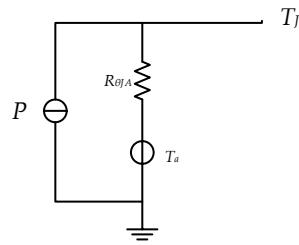
Figur 5.20 viser effektkredsløbet – gate driver kredsløb er ikke vist. Q1, Q2, Q3 udgør den høje side og Q4, Q5 og Q6 udgør den lave side af effektkredsløbet. A, B og C er forbindelsen til motorens spoler, AH, BH og CH er styresignaler til den høje side og AL, BL og CL er styresignaler til den lave side. Styresignalerne kommer fra gate driver kredsløbet som ikke er vist på figuren.

Der anvendes IRLZ44N N-kanal MOSFET til drivere på den lave side og FQP27P06 P-kanal MOSFET som drivere på den høje side. Ifølge databladet kan IRLZ44N holde til en strøm på 47 A kontinuert igennem drain og source, og FQP27P06 kan holde til 27 A¹. Strømmen der kan løbe igennem transistorerne er dog begrænset af effekten som kan afsættes i transistorerne. Effekten transistorerne kan holde til er begrænset af transi-

¹Dette er specificeret ved en junction temperatur på 25 °C.

storhusets termiske egenskaber. Begge MOSFET kommer i TO-220 transistorhuse og har ifølge databladet en termisk modstand fra selve den integrerede chip til omgivelsestemperaturen, junction-to-ambient, på $R_{\theta JA} = 62 \text{ }^{\circ}\text{C W}^{-1}$. Det vil sige at transistorerne vil hæve sig $1 \text{ }^{\circ}\text{C}$ over omgivelsestemperaturen for hver watt der afsættes.

Det termiske kredsløb kan ækvivaleres med et elektrisk kredsløb[19]: Effekten afsat er modeleret som en strømkilde, de enkelte termiske modstande er modeleret som modstande og omgivelsestemperaturen er modeleret som en spændingskilde. Figur 5.21 viser det termiske kredsløb for transistoren uden monteret køleplade.



Figur 5.21: Termisk kredsløb for MOSFET uden køleplade.

Effekten afsat i en MOSFET afhænger af strømmen der løber mellem drain og source samt dens modstand $R_{DS(on)}$. On-modstanden $R_{DS(on)}$ afhænger af spændingen over gate og source V_{GS} . N-kanalerne på den lave side vil kunne drives med mindst 5 V og P-kanalerne på den høje side vil kunne drives med op til forsyningsspændingen over source og drains, $V_{GS} = -7,2 \text{ V}$. Dette kan findes under databladet for MOSFET transistorerne under karakteristikken for I_D vs. V_{DS} .

For IRLZ44N med $V_{GS} = 5 \text{ V}$ er modstanden angivet til $R_{DS(on)} = 25 \text{ m}\Omega$. FQP27P06's on modstand er opgivet til $R_{DS(on)} = 70 \text{ m}\Omega$. P-kanalerne har den højeste modstand og sætter begrænsningen for strømmen idet strømmen skal løber igennem både en P-kanal og en N-kanal på samme tid, altså vil der afsættes mest effekt i P-kanalerne og derved begrænser strømmen.

$$P = I^2 \cdot R_{DS(on)} \quad (5.2)$$

$$T_J = P \cdot R_{\theta JA} + T_A \quad (5.3)$$

Effekten afsat i transistoren kan simpelt beregnes ved joules lov(5.2). Junction temperaturen kan findes ved (5.3), hvor T_J er junction temperaturen og T_A er omgivelsestemperaturen. Ligningerne kan omskrives til (5.4) for at udregne maksimalstrømmen givet ud fra de termiske begrænsninger.

$$I_{max} = \sqrt{\frac{T_J - T_A}{R_{\theta JA}} \cdot R_{DS(on)}} \quad (5.4)$$

Transistorerne selv er opgivet til en maksimal tilladt junction temperatur på $175 \text{ }^{\circ}\text{C}$ ifølge databladet, men dette er langt over hvad der kan anses som rimeligt. Maksimal junction temperatur dimensioneres til $100 \text{ }^{\circ}\text{C}$, altså vil der være en temperaturstigning på $75 \text{ }^{\circ}\text{C}$ over omgivelsestemperaturen². Den maksimale strøm vil således kunne udreg-

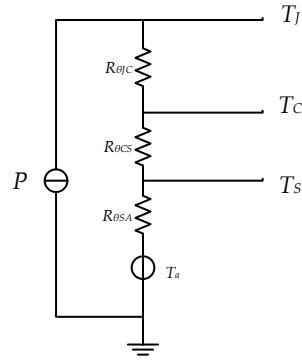
²Omgivelsestemperaturen antages at være $25 \text{ }^{\circ}\text{C}$.

nes:

$$I_{\max} = \sqrt{100 \text{ } ^\circ\text{C} - 25 \text{ } ^\circ\text{C}} \cdot 62 \text{ } ^\circ\text{C W}^{-1} \cdot 70 \cdot 10^{-3} \Omega = 4,2 \text{ A} \quad (5.5)$$

P-kanalerne vil altså kunne leve ca. 4 A og stadig være indenfor sikre termiske rammer. For at forøge dette kan der enten vælges en MOSFET med bedre karakteristikker (lavere $R_{DS(on)}$) eller transistoren kan køles med køleplade og derved sænke den termiske modstand til omgivelserne.

Anvendes clip-on kølefinnen fra Cypax, som er opgivet til en termisk modstand $R_{\theta SA} = 21 \text{ } ^\circ\text{C W}^{-1}$ [20] vil de termiske rammer kunne udvides. Der vil være en termisk modstand fra junction til transistorhuset, og fra transistorhuset til kølepladen samt fra kølepladen til omgivelserne. Figur 5.22 skitserer det termiske kredsløb med kølepladen monteret.



Figur 5.22: Termisk kredsløb med påmonteret køleplade.

Det antages at der er god termisk kontakt mellem transistorhuset og kølepladen ($R_{\theta CS}$). For at indregne en sikkerhedsmargin sættes den termiske modstand til at være $R_{\theta CS} = 2 \text{ } ^\circ\text{C W}^{-1}$. Den termiske modstand fra junction-til-case er opgivet fra databladet til $1,25 \text{ } ^\circ\text{C W}^{-1}$. Den samlede termiske modstand fra junction-til-ambient m. kølefinne estimeres således til $R_{\theta tot} \approx 24 \text{ } ^\circ\text{C W}^{-1}$

$$I_{\max} = \sqrt{\frac{100 \text{ } ^\circ\text{C} - 25 \text{ } ^\circ\text{C}}{24 \text{ } ^\circ\text{C W}^{-1} \cdot 70 \cdot 10^{-3} \Omega}} = 6,7 \text{ A} \quad (5.6)$$

Anvendes kølepladen vil der kunne leveres ca. 7 A og stadig være indenfor et sikkert termisk område. Kredsløbet er dimensioneret til at leve 5 A for at være give en således der er en sikkerhedsmargin. Ved en strøm på $I = 5 \text{ A}$ vil junction hæve sig $(5 \text{ A})^2 \cdot 70 \cdot 10^{-3} \Omega \cdot 22,25 \text{ } ^\circ\text{C W}^{-1} = 39 \text{ } ^\circ\text{C}$. Selve transistorhuset vil hæve sig $(5 \text{ A})^2 \cdot 70 \cdot 10^{-3} \Omega \cdot 21 \text{ } ^\circ\text{C W}^{-1} = 37 \text{ } ^\circ\text{C}$. Heatsinken vil hæve sig ca. samme med samme temperatur som transistorhuset pga. den lave termiske modstand fra junction til case ($R_{\theta JC} = 1,25 \text{ } ^\circ\text{C W}^{-1}$).

5.2.5.2 Drive på MOSFET gate

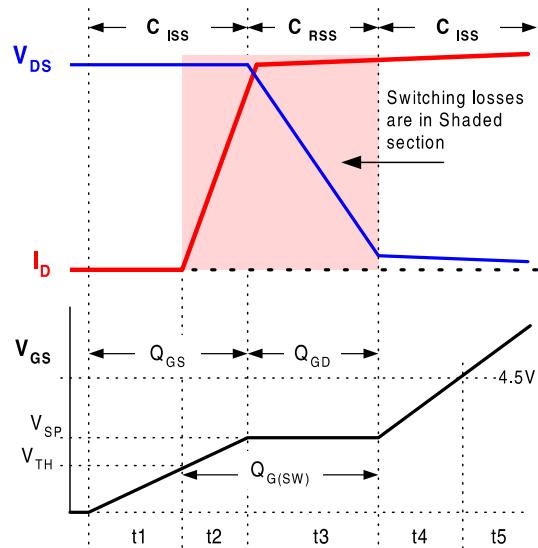
MOSFET gaten fremstår som en kondensator for den del af kredsløbet som tænder for transistoren. Ifølge databladet for IRLZ44N og FQP27P06 Hvis gate kondensatoren er afladt vil den ses som en kortslutning til stel for kredsløbet der skal tænde for transistoren. MOSFET switch operationen bliver mere effektivt, altså afsættes der mindre

effekt, hvis rise-time og fall-time minimeres[21]. For at minimere rise-time og fall-time skal kondensatoren på gaten oplades og aflades så hurtigt som muligt. Drive kredsløbet skal således kunne leve og modtage en stor strøm kortvarigt. Der er vigtigt at der er en lille impedans mellem gaten og drive kredsløbet.

Et alternativ vil være at anvende en integreret løsning til at drive gaten på transistorerne. Nogle løsninger indeholder indbyggede ladningspumper eller andre mekanismer der kan hæve spændingen på gate. Dette gør det muligt at anvende N-kanaler på den høje side af effekttrinnet. N-kanaler har typisk lavere $R_{DS(on)}$ end P-kanaler og er derfor mere eftertragtet. Der er valgt at bruge diskrete komponenter i denne sammenhæng for at undersøge gate driver funktionaliteten.

5.2.5.3 Effektivitet

På inverterens effekttrin vil der være tab i transistorerne. Der vil være rent ledningsstab i transistorerne som er proportionel med strømmen der leveres til motoren. Der vil opstå et tab idet transistoren skal skifte fra at være slukket (cut-off) til at være tændt (saturation). Dette skift sker ikke øjeblikkeligt på grund af forskellige kapacitanser i MOSFET opbygningen[22, Afsnit: *Dynamic Characteristics*].



Figur 5.23: Waveform for MOSFET switch. Figur taget fra [21, Figure 3. High-Side Switching losses and Q_G].

Figur 5.23 viser waveform for MOSFET når den tænder. Tabet er i området hvor strømmen I_D stiger imens V_{GS} falder (markeret på figuren). Når spændingen og strømmen er settet og ikke har en hældning mere, så er tabet rent resistiv i transistoren. Der opstår et plateau i opladningen af gate kapaciteten kendt som Miller effekten[22, Afsnit: *Gate Charge*].

Det samlede tab kan approksimeres til (5.7) hvis det antages at ledningstab og switching tab er de største. Der er flere kilder til tab i MOSFET transistorer når de bruges til at switche på et load, men disse antages at være ubetydelige[23].

$$P_{Tab} = P_{\text{ledningstab}} + P_{SW} \quad (5.7)$$

Ledningstabet i transistoren kan simpelt findes ved on-modstanden $R_{DS(on)}$ og strømmen der løber mellem drain og source, $P_{ledningstab} = I^2 \cdot R_{DS(on)}$. Det antages at der maksimalt løber 5 A igennem transistoren. Ledningstabet for N og P-kanal vil således kunne approksimeres til følgende:

$$P_{ledningstab, IRLZ44N} = (5 \text{ A})^2 \cdot 25 \cdot 10^{-3} \Omega \approx 0,6 \text{ W} \quad (5.8)$$

$$P_{ledningstab, FQP27P06} = (5 \text{ A})^2 \cdot 70 \cdot 10^{-3} \Omega \approx 1,8 \text{ W} \quad (5.9)$$

Switching ledningstabet er proportionel med den tid transistoren er i området hvor den ikke er helt on eller off. Switching tabet vil kunne approksimeres med ligning (5.10)[21], hvor V_{in} er forsyningen til transistoren, I_{out} er strømmen igennem source og drain, F_{SW} er switching frekvensen, $t_{s(L-H)}$ og $t_{s(H-L)}$ er henholdsvis tiden det tager at skifte fra low-til-high og high-til-low på gaten.

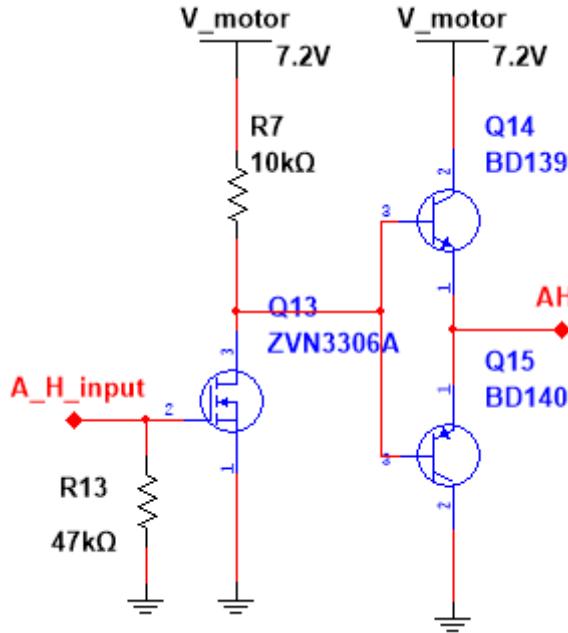
$$P_{SW} = \left(\frac{V_{in} \cdot I_{out}}{2} \right) \cdot F_{SW} \cdot (t_{s(L-H)} + t_{s(H-L)}) \quad (5.10)$$

Switch tiderne $t_{s(L-H)}$ og $t_{s(H-L)}$ afhænger af drive trinnets evne til at lade gaten op. Det er derfor ønskværdigt at drive trinnet har en tilstrækkelig lav udgangsimpedans og kan leve tilstrækkelig høj strøm, så tiderne og derved effekttabet minimeres.

5.2.5.4 Gate drive kredsløb

Gate kredsløbet gør det muligt at tænde for MOSFET transistorerne hurtigere. Idet drivekredsløbet ser en MOSFET transistorerne som en kapacitans er rise-time og fall-time bestemt ud fra drivekredsløbets udgangsimpedans. Kapacitansen sammen med udgangsimpedansen og evt. printbaner danner et RC-led hvor tidskonstanten begrænser rise- og fall-time. Hurtigere switch tider er ønsket pga. effektiviteten, så driveren skal have lav udgangsimpedans og være i stand til at leve stor nok strøm til at imødekomme den ønskede switch tid.

Der er ønsket hurtig rise-time og hurtig fall-time, så drivekredsløbet skal være i stand til at både sink og source stor strøm til gaten.

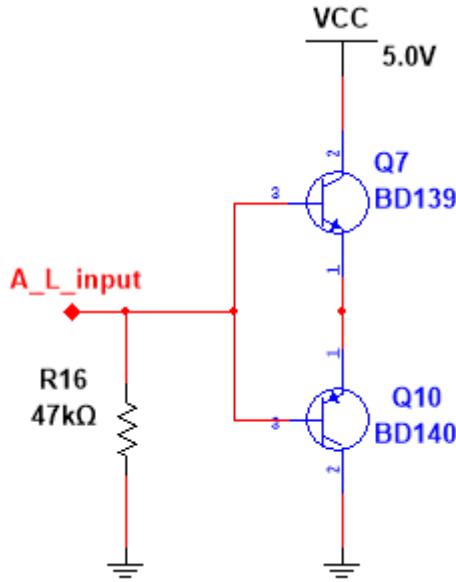


Figur 5.24: Drivekredsløb til high-side. Kredsløbet er dupligeret til de 3 outputgrene.

Figur 5.24 viser high-side drive kredsløbet. Kredsløbet er dupligeret i hver af de 3 output grene. Der anvendes et komplimentær emitter-følger output trin (Q14, Q15) til at levele/modtage ladningerne fra gaten. Q13 er en inverterende forstærker, som inverterer signalet således den høje side tænder når modulet får et højt signal. Q13 virker også som en buffer for PSoC'en, det er en N-kanal MOSFET og har høj impedans på gaten. Modstand R13 er en pull-down modstand så kredsløbet starter med at være slukket.

Det komplimentære emitter-følger trin har en input impedans givet ved $R_i \approx \beta \cdot R_E$ og output impedans $R_o \approx \frac{R_G}{\beta} > \frac{1}{g_m}$, hvor β er transistorens strømforstærkning, R_E er modstanden ved emitter, R_G er modstanden ved base og g_m er transistorens transkonduktans[19]. Ved et skift på MOSFET transistorens gate fra lav til høj, vil modstanden på emitterbenet først ses som en kortslutning, dvs. meget lav R_E . Dette betyder at emitter-følgeren har meget lav udgangsmodstand. Indgangsmodstanden starter meget lav, men stiger idet gaten oplades. Dette gør at det tidlige trin belastes i starten, og skal være i stand til at levele nok strøm. Ved opladning vil der løbe en strøm igennem R_7 og ind på basen af Q14, som så oplader gaten igennem emitterbenet. Udgangsmodenstanden af inverter trinnet ved opladning er således givet ved R_{10} .

Ved et skift på gaten fra høj til lav, vil modstanden på emitterbenet først ses som uendelig høj. Indgangsmodstanden starter derfor uendelig høj, men daler som gaten aflades. Udgangsmodenstanden afhænger af indgangs transistorens, Q13, on modstand.



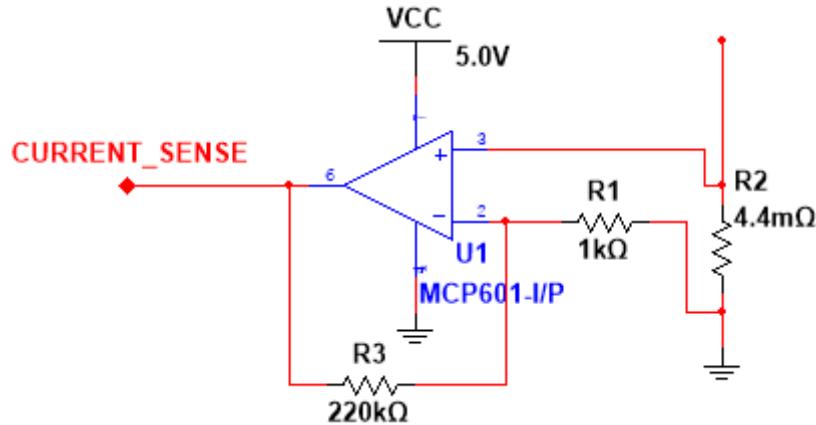
Figur 5.25: Gate driver til low-side. Kredsløbet er dupligeret til de 3 outputgrene.

Figur 5.25 viser drive kredsløbet til den lave side. Kredsløbet virker på lignende måde som driveren til den høje side, blot uden det inverterende indgangstrin.

Drive kredsløbet forbedrer tiden det tager at oplade og aflade gaten i forhold til hvis den var direkte tilkoblet. For at optimere på kredsløbet bør der evt. undersøges bedre muligheder for udgangstrinnet til gaten. Low-side driverne bør også have et forstærker-trin mere, således at transistorernes strømforstærkning kan udnyttes mere. Indgangs- og udgangsimpedanserne vil kunne forbedres ved evt. at anvende en anden topologi samt ved at tage forbehold for de forskellige trins udgangsimpedanser (se modstand R7).

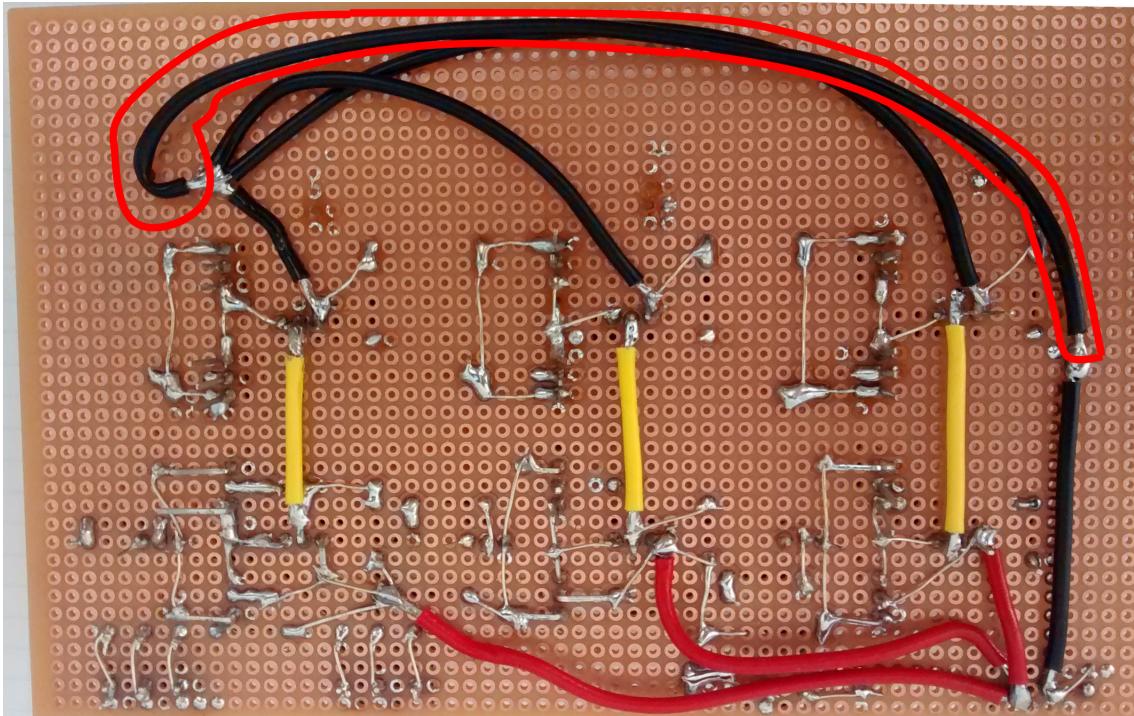
5.2.5.5 Current sense

Effektkredsløbet kan leveje større mængder strøm end der er termisk stabilt for transistorerne. Dette betyder at ESC controlleren skal begrænse hvor hårdt effektkredsløbet drives. For at gøre dette monitoreres strømmen der løber igennem effektkredsløbet med en shunt-modstand. Shunt-modstanden ideelt en højpræcisions lav-ohms modstand, hvor spændingen over den vil være proportionel med strømmen igennem effektkredsløbet.



Figur 5.26: Shunt-modstand (R2) med operationsforstærker. Outputtet herfra samples i ADC på PSoC.

Figur 5.26 viser current-sense kredsløbet. I dette tilfælde er der anvendt et ledningsstykke fremfor en reel modstand til shunt-modstanden (R2). Figur 5.27 viser ledningen monteret på print. Ledningen er af typen H05ZK med tværsnitsareal på $0,75\text{ mm}^2$ og er specificeret til at have en modstand på $26\Omega\text{ km}^{-1}$ tilsvarende $26 \cdot 10^{-3}\Omega\text{ m}^{-1}$ [24]. Der er anvendt et 17 cm langt ledningsstykke, så shunt-modstanden har en modstand $R = 26 \cdot 10^{-3}\Omega\text{ m}^{-1} \cdot 17 \cdot 10^{-2}\text{ m} \approx 4\text{ m}\Omega$. Med en strøm $I = 5\text{ A}$ vil spændingsfaldet over modstanden være $U = 4\text{ m}\Omega \cdot 5\text{ A} = 20\text{ mV}$. Dette forstærkes 220 gange så det nærmer sig et spænd fra 0 V til 5 V. Idet der anvendes et ledningsstykke som modstand vil værdien ikke kunne antages at være særlig nøjagtig. Dette er medregnet i effektkredsløbets sikkerhedsmargin.



Figur 5.27: Shunt-modstand: Ledningsstykke monteret på veroboard print.

5.2.5.6 EMC-mæssige overvejelser

Figur 5.27 viser shunt-modstanden monteret på print. Ulempen ved at gøre det med et ledningsstykke er det større areal strøm-sløjferne uundgåeligt vil have. Dette har EMC mæssig betydning for den støj der vil udledes fra kredsløbet. Idet der løber en forholdsvis stor strøm i kredsløbet skal der nøje planlægges hvordan printbanerne løber. Store strømsløjfer vil udlede kraftigere elektromagnetiske felter, og det er derfor ønskværdigt at minimere arealet hvorpå strømmen løber i[25].

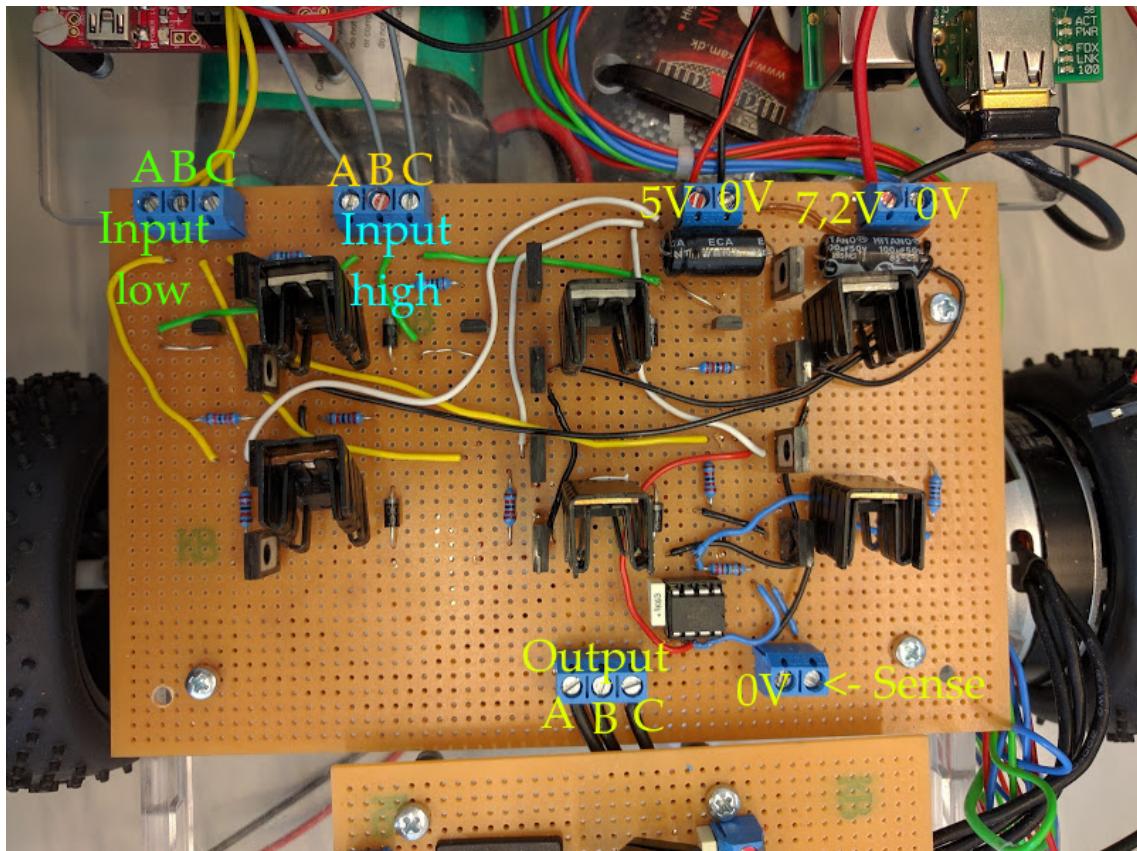
På figur 5.20 er dioderne D1 til D6 vist. De anvendes til at begrænse transienten der uundgåeligt dannes over motorenens spoler. Spændingen over en spole er proportionel med $\frac{dI}{dt}$, så der vil der opstå transient i det motoren pludselig tændes eller slukkes. Dioden på figuren er vist som 1N4007, og er en ordinær ensretter diode. Et bedre valg havde været en Schottky diode med tilstrækkelig max ledevne. Schottky har lavere spændings-tærskel før de begynder at lede, så den vil kunne tage mere af transienten end den ordinære ensretter diode.

Selve effekttrinnet bør være kraftigt afkoblet over spændingsforsyningen, så dyk i spændingen så vidt muligt kan undgås. Gate drive kredsløbet bør være afkoblet med en tilpas kondensator placeret fysisk tæt på kredsløbet.

Kredsløbet vil i et vis omfang kunne ombygges så det blev isoleret fra inputkredsløbet. En række optokoblere på indgangen vil isolere kredsløbet, og gøre det nemmere at undgå forstyrrelser andre steder i systemet.

5.2.5.7 Modultest

ESC-modulet er testet ud fra et blackbox princip, hvor der testes med forskellige inputs samtidig med at outputtet testes. Der whitebox testes på enkelte dele af modulet for at verificere



Figur 5.28: Det færdige modul på print. Input/output terminaler er markeret.

Ved blackbox testes der ved forskellige kombinationer af input. Der er ikke testet for ugyldige input, eksempelvis hvis der aktiveres høj og lav side på samme fase, da dette forventes at kunne ødelægge modulet. Der testes ved at sætte alle høj side input lav eller høj. Lav eller høj input svarer til lav / høj niveau fra PSoC input/output ben. Der testes ved 0 V og 5 V på inputtet. Outputtet er målt med multimeter mellem output terminalen og stel referencen på printet. Hvis multimeterets udskrift "drifter" eller ikke indstiller sig på en endelig værdi antages outputtet at være floating. Printet er forsynet fra en laboratorie forsyning med 5 V og 7,2 V, repræsentativ for den forventede forsyningsspænding fra batteriet.

Input - Høj side			Output		
A	B	C	A	B	C
L	L	L	floating	floating	floating
H	H	H	7V	7V	7V

Tabel 5.9: Input på høj side vs. output

Input - lav side			Output		
A	B	C	A	B	C
L	L	L	floating	floating	floating
H	H	H	0V	0V	0V

Tabel 5.10: Input på lav side vs. output

Tabel 5.9 og 5.10 viser input/output kombinationen for test ved input på henholdsvis høj side og lav side. Som det ses af tabellen, så passer outputtet forventeligt med kombinationen af input.

Motoren er desuden testet ved at forsyne den med laboratorieforsyningerne og motoren tilsluttet. Faserne tændes således at det kun er en af gangen der aktiveres. Hertil ses det at laboratorieforsyningen strømbegrænser ved 2 A, og at motoren giver betydelig modstand ved en af faserne.

Motoren er testet sammen med regulatoren, og kan med laboratorieforsyningen dreje rundt. Dette gælder også med batteriforsyning.

Kredsløbet bør stress testes ved at belaste den så den leverer 5 A til et load, og så monitorere temperaturen på transistorerne. Dette er ikke testet grundet tidsmangel.

Rise- og fall-tider på gaten bør undersøges nærmere for at undersøge om de er på et tilstrækkeligt niveau. Dette er ikke testet grundet tidsmangel.

5.2.6 Montering af Hall sensor

I projektet er der valgt at bruge en brushless DC-motor. Det betyder at der skal komme feedback om motorens position for at få jævn omdrejning. Det er implementeret ved hjælp af tre Hall-effekt sensorer som kan give feedback om motorpositionen. Ved at bruge tre sensorer kan positionen af motoren bestemmes til motorstyringen. Det er valgt at bruge Hall-effekt sensorer i stedet for at bruge back-EMF. back-EMF fungerer ikke så godt ved lav hastighed da størrelsen af back-EMF er proportional med omdrejningshastigheden. back-EMF kræver også en open-loop start hvor motoren ikke kører så godt[26]. Da motoren skal virke ved lave hastigheder er det en fordel at implementere motorfeedback med Hall sensorer. Der skal bruges tre Hall-effekt sensorer implementeret som beskrevet i afsnittet om Hall-effekt sensorer. I det følgende beskrives hvordan de anvendte Hall-effekt sensorer skal placeres i motoren.

Motoren består af 12 spoler. Der er to slags grader: elektriske grader og mekaniske grader. De mekaniske grader(M°) er det antal omdrejninger motoren drejer fysisk. 360 elektriske grader svarer til at to magneter er drejet forbi en spole. Antal mekaniske grader pr. elektrisk rotation kan udregnes ved at dividere 360 mekaniske grader med antallet af pol-par. I den anvendte motor er der 14 magneter og dermed 7 pol-par:

$$\frac{M^\circ}{E_{\text{rot}}} = \frac{360}{\text{pp}} = \frac{360}{7} = 51.4^\circ$$

Softwaremæssigt bliver motorstyringen implementeret med seks trin og faserne skifter hver gang motoren er drejet 120° elektriske grader. Det er derfor vigtigt at Hall-effekt sensorer kan registrere når motoren har drejet 120 elektriske grader. Antallet af mekaniske graders rotation pr. 120 elektriske grader kan så udregnes:

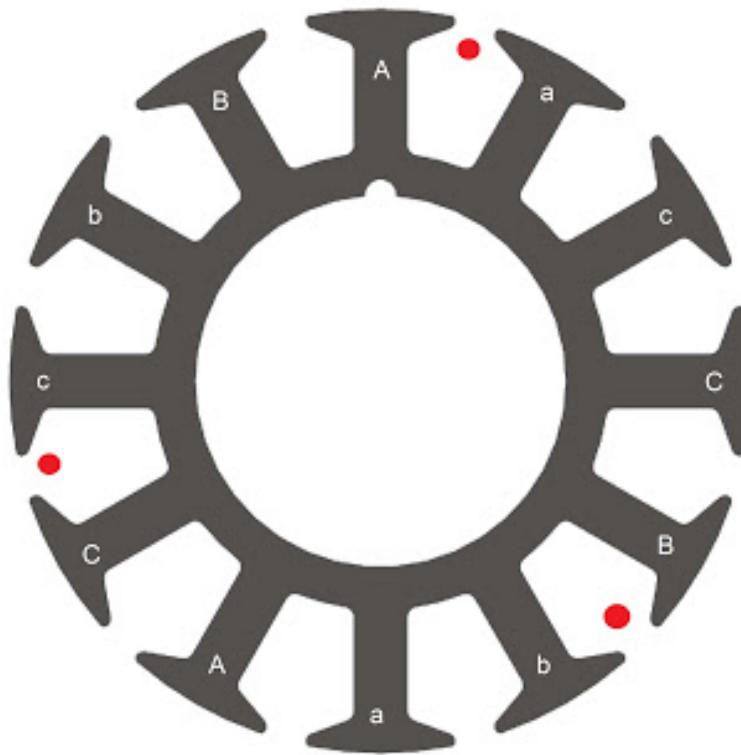
$$\frac{M^\circ}{E_{120^\circ}} = \frac{360}{3 \cdot \text{pp}} = \frac{360}{7} = 17.1^\circ$$

Det er valgt at placeringen af Hall-effekt sensorer skal være i mellem spolerne. Det er ikke muligt at placere hall sensorerne udenfor motoren da de valgte Hall-effekt sensorer ikke kan registrere magneterne. Antallet af mekaniske grader pr. hul i mellem spolerne kan udregnes:

$$\frac{M^\circ}{\text{spole}} = \frac{360}{12} = 30^\circ$$

Ligningen der skal løses for at få antallet af spolerne der kan være i mellem spolerne:

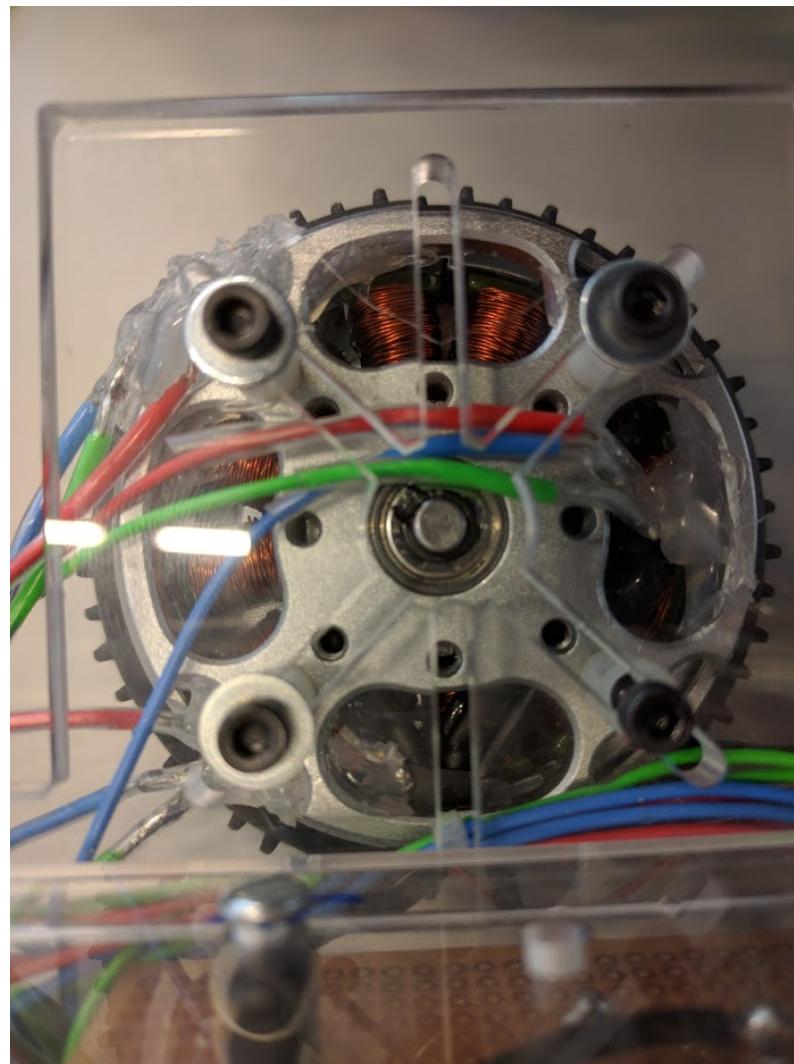
$$\frac{E_{120^\circ} \cdot n}{M_{\text{spole}}} = m$$



Figur 5.29: Placering af de 3 hall sensorer i mellem spolerne i motoren [26]

Her er m antallet af spoler og n er et helt tal. Det første n hvor ligningen har en løsning er 7 hvilket giver fire spoler i mellem hver Hall-effekt sensor. Da der kun er 12 spoler er den eneste mulighed fire spoler mellem hver Hall-effekt sensorer. For at montere Hall-effekt sensorer er motoren åbnet og Hall-effekt sensorer er sat fast med varm lim i mellem spolerne. Sensorerne er placeret så tæt på 120° som muligt se fig 5.29 for en tegning af placeringen af Hall-effekt sensorer . Selv med en åben motor er det svært at placere Hall-effekt sensorerne da de skal placeres på den side af motoren der ikke kan åbnes. Det betyder at sensorerne ikke sidder så præcist på motoren som ønsket.

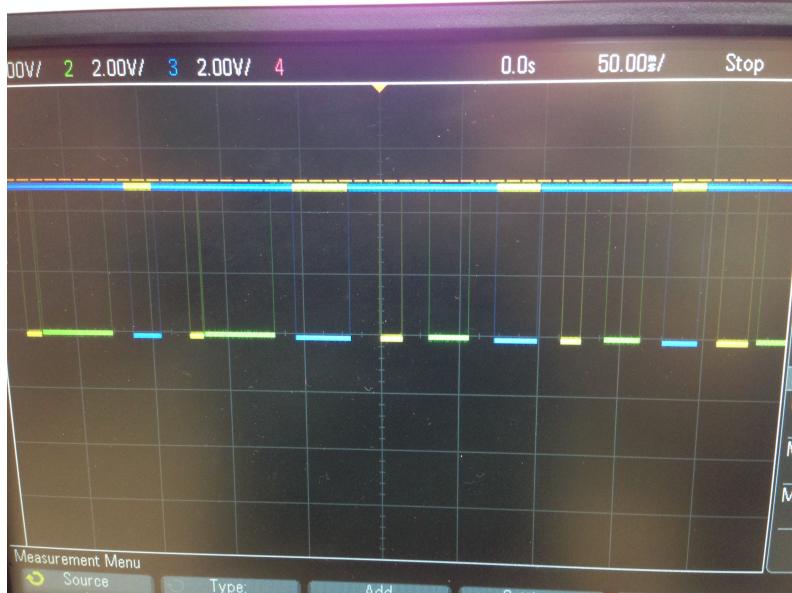
På fig. 5.30 kan et billede af den endelige montering af Hall-effekt sensorer ses.



Figur 5.30: Billede af Motoren efter Hall-effekt sensorer er monteret

5.2.6.1 Modultest

I testen af placeringen er der målt på de tre outputs fra Hall sensorerne samtidig med at motoren er drejet manuelt rundt. På fig 5.31 kan outputtet fra de 3 Hall sensorer ses. Hall-effekt sensorerne skal give et højt output halvdelen af tiden og lavt den anden. Det kan ses at det ikke er tilfældet for de monterede Hall-effekt sensorer. Når alle tre Hall sensorer er høje på samme tid er det ikke sikkert at alle 6 konfigurationer af output har et unikt input fra Hall-effekt sensorerne. Det betyder at motorstyringen ikke bliver så præcis som ønsket da motoren har "døde" punkter hvor der ikke er et output.



Figur 5.31: Output fra de tre Hall-effekt sensorer. Hall-effekt sensorerne skal være høj halvdelen af tiden.

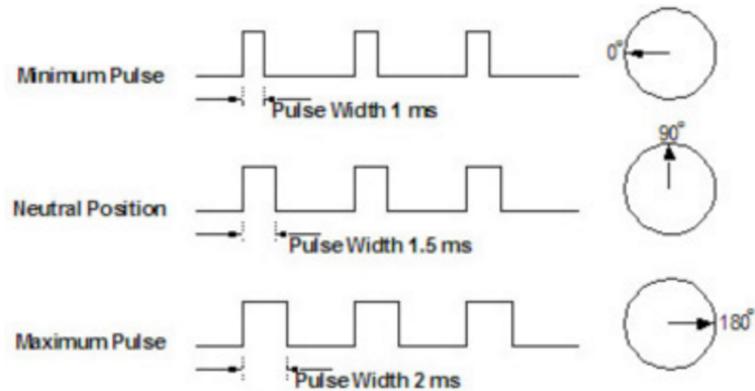
5.2.7 Servomotor

Til at styre retningen på bilen bruges en servomotor, der kobles til bilens forhjul. Servomotoren har tre ledninger, forsyningsspænding (rød), stel (sort) og signal (hvid). Servomotorens vinkel kan styres vha. et PWM-signal på signalledningen, hvor signalets duty-cycle bestemmer vinklen.

En servomotor kan typisk dreje 90 grader i begge retninger, fra et udgangspunkt fra 0 grader. Dette kan variere, hvor især kraftigere motorer kan have mindre rækkevidde. Motoren der bruges i dette projekt har en rækkevidde på 180 grader i alt. I forhold til PWM signalet, forventer servomotoren en puls hvert 20. millisekund. Dette giver en frekvens på

$$f = \frac{1}{0.02s} = 50\text{Hz}$$

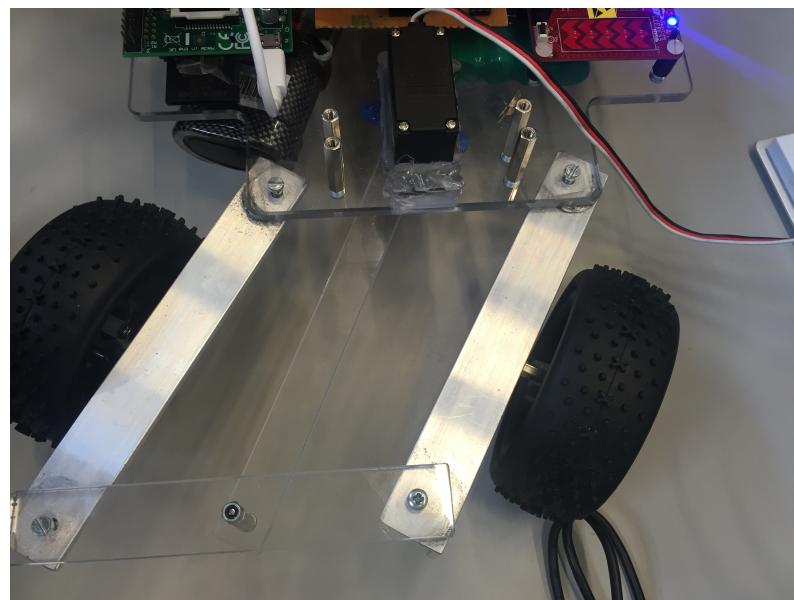
På figur 5.32 ses de pulslængder, der giver en vinkel på 0, 90 og 180 grader. Med en periode på 50Hz giver det en en duty-cycle på henholdsvis 5%, 7.5% og 10%. [27]



Figur 5.32: Princippet i servomotoren

5.2.7.1 Modultest

Til at teste servoens positioner bruges Analog Discovery for at simulere et PWM-signal, samt en 5V spændingsforsyning. På figur 5.33 ses testopstillingen for servoen. Den hvide ledning forbinder til PWM signalet fra Analog Discovery, den røde forbinder til 5V og den sorte forbinder til stel.



Figur 5.33: Modultest af servo

Med udgangspunkt i de teoretiske værdier, og med tanke på at hjulene ikke skal dreje helt ud i 180 og 0 grader, findes følgende værdier:

Venstre	5%
Midt	6.5%
Højre	9%

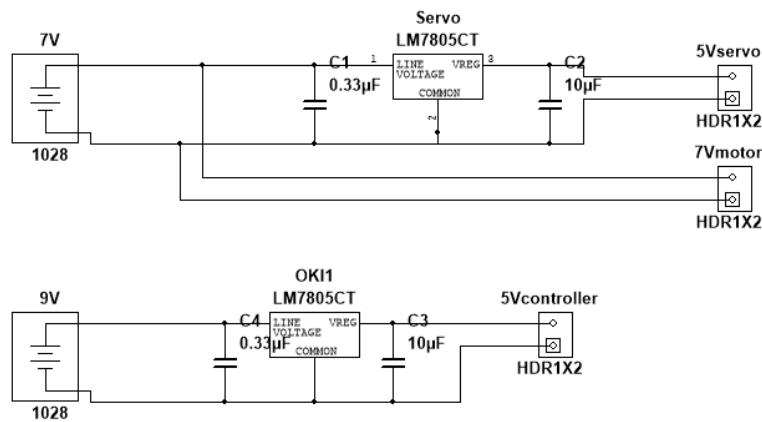
Dette viser at servoen ikke har de teoretiske værdier. Dette er afvigelser der kan tages højde for ved kalibrering softwaremæssigt.

5.2.8 Forsyning

Dette afsnit vil komme ind på hvordan forsyningsprintet er designet. Dette print skal forsyne motor, servo, Raspberry Pi, PSoC, og sensorer.

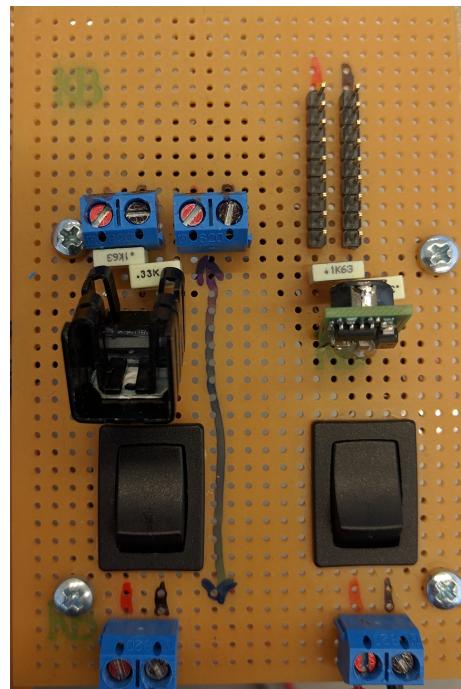
Bilen bruger to batterier, et på 9.6V og et 7.2V. Det er besluttet, at 9.6V batteriet skal bruges til Raspberry Pi, PSoC og sensorer, da det har den laveste kapacitet. Da der kun skal bruges 5V fra det batteri, er der valgt en DC/DC-converter.[28] Den kan levere 1.5 A, hvilket er rigeligt, til det der trækkes af strøm fra de forskellige dele.

Batteriet på 7.2V skal bruges til motoren, da det har flere amperetimer. Der skal ikke reguleres på spændingen til motoren. Der skal dog kun være 5V over servomotoren. Til det er der valgt en LM7805 [29], da den kan holde til strømme op til 1A, og servo'en kommer aldrig til at trække mere end 700mA



Figur 5.34: Kredsløbsdiagram for forsyningskredsløb

På figur 5.34 ses hvordan der reguleres på batterierne. Det er værd at lægge mærke til, at DC/DC regulatoren er en drop-in udskiftning for LM7805, hvilket betyder, at det samme kredsløb kan bruges.

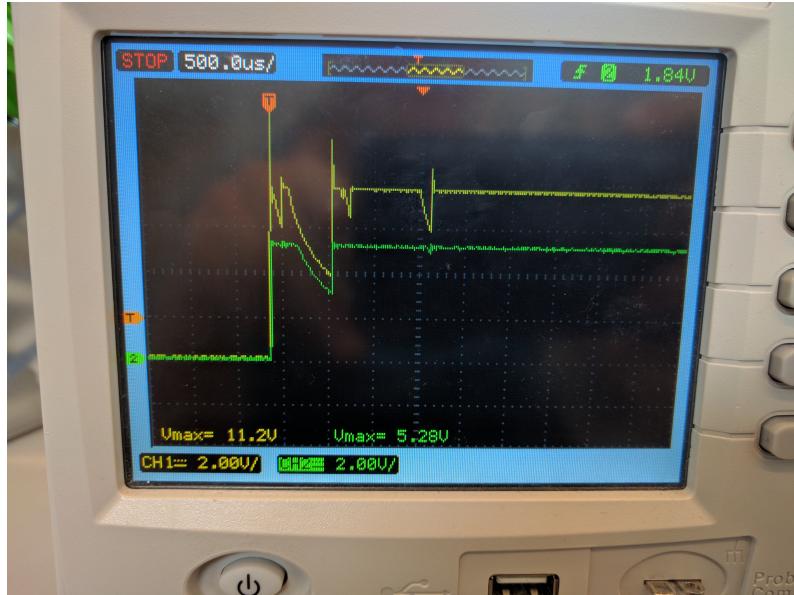


Figur 5.35: Det færdige print

På figur 5.35 kan det ses, at der over det planlagte kredsløb er monteret to store kontakter. De skal sikre, at bilen kan slukkes på en sikker måde, uden der risikeres kortslutning af nogle batteriledninger. Knapperne er designet til 230V, så de kan sagtens holde til de forholdsvis små spændinger.

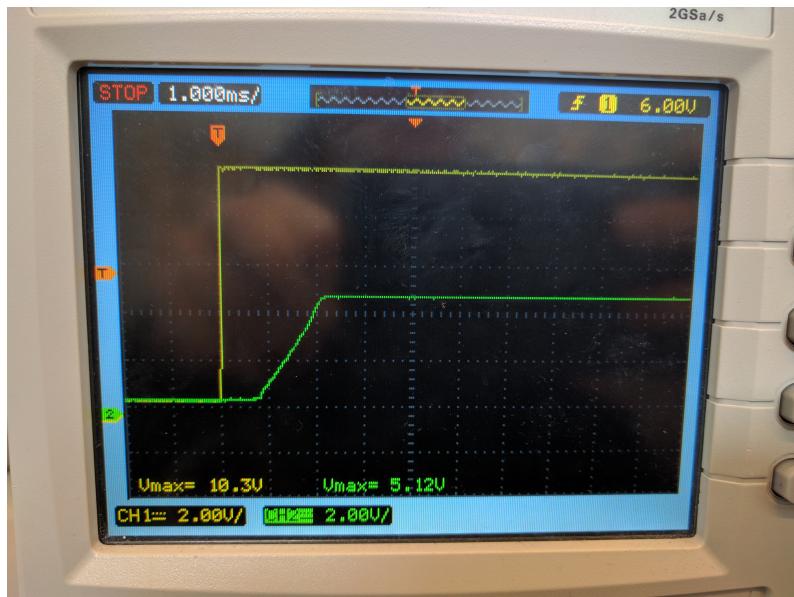
5.2.8.1 Modultest

For at teste forsyningsprintet er der kigget på hvordan regulatorerne opfører sig under opstart og efterfølgende. Det ønskes ikke, at der er et betydeligt overshoot og ripples på spændingen, når det forsynes.



Figur 5.36

På figur 5.36 ses indsvingningen for LM7805. Det ses, at den hurtigt indfinder sig på de forventede 5V. Der sker heller ikke noget oversving, som er værd at kommentere på. Efter indsvingningen ses der ikke nogen form for ripple.



Figur 5.37

På figur 5.37 ses det, at DC/DC-converteren er lidt længere om at svinge ind end LM7805. Dog har dette ikke nogen rigtig betydning. Der er heller ikke noget oversving eller ripples her.

5.3 Softwaredesign, -implementering og -test

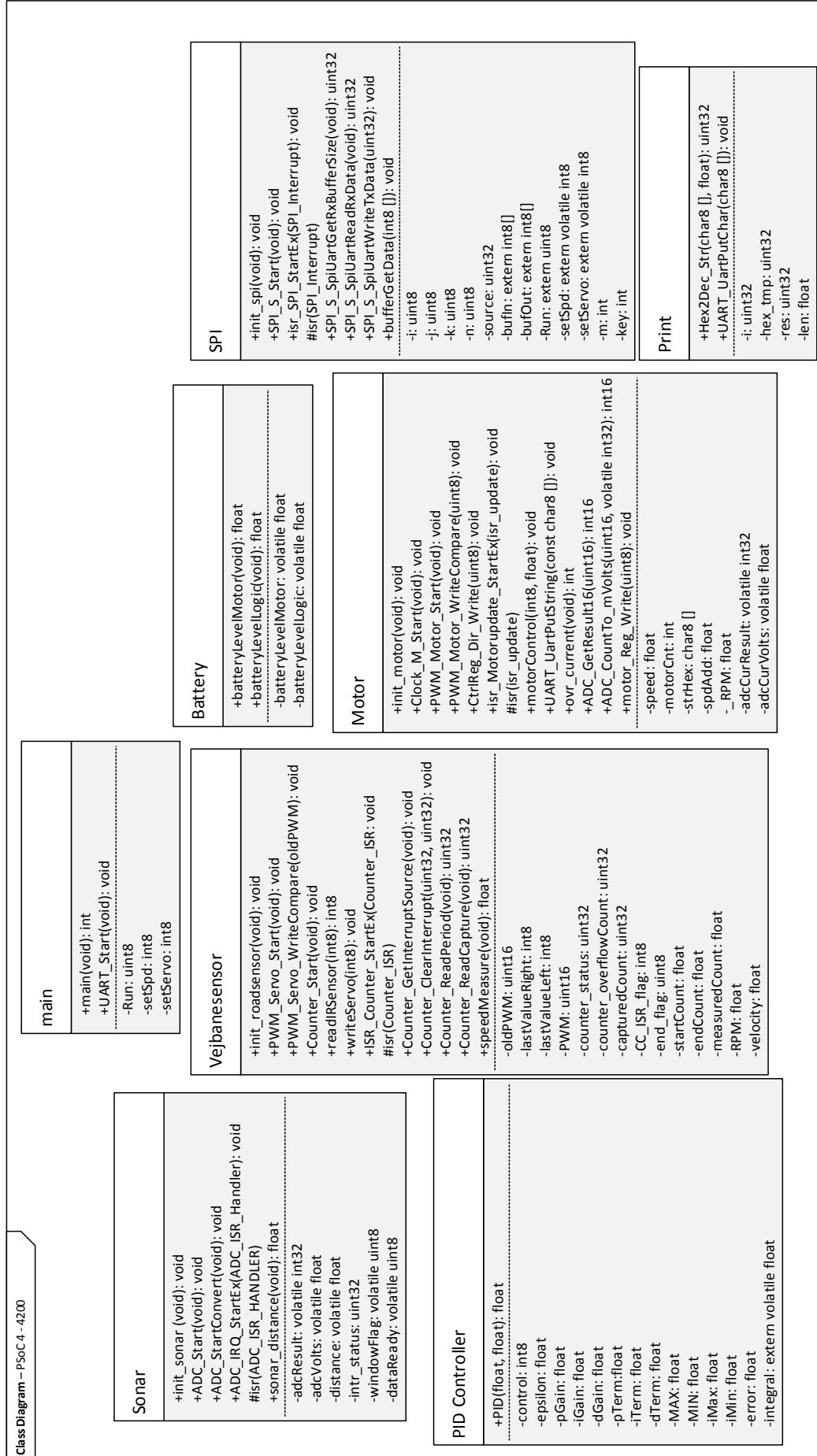
De følgende tre afsnit gennemgår software-arkitekturen for hhv. PSoC 4, Raspberry Pi, og GUI.

5.3.1 PSoC-software

De følgende afsnit giver indsigt i koden på PSoC'en. Først gennemgås den overordnede softwarearkitektur med et klassediagram og tilhørende funktionsbeskrivelser. Derefter gennemgås hver af de syv blokke (main foruden) i deres eget afsnit.

5.3.1.1 Overordnet PSoC arkitektur

PSoC 4 er kort sagt ansvarlig for at indsamle data fra sensorerne, styre de to motorer, regulere farten, og kommunikere over SPI (og UART). PSoC er programmeret i PSoC Creator 3.3. Udviklingsmiljøet har indbyggede funktionsblokke, der forbinder pins på PSoC 4 til udviklingsboardet. På den måde kan der spares tid ved at benytte indbyggede funktioner som f.eks. ADC og Counter's. På næste side ses klassediagrammet for AutoCar.



Figur 5.38: Klasediagram for PSoC-software

Nedenfor beskrives hver af funktionerne.

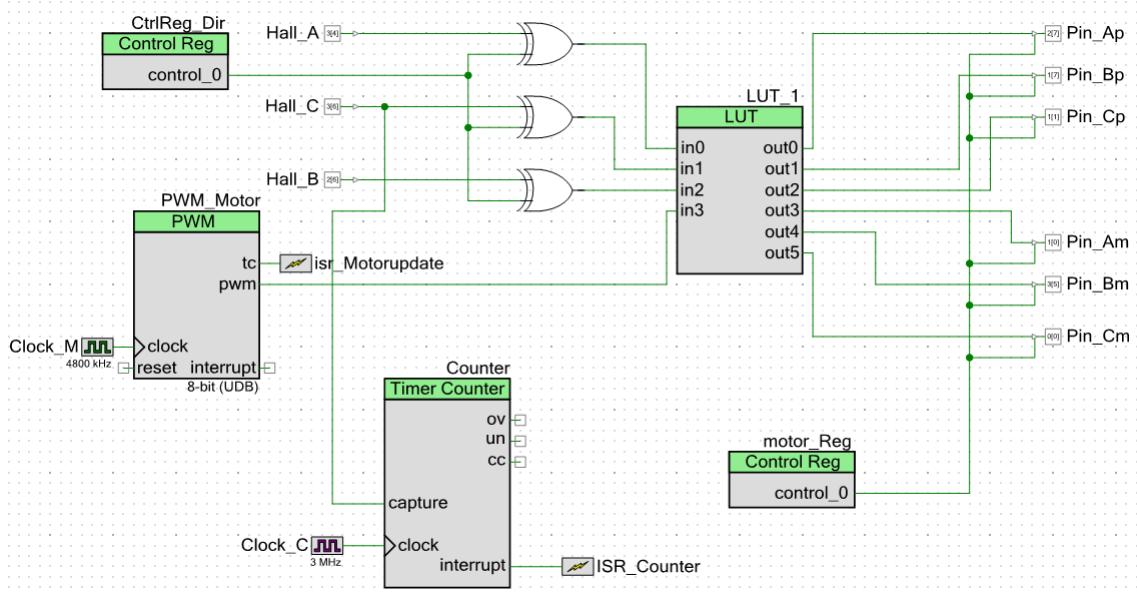
Main	UART_Start
Prototype	void UART_Start(void)
Kort beskrivelse af funktionalitet	Starter UART til brug på PuTTy-terminal
Sonar	init_sonar
Prototype	void init_sonar(void);
Kort beskrivelse af funktionalitet	Initierer sonaren
Sonar	ADC_Start
Prototype	void ADC_Start(void);
Kort beskrivelse af funktionalitet	Starter ADC'en til sonar og overcurrent
Sonar	ADC_StartConvert
Prototype	void ADC_StartConvert(void);
Kort beskrivelse af funktionalitet	Starter ADC-konvertering
Sonar	ADC_IRQ_StartEX(ADC_ISR_Handler)
Prototype	void ADC_Start(void);
Kort beskrivelse af funktionalitet	Tildeler ADC'en en interrupt-linje
Sonar	isr(ADC_ISR_Handler)
Prototype	isr(ADC_ISR_Handler)
Kort beskrivelse af funktionalitet	ADC-interrupt til sonar
Sonar	sonar_distance
Prototype	float sonar_distance(void);
Kort beskrivelse af funktionalitet	Sonarens afstand til nærmeste objekt
PID Controller	PID
Prototype	float PID(float, float);
Kort beskrivelse af funktionalitet	Controller til regulering af DC-motor
Print	Hex2Dec_Str
Prototype	uint32 Hex2Dec_Str(char8 [], float);
Kort beskrivelse af funktionalitet	Oversætter float til uint32 i en buffer
Print	UART_UartPutChar
Prototype	void UART_UartPutChar(char8 []);
Kort beskrivelse af funktionalitet	Outputter tal til PuTTy-terminal
SPI	init_spi
Prototype	void init_spi(void);
Kort beskrivelse af funktionalitet	Initierer SPI
SPI	SPI_S_Start
Prototype	void SPI_S_Start(void);
Kort beskrivelse af funktionalitet	Starter SPI-slaven på PSoC'en
SPI	isr_SPI_StartEx
Prototype	void isr_SPI_StartEx(SPI_interrupt);
Kort beskrivelse af funktionalitet	Starter SPI-interrupt

SPI	SPI_S_SpiUartGetRxBufferSize
Prototype	void SPI_S_SpiUartGetRxBufferSize(uint32);
Kort beskrivelse af funktionalitet	Finder størrelse på modtager-bufferen
SPI	SPI_S_SpiUartReadRxData
Prototype	void SPI_S_SpiUartReadRxData(uint32);
Kort beskrivelse af funktionalitet	Læser data fra modtagerbufferen
SPI	SPI_S_SpiUartWriteTxData
Prototype	void SPI_S_SpiUartWriteTxData(uint32);
Kort beskrivelse af funktionalitet	Skriver data til senderbufferen
SPI	bufferGetData
Prototype	void bufferGetData(int8 []);
Kort beskrivelse af funktionalitet	Navigerer modtager-arrayet og skriver de rigtige data til det rigtige sted
Battery	batteryLevelMotor
Prototype	float batteryLevelMotor(void);
Kort beskrivelse af funktionalitet	Motor-batteriets status
Battery	batteryLevelLogic
Prototype	float batteryLevelLogic(void);
Kort beskrivelse af funktionalitet	Logik-batteriets status
Vejbanesensor	init_roadsensor
Prototype	void init_roadsensor(void);
Kort beskrivelse af funktionalitet	Initierer vejbanesensor
Vejbanesensor	PWM_Servo_Start
Prototype	void PWM_Servo_Start(void);
Kort beskrivelse af funktionalitet	Starter servo-motorens PWM-blok
Vejbanesensor	PWM_Servo_WriteCompare
Prototype	void PWM_Servo_WriteCompare(void);
Kort beskrivelse af funktionalitet	Skriver værdi til servo-motorens PWM-blok
Vejbanesensor	Counter_Start
Prototype	void Counter_Start(void);
Kort beskrivelse af funktionalitet	Starter tachometerets Counter på Hall-effekt sensor C
Vejbanesensor	readIRSensor
Prototype	int8 readIRSensor(int8);
Kort beskrivelse af funktionalitet	Læser status på enten højre eller venstre vejbanesensor
Vejbanesensor	writeServo
Prototype	void writeServo(int8);
Kort beskrivelse af funktionalitet	Behandler servo-motoren

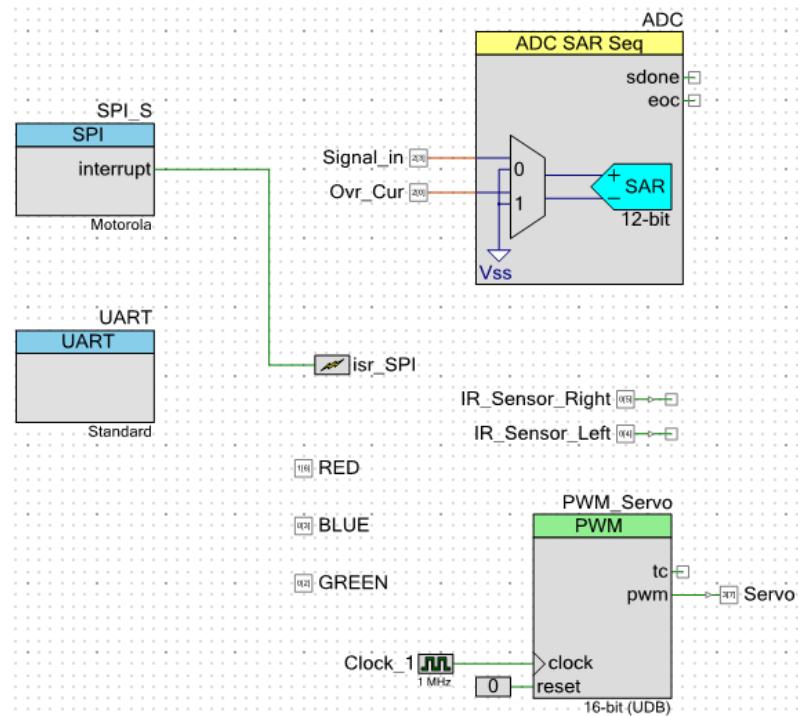
Vejbanesensor	ISR_Counter_StartEx
Prototype	void ISR_Counter_StartEx(Counter_ISR);
Kort beskrivelse af funktionalitet	Starter tachometerets Counter-interrupt
Vejbanesensor	isr(Counter_ISR)
Prototype	isr(Counter_ISR)
Kort beskrivelse af funktionalitet	Tachometerets Counter-interrupt
Vejbanesensor	Counter_GetInterruptSource
Prototype	void Counter_GetInterruptSource(void);
Kort beskrivelse af funktionalitet	Læser counterens status-register
Vejbanesensor	Counter_ClearInterrupt
Prototype	void Counter_ClearInterrupt(uint32, uint32);
Kort beskrivelse af funktionalitet	Clear counterens interrupt
Vejbanesensor	Counter_ReadPeriod
Prototype	uint32 Counter_ReadPeriod(void);
Kort beskrivelse af funktionalitet	Læser counter-period, som er sat i counter-konfigurationen
Vejbanesensor	Counter_ReadCapture
Prototype	uint32 Counter_ReadCapture(void);
Kort beskrivelse af funktionalitet	Læser counterens værdi på falling edge af capture-signalet
Vejbanesensor	speedMeasure
Prototype	float speedMeasure(void);
Kort beskrivelse af funktionalitet	Måler motorens fart med tachometeret
Motor	init_motor
Prototype	void init_motor(void);
Kort beskrivelse af funktionalitet	Initierer motoren
Motor	Clock_M_Start
Prototype	void Clock_M_Start(void);
Kort beskrivelse af funktionalitet	Starter motor-clock
Motor	PWM_Motor_Start
Prototype	void PWM_Motor_Start(void);
Kort beskrivelse af funktionalitet	Starter motorens PWM-blok
Motor	PWM_Motor_WriteCompare
Prototype	void PWM_Motor_WriteCompare(uint8);
Kort beskrivelse af funktionalitet	Skriver en værdi til motorens PWM-blok
Motor	Ctrl_Dir_Write
Prototype	void Ctrl_Dir_Write(uint8);
Kort beskrivelse af funktionalitet	Skriver en retning til motoren, frem eller tilbage

Motor	isr_Motorupdate_StartEx
Prototype	void isr_Motorupdate_StartEx(isr_update);
Kort beskrivelse af funktionalitet	Starter motorens interrupt
Motor	isr(isr_update)
Prototype	isr(isr_update)
Kort beskrivelse af funktionalitet	Motorens interrupt
Motor	motorControl
Prototype	void motorControl(int8, float)
Kort beskrivelse af funktionalitet	Beregner reguleringen med kald til PID og skriver værdi til motoren med PWM_Motor_WriteCompare. Skriver til sidst fart og RPM ud på PuTTy-terminal
Motor	UART_UartPutString
Prototype	void UART_UartPutString(const char8 []);
Kort beskrivelse af funktionalitet	Skriver streng på UART
Motor	ovr_current
Prototype	int ovr_current(void);
Kort beskrivelse af funktionalitet	Sætter overcurrent-flaget
Motor	ADC_GetResult16
Prototype	int16 ADC_GetResult16(uint16);
Kort beskrivelse af funktionalitet	Får resultatet af ADC'ens konvertering af overcurrent-signalet
Motor	ADC_CountsTo_mVolts
Prototype	int16 ADC_CountsTo_mVolts(uint16, volatile int32);
Kort beskrivelse af funktionalitet	Oversætter ADC'ens resultat til mVolt
Motor	motor_Reg_Write
Prototype	void motor_Reg_Write(uint8);
Kort beskrivelse af funktionalitet	Kontrolregister der kan tænde eller slukke for DC-motoren uafhængigt af motorens PWM.

Ud over disse funktioner genererer PSoC'en automatisk funktioner og definitioner ud fra hvilke blokke der benyttes i TopDesign. Nedenfor ses to screenshots af TopDesignet for programmet.



Figur 5.39: TopDesign side 1

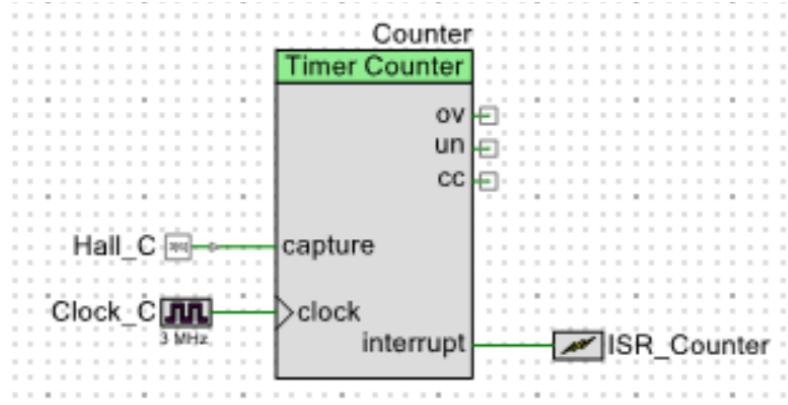


Figur 5.40: TopDesign side 2

Side 1 består af DC-motor blokken og Servo-counter der er forbundet til Hall_C.
 Side 2 består af SPI-slaven, UART til SPI og PuTTY-terminalen, ADC'en til sonaren og over-current, de to vejbanesensorer, PWM-blokken for servo-motoren, og endelig de tre LED'er på PSoC'en der har været brugt til test. Funktionaliteten af de forskellige blokke på begge sider i TopDesign er beskrevet i deres respektive afsnit.

5.3.2 Tachometer

For at måle farten bruges den ene af de tre hall sensorer som et tachometer. Det implementeres ved, at koble en timer-counter til den valgte hall sensor(Hall c). Opstillingen af det ses på figur 5.41.



Figur 5.41: Top design for tachometer

Timer-counter registrerer hvornår hall sensoren går lav. Ved at gemme to falling edge events, kan antallet af counts i en periode beregnes. Funktionen der håndterer tachometeret er

```
1 float speedMeasure()
```

Den returnerer farten som en float. ISR-Counter bruges til at sætte et flag, når der kommer en puls fra Hall-c. Hvis interruptet bliver forsaget af et compare event, gemmer den værdien. Samtidig tæller den et overflow op med én og clearer interruptet, hvis det bliver forsaget af et terminal count.

```

1 if ((counter_status & Counter_INTR_MASK_CC_MATCH) ==
2 Counter_INTR_MASK_CC_MATCH)
3 {
4     capturedCount = Counter_ReadCapture();
5
6     CC_ISR_flag = TRUE;
7     ...
8
9     if ((counter_status & Counter_INTR_MASK_TC) ==
10 Counter_INTR_MASK_TC)
11 {
12     counter_overflowCount++;
13     ...

```

Til at bestemme antal counts på en periode bruges formlen:

```
1 measuredCount = (counter_overflowCount * counter_periodCount)
2 + (endCount - startCount);
```

Den finder længden på perioden og lægger til, alt efter hvilken periode det er. Dette bruges til at bestemme periodetiden, ved at dividere med frekvensen der samples med. Ud fra denne tid kan omdrejningstallet og farten findes.

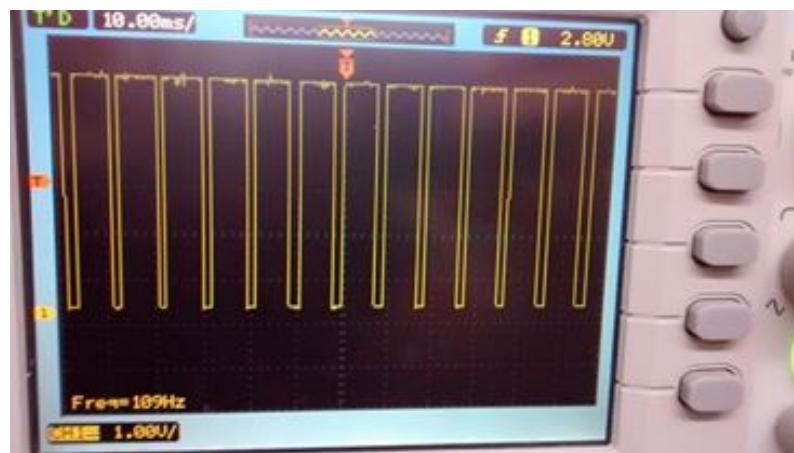
5.3.2.1 Modultest

For at teste tachometeret bruges et oscilloskop, til at finde frekvensen på output signalet fra hall-sensoren, når motoren kører. Dette er gjort på figur 5.42, hvor frekvensen er fundet til 109Hz. Ved at bruge formlerne for omdrejningstallet og farten fås

$$\text{RPM} = \frac{109\text{Hz} \cdot 60}{7} = 934.3$$

$$\text{fart} = \frac{2 \cdot \pi \cdot R \cdot \text{RPM}}{1000} \cdot 60 = 15.1\text{km/t}$$

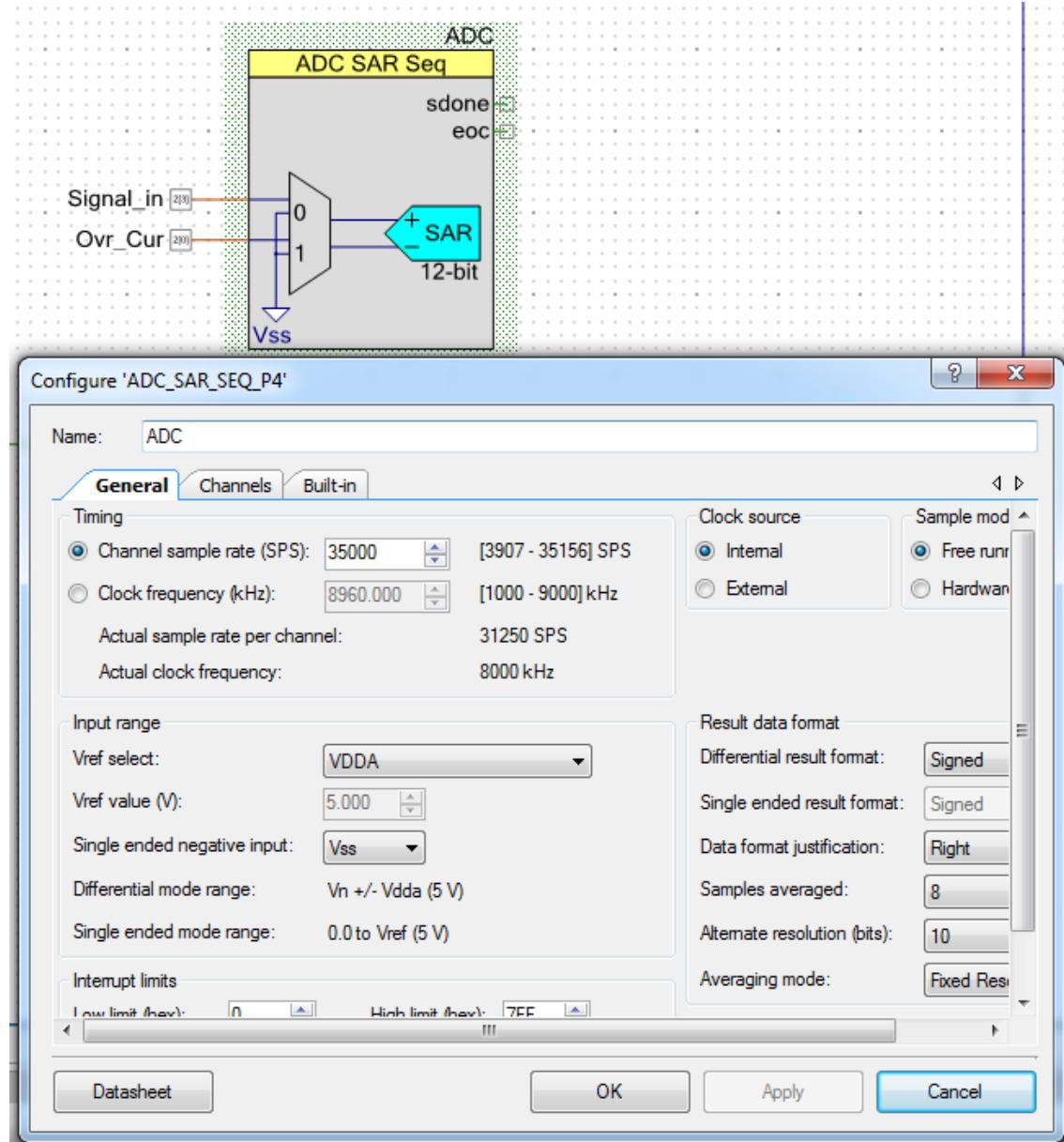
Ved at udskrive de beregnede værdier fra PSoC'en på en terminal, fås ved en fart på 15km/t et omdrejningstal på 869. Det viser, at tachometeret er tilstrækkeligt præcist i forhold til de krav der er blevet stillet.



Figur 5.42: Modultest af tachometer

5.3.3 Sonar

Sonarenes output er forbundet til PSoC 4s SAR ADC. ADC'en er sat op til single, med VDDA som reference-spænding. Det betyder at ADC'en kan læse inputs mellem 0 og 5V. 5V ville svare til en afstand for sonaren på $\frac{5\text{V}}{3.787\frac{\text{mV}}{\text{cm}}} - 2.561\text{cm} = 1318\text{cm}$, hvilket er mere end nok, når sonaren er begrænset til 6.45m. Nedenfor er ADC-blokken og dens settings i PSoC Creator vist.



Figur 5.43: ADC i PSoC Creator

ADC'en er 12 bit, hvilket vil sige, at dens højeste værdi er 4096. Sonaren skal ifølge kravene kunne give outputs mellem 20cm og 250cm. Uanset om databladets konvertering, fra cm til volt på sonaren, eller den senere kalibrerede værdi vælges, er der ikke brug for mere end $\cong 1000$ forskellige værdier fra ADC'en. Altså er 12 bits båndbredde rigeligt ift. det forventede input. SPI exchange foregår ved ca. 1000Hz, hvilket er beskrevet i SPI-afsnittet senere, blev det valgt at sætte "Samples averaged" til 8. Det gør sonaren mere præcis, og ADC'en vil stadig sample hurtigt nok. Sonarens interrupt-routines prototype ses nedenfor.

```
1 CY_ISR_PROTO(ADC_ISR_Handler);
```

I interruptet checkes det om ADC'en er færdig med at skanne sine kanaler. Dvs. hvorvidt End-Of-Scan masken er 1. Hvis ADC'en er færdig, sættes dataReady-flaget, hvilket medfører at sonar_distance kaldes i main(). Nedenfor er sonar_distance-funktionen vist.

```

1 float sonar_distance (void)
2 {
3     volatile int32 adcResult;
4     volatile float adcVolts;
5     volatile float distance= 0;
6
7     adcResult = ADC_GetResult16(CHANNEL_0);           // Get Reading
8     adcVolts = ADC_CountsTo_mVolts(CHANNEL_0, adcResult); // Convert to volts
9     distance = (adcVolts)/3.93-4;                  // Calibrated linear regression
10
11    return distance;
12 }
```

Data'en læses ind på ADC'en fra sonarenens analoge output. Det er vigtigt, at notere inputtet er sat til volatile, for at fortælle compileren, at værdien kan ændre sig når som helst. Ellers vil compileren typisk optimere disse variable ud, da den nemt kan finde værdien en gang og ikke bekymre sig om det igen. Volatile tvinger compileren til at behandle variablerne anderledes. ADC'en konverterer den analoge spænding på Channel 0-pinnen (PSoC'ens pin 2[3]), til en 32-bit integer. Derefter kan det omsættes til mV med CountsTo_mVolts. Ud fra regressionsanalysen kan det konverteres til distance. Ud fra kravspecifikationen skal sonaren kunne outputte værdier mellem 20 og 250 cm. Da der i spi-bufferen er plads til 255 forskellige værdier, skal distance bare konverteres fra float til spi-bufferens int8-format som ønsket. Kravene angiver desuden, at sonaren skal have en oplosning på 5 cm. Efter typecasting hvor decimalerne forsvinder, er der stadig en oplosning på 1 cm, fem gange bedre end specificeret.

Til at teste sonaren er PSoC'ens tre LED'er benyttet. Nedenfor ses et lille udpluk af testkoden i main, som er benyttet til test.

```

1 if(dataReady != 0u)
2 {
3     if(distance < 20)
4     {
5         GREEN_Write(1);
6         BLUE_Write(1);
7         RED_Write(0);
8     }
9     else if(20 < distance && distance < 100)
10    {
11        BLUE_Write(1);
12        RED_Write(1);
13        GREEN_Write(0);
14    }
15    else
16    {
17        RED_Write(1);
18        GREEN_Write(1);
19        BLUE_Write(0);
```

```

20     }
21 dataReady=0u;
22 }
```

Ved at ændre på værdierne i if/else statement, kan det ses hvilken afstand PSoC'en har vurderet det til at være samt få en af de tre farver på LED'en (eller flere). Resultatet sammelignes med målebåndet og oscilloskopet for at lave data-tabellen fra sonarens hardware-afsnit, og derefter regressionen. Under testen anvendes den regression, der er opgivet i databladet. Derefter justeres den til et mere præcist resultat, hvilket ses i den tidligere kode. Til sidst i testkoden skal dataReady resettes, så der først er adgang til en sonar-læsning næste gang sonar-interruptet har kørt.

5.3.4 Strømbegrænsning

For at beskytte motorkredsløbet er der implementeret en strømbegrænsning. Det er gjort med en modstand, så der kan måles en spænding med ADC'en på PSoC'en. Der bruges de samme ADC indstillinger som for sonaren. ADC'en kan læse inputs mellem 0 og 5V, hvilket stemmer overens med at strømbegrænsningen svarer til en spænding på 4,8V. Strømbegrænsningen bruger samme interrupt som sonaren, for at tjekke om ADC'en er færdig med at konvertere. Efter konverteringen kaldes ovr_current() funktionen for at kontrollere, om der løber en for høj strøm gennem motorkredsløbet. ovr_current() funktionen returnerer true, hvis strømmen gennem motorkredsløbet er for højt og motoren skal slukkes.

```

1 int ovr_current(){
2     volatile int32 adcCurResult;
3     volatile float adcCurVolts;
4
5     adcCurResult = ADC_GetResult16(CHANNEL_2);           /* Get
6         Reading */
7     adcCurVolts = ADC_CountsTo_mVolts(CHANNEL_2, adcCurResult)-100;    /*
8         Convert to volts */
9     //Offset svarer til ca. 4.8V
10    if(adcCurVolts > 4800){
11        return TRUE;
12    }
13    else{
14        return FALSE;
15    }
16 }
```

Strømbegrænsningen er testet ved at bruge PSoC'ens indbyggede LED og koden nedenunder

```

1 if(dataReady != 0u)
2 {
3     if.ovr_current)
4     {
```

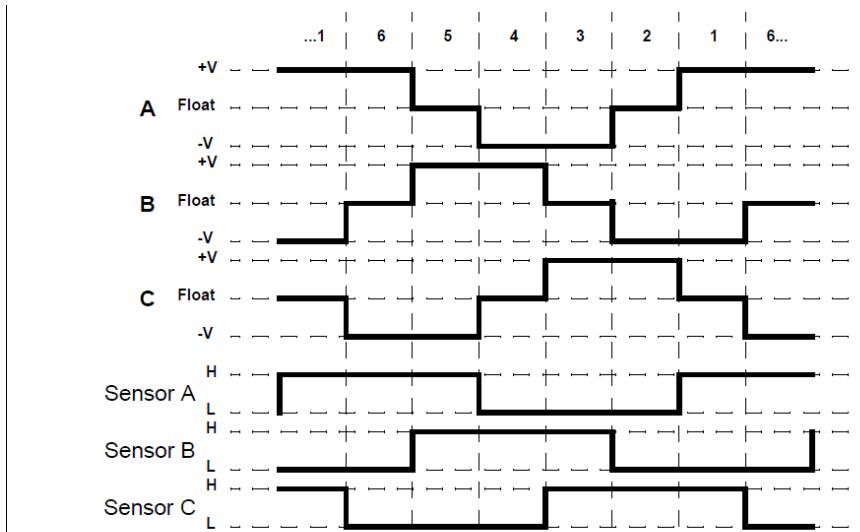
```

5         BLUE_Write(0);
6     }
7     else if(!ovr_current)
8     {
9         BLUE_Write(1);
10    }
11
12 dataReady=0u;
13 }
```

Det er testes ved at påføre PSoC'ens ADC-input forskellige spændinger. Det gøres indtil den blå LED lyser, hvilket indikerede at den påtrykte spænding er over den valgte grænse. For en påtrykt spænding på 4,8V, var det nødvendigt at trække 100 fra resultatet af PSoC'ens indbyggede funktion til at konvertere ADC'ens resultat til millivolt(ADC_CountsTo_mVolts). Det er ikke undersøgt hvorfor det er nødvendigt.

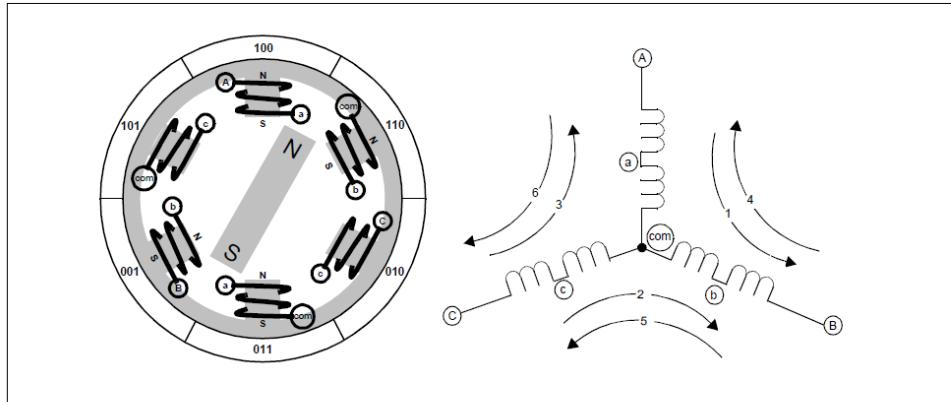
5.3.5 ESC

Motorstyringen er implementeret således outputtet er et 3 faset signal som enten er højt, lavt eller float. Se fig 5.44 for output til motoren. Ud fra inputtet fra Hall sensorerne skal outputtet ændres. På 5.44 kan inputtet fra Hall sensorerne ses. A, B og C er forbundet i ét punkt, så der er 6 forskellige måder faserne kan tændes og slukkes på, når der kun er 2 faser der er tændt og slukket af gangen.



Figur 5.44: Output fra Hall sensor A, B og C og input til motorfase A, B og C

På figur5.46 kan et simplificeret motorbillede ses. Her er magneten placeret indvendigt og ikke udvendigt, som i den anvendte motor, men principippet er det samme.



Figur 5.45: Simplificeret motorbillede[30]

Magnetens position kan opnås ved at lede strøm fra A til B, svarende til pil 1 på figuren. For at dreje magneten til positionen 60 grader mod uret skal strømmen løbe fra A til C svarende til pil 6 på figuren. For at opnå maksimal torque skal der tændes for den ønskede fase 90 elektriske grader, før den ønskede position opnås. Alle 6 mulige ændringer motoren kan udsættes for, kan ses på fig 5.44. Der er 6 output til ESC kredsløbet. Tre der kan sætte hhv. A, B og C høj og tre der kan sætte dem lav. På PSoC'en er det implementeret som en lookup tabel[30].

Tabel 5.11: Lookup tabel til motor output

Hall C	Hall B	Hall A	A høj	B høj	C høj	A lav	B lav	C lav
0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	1
0	1	0	0	0	1	0	1	0
0	1	1	1	0	0	0	1	0
1	0	0	0	1	0	1	0	0
1	0	1	0	1	0	0	0	1
1	1	0	0	0	1	1	0	0
1	1	1	0	0	0	0	0	0

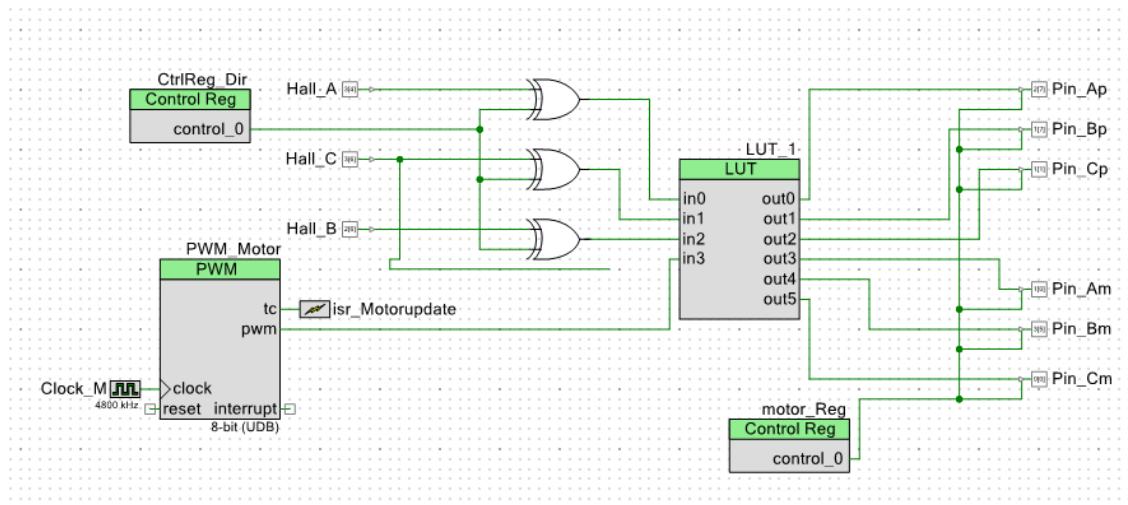
Der er kun output når alle sensorerne ikke er lave eller alle ikke er høje. For at bestemme lookup tabellen er det vigtigt, at der ikke tændes for både den høje og lave side på samme fase. Den implementerede lookup tabel er afhængig af, at det rigtige Hall sensor input passer med den rigtige indgang i lookup tabellen. For at finde den rigtige konfiguration af Hall sensor input, kan det testes ved at teste output fra Hall sensor ved forskellige input til motoren.

På figur 5.11 kan det ses, hvordan ESC'en er implementeret. De tre hall sensorer er igennem xor gates. Det skyldes, at motoren kører i den anden retning hvis output fra hall

sensorerne inverteres inden look up tabellen. Ændring af retning er kun at skrive til registeret.

Der er implementeret et motor Register som skal være 1 for at output til ESC kredsløbet ikke er sat til 0. Der bruges et PWM signal til at regulere spændingen over motoren. Dette gøres ved at sende et PWM signal ind i look up tabellen. Hvis PWM inputtet til lookup tabellen er lavt, er output til den lave side(pin_Am, pin_Bm og pin_Cm) upåvirket. Derimod vil de positive udgange til motoren blive sat til 0 i lookup tabellen.

Hvis PWM signalet er højt, vil de positive udgange være som om, at PWM signalet ikke var en del af look up tabellen(se fig 5.11 for lookup tabellen). PWM signalet bliver derfor kun implementeret på de positive udgange til motorstyringsprintet. PWM signalet kan bruges til at regulere farten på motoren. Dette er implementeret og beskrevet i controller afsnittet.



Figur 5.46: PSoC implementation af ESC. Det indeholder en lookup tabel til output. Et register til at sætte output lig 0 . Et andet register til at ændre retning samt et PWM-signal til at give PWM på output

5.3.5.1 Modultest

Ved test er tabel 5.12 kørt igennem og outputtet er aflæst på Hall sensor output. I tabellen ændres motoroutputtet og Hall sensorerne aflæses.

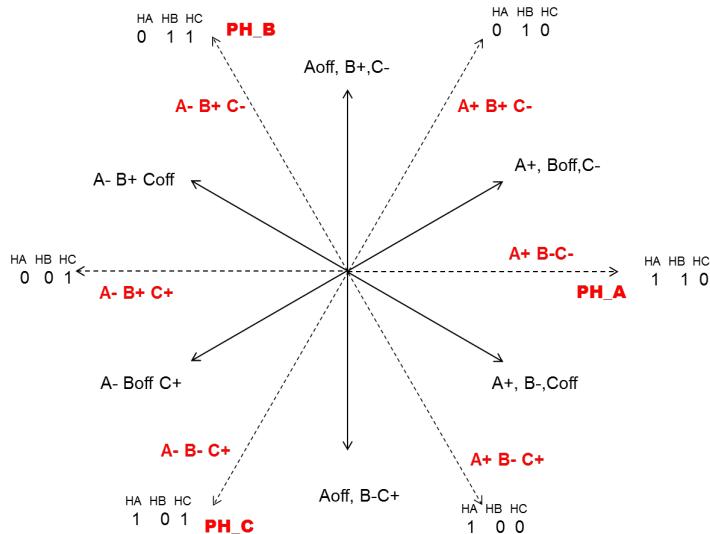
Tabel 5.12: Test tabel til Hall sensor konfiguration og output fra testen.

Nr.	A	B	C	Hall C	Hall B	Hall A
1	+	-	-	1	1	0
2	+	+	-	0	1	1
3	-	+	-	0	1	1
4	-	+	+	0	1	1
5	-	-	+	1	1	0
6	+	-	+	1	1	0

På tabel 5.12 kan det ses, at Hall sensor B giver et højt signal for de 6 motorinput. Hvis Hall sensorerne er lave halvdelen af tiden og høje den anden, er det muligt at bruge fig.5.47 til at bestemme konfigurationen af Hall sensorerne.

På Fig.5.47 ses de forskellige testoutput til motoren og det forventede resultat fra testen. Ved hver fase der er testet, kan det tilsvarende output som motoren skal have, findes ved at dreje 90 grader i den retning som rotationen svarer til.

Da Hall sensorerne ikke giver det forventede output, er der ikke 6 unikke kombinationer af Hall sensorerne. Det er derfor ikke muligt, at bestemme sensorkonfigurationen på denne måde. Fig 5.47 er brugt til at danne den brugte lookup tabel. Det er gjort ved at dreje 90° i den ønskede rotationsretning og aflæse hvad motorinputtet skal være. For $HA=0$, $HB=1$ og $HC=1$ samt rotation med uret, fås et motoroutput på A+, Boff og C-.



Figur 5.47: Testbillede af Hall sensor ved forskellige motoroutput[31]

Da det ikke var muligt, at bestemme konfigurationen af Hall sensorerne ved hjælp af ovenstående test, blev konfigurationen bestemt ved at ændre imellem Hall inputs. Output forbindes til ESC kredsløbet og derfra videre til motoren. Hall sensorerne ændres indbyrdes, indtil motoren begyndte at dreje rundt. Motoren kan have svært ved at starte, hvilket kan skyldes at Hall sensorerne ikke giver det ønskede output. Lige så snart motoren er begyndt at dreje rundt, betyder det ikke så meget, at der er "døde"områder, hvor output til motoren ikke har nogen spændingsforskæl. Det betød også, at motoren har svært ved at holde sig i gang ved lave omdrejninger.

5.3.6 Servo

Herunder ses prototypen til at læse vejbanesensorne:

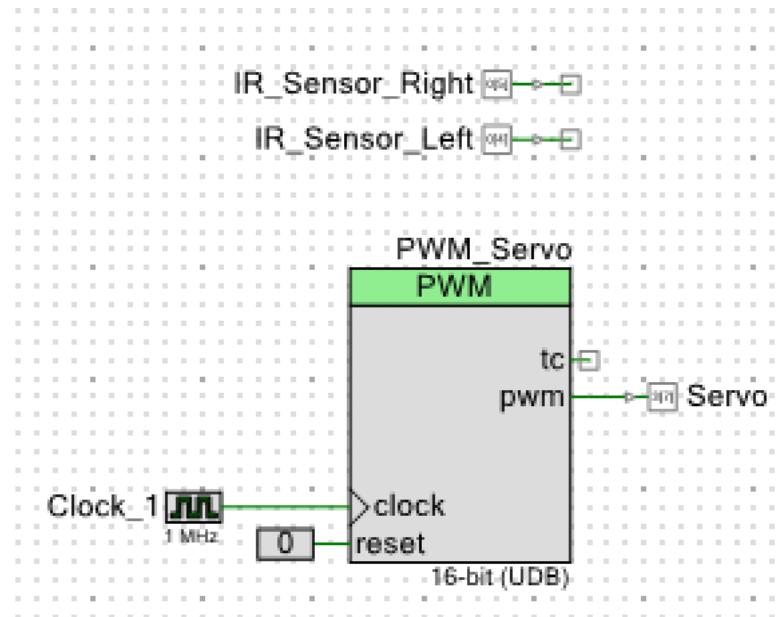
```
1 int8 readIRSensor(int8 sensor)
```

Dens parameter fortæller om det er den højre eller venstre sensor den skal læse fra, og returnerer et højt eller lavt signal alt efter om den detekterer en vejstriben eller ej. Herunder ses, at ved inputtet 0, skal funktionen læse fra den venstre vejbanesensor. Her bruges funktionen read til at læse det aktuelle output fra sensoren, hvilken der til sidst returneres.

```
1 if (sensor == 0)
```

```

2 {
3     lastValueLeft = IR_Sensor_Left_Read();
4     return lastValueLeft;
5 }
```



Figur 5.48: Topdesign for servo & vejbanesensor

For at kunne ændre retningen på bilen bruges en PWM-blok i PSoC'en, for løbende at kunne ændre det PWM-signal der skrives til servo'en. Top designet for denne del er vist på figur 5.48. Den opsættes til en periode på 20 ms. Ved en clock-frekvens på 1 MHz betyder det, der går 19999 counts på en periode. PWM-blokken opsættes til en oplosning på 16-bit, for at opnå en større interval, og dermed større nøjagtighed, på PWM-signalet.

Der modtages en værdi mellem -100 og 100 fra raspberry pi, som afhænger af hvor meget der skal drejes. -100 svarer til 100% til venstre og 100 svarer til 100% til højre. Prototypen til funktionen ser således ud

```
1 void writeServo(int8 dir)
```

Som parameter tager den, direction der sendes fra Raspberry Pi. For kunne skrive et PWM-signal til servo'en beregnes først den compare-værdi der skal bruges for at dreje til venstre, højre og ligeud.

Venstre	5%	1000
Midt	6.5%	1300
Højre	9%	1800

For at beregne direction om til det tilsvarende PWM-signal laves der to lineære former, én for at dreje til venstre og én for at dreje til højre. Dette opsættes med if-sætninger, som agerer efter hvad inputtet dir er. Nedenfor ses den if-sætning der bruges hvis bilen skal dreje til venstre. Ud over denne if-sætning er der en som drejer til højre, en der sætter den helt til venstre, en der sætter den helt til højre og en default til midten.

```
1 if (dir <= 0 && dir > -100)
```

```
2 PWM = 3*dir + 1300;
```

Til sidst udskrives PWM signalt til servoen med PSoC funktionen

```
1 PWM_Servo_WriteCompare(PWM);
```

5.3.7 Controller

Controlleren er ansvarlig for at styre motorens hastighed. En PID-controller forsøger at minimere fejlen på et system ud fra den ønskede værdi. Ved at trække den målte værdi fra den ønskede fås "fejlen". PID-controlleren behandler fejlen og sender en værdi tilbage til systemet. Værdien er delt op i tre led, produkt-leddet, integral-leddet, og differens-leddet. Ud fra inputtet og den ønskede værdi beregner controlleren en værdi, der skal tillægges eller trækkes fra inputtet. Over tid vil inputtet nærme sig den ønskede værdi. Formlen for en PID-controller i tidsdomænet ses nedenfor.

$$u(t) = K_p \cdot e(t) + K_i \int 0^t e(\tau) d\tau + K_d \cdot \frac{de(t)}{dt}$$

I LaPlace-domænet er formlen selvfølgelig

$$L(s) = K_p + \frac{K_i}{s} + K_d \cdot s$$

Produktleddet sender altså en lineær regulering tilbage til systemet kun afhængig af K_p og fejlen i det øjeblik controller-funktionen kaldes. Hvis K_p sættes for højt, kan det resultere i, at systemet bliver ekstremt ustabilt. På den anden side kan reguleringen blive for svag hvis leddet er for lille. Integral-leddet sender den akkumulerede fejl tilbage fra systemet blev startet. Fordi integral-leddet bliver større og større som tiden går, kan det medføre overshoot. Differens-leddet forsøger at forudsige den fremtidige værdi ud fra fejlens hældning over tid. Derved bidrager leddet med stabilitet og at systemet indstiller sig hurtigere. Differens-leddet implementeres dog sjeldent i praksis, da dets effekt kan være lidt svær at bedømme. Kun ca. en fjerdedel af anvendte controllere bruger dette led[32].

Reguleringen går ud på, at finde de rigtige værdier af K_p , K_i , og K_d hvilket ofte kan være svært. Hvis der kan opstilles en overføringsfunktion for systemet, eller et paent plot af systemets værdi over tid, er det muligt at bestemme nogle af parametrerne.

I AutoCar er der implementeret en PI-controller, da det blev vurderet at differens-leddet ikke var nødvendigt. Differens-leddet er dog stadig i funktionen, bare i comments, så den kan indsættes hvis det er nødvendigt. De mest interessante linjer fra Controller-funktionen ses nedenfor

```
1 ...
2 float error = setSpeed - measuredSpeed;
3
4 pTerm = pGain*error;
5 ...
6 iTerm = iGain*integral;
```

```

7
8 //dTerm = dGain(error - pre_error)/dt;
9
10 control = pTerm + iTerm; // + dTerm;
11 ...
12 //pre_error = error;

```

I funktionen er der også implementeret begrænsning. Integral-leddet slås fra hvis error er under en hvis grænse, og funktionen har en begrænsning på dens retur-værdi i begge retningner. Dermed undgås det, at controlleren overjusterer alt for hurtigt. Controller-funktionen kaldes hver gang motor-interruptet har kørt 19 gange. Dette er valgt fordi motorens reset time er $53.3\mu s$, og dermed går der ca. 19 resets på 1ms. Reguleringen foregår med ca. 1000Hz. For at teste funktionaliteten af PI-controlleren sættes et oscilloskop til Hall effekt-sensorerne og får PSoC'ens UART til at sende printouts til PuTTy-terminalen. Det gøres for at kunne måle omdrejningshastigheden hhv. fysisk og kodemæssigt. Forskellige værdier testes for K_p og K_i , og endte til sidst på hhv. 0.04 og 0.02. Derefter sættes speed til forskellige værdier i koden, og værdierne sammenlignes. For en fart på 15 km/t gav PuTTy 869 RPM, hvor oscilloskopet gav en frekvens på 100 Hz. Formlen for udregning af hjulets RPM er

$$RPM = \frac{f \cdot 60 \frac{\text{min}}{\text{h}}}{7} f$$

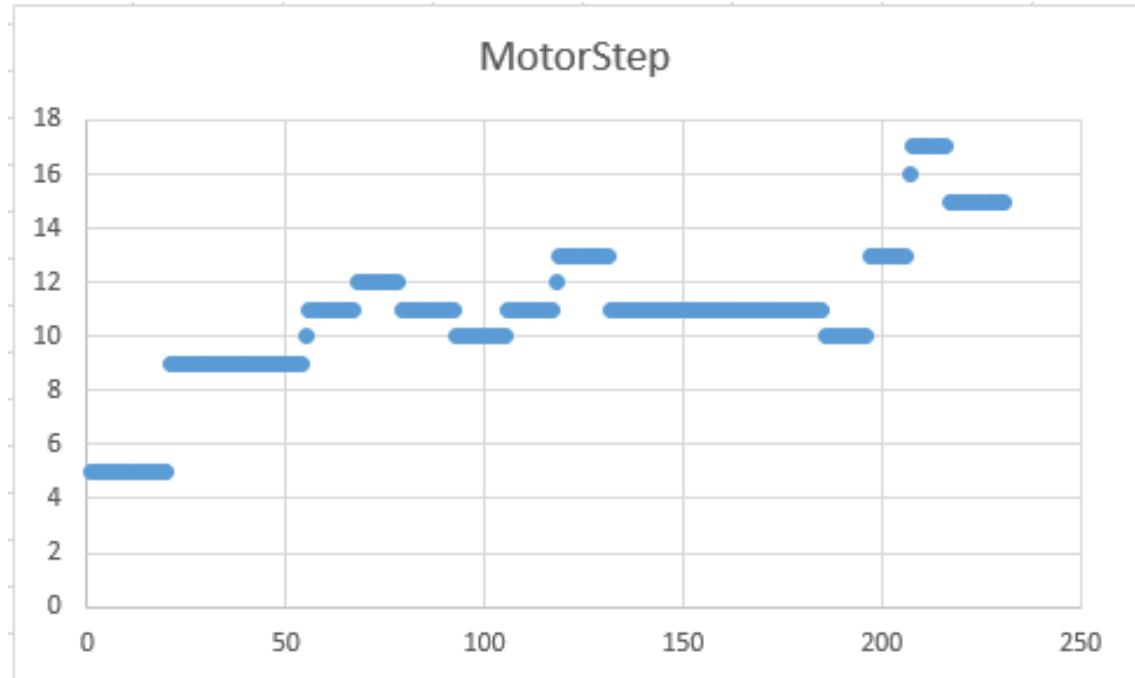
Den dimensionsløse konstant i nævneren skyldes, at syv magneter i hjulet opfanger et signal med Hall-effekten. Derfor skal der divideres med disse, for at få en hel omgang. Den fysiske RPM er altså

$$RPM = \frac{100\text{Hz} \cdot 60 \frac{\text{min}}{\text{h}}}{7} f \cong 857$$

For en fart på 30 km/t gav PuTTy-terminalen 1754 RPM, og oscilloskopet gav en frekvens på 200 Hz svarende til

$$RPM = \frac{200\text{Hz} \cdot 60 \frac{\text{min}}{\text{h}}}{7} f \cong 1714$$

Alt i alt passer disse resultater godt. For at forsøge at bestemme nogle af værdierne, blev der lavet et plot af farten over tid, når farten var indstillet til 15 km/t, set nedenfor.



Figur 5.49: PID Controller step

Hvis kurven fittes til dette datasæt ville den langt fra være påen. Det lykkedes derfor ikke at bestemme PID-værdierne på en fornuftig måde, andet end at prøve sig frem. Trods dette er der designet og implementeret en fuldt ud fungerende PI-controller med mulighed for et d-led, høj præcision, og lav settle time.

5.3.8 SPI

Kommunikationen mellem PSoC og Raspberry Pi foregår over SPI. Selve SPI-protokollen er beskrevet mere uddybende i det dertilhørende afsnit. Dette afsnit vil se på hvordan PSoC'en behandler sine to SPI-buffere. SPI er interrupt-baseret, og det er derfor oplagt at se på interruptet først. Nedenfor ses de mest interessante linjer fra SPI-interruptet.

```

1 ...
2 j = SPI_S_SpiUartGetRxBufferSize();
3 ...
4 for(i=0u; i<7; i++)
5 {
6 bufIn[n+i] = SPI_S_SpiUartReadRxData();
7 }
8 bufferGetData(bufIn);
9
10 if (j>0)
11 {
12 for(k=0u; k<7; k++)
13 {
14 SPI_S_SpiUartWriteTxData(bufOut[n+k]);
15 }
```

```

16 }
17
18 setSpd = kbuf [(key)%7];
19 ...
20 setServo = kbuf [(key+1)%7];
21 Run = kbuf [(key+2)%7];

```

Som det ses lægges syv bytes data, fra Raspberry Pi over i modtager-bufferen, og så lægges PSoC'ens data i afsender-bufferen til Raspberry Pi. Hvor hurtigt SPI-exchange foregår er bestemt af Raspberry Pi, og vil blive forklaret i det afsnit. Det eneste uventede i interruptet er kaldet til bufferGetData-funktionen. Denne funktion er nødvendig, fordi det ikke er sikkert hvilken byte der først bliver lagt i Rx-bufferen. Derfor er det svært at behandle dataen som ønsket. Løsningen er, at der skal afsendes seks data-bytes (og en check-byte), men at der kun modtages tre data-bytes (og fire unødvendige). Disse fire "unødvendige"bytes kan bruges som keys til at navigere et ellers uordnet array, og det er netop det bufferGetData-funktionen gør. Ved at sætte Raspberry Pi's SPI-output bytes til [0]-Ønsket Fart [1]-Servo-vinkel [2]-Start/stop mode [3]-128 [4]-2 [5]-128 [6]-4, kan dette gøres med nogle nestede conditionals, set i koden nedenfor.

```

1 void bufferGetData(int8 kbuf [])
2 {
3     int m = 0;
4     int key = 0;
5
6     for(m=0;m<7;m++)
7     {
8         if (kbuf [m] == -128)
9         {
10             switch(kbuf [(m+1)%7])
11             {
12                 case -2:
13                     key = m+4;
14                     break;
15
16                 default:
17                     key = m+2;
18                     break;
19             }
20         }
21     }
22
23     setSpd = kbuf [(key)%7];
24
25     setServo = kbuf [(key+1)%7];
26
27     Run = kbuf [(key+2)%7];
28 }

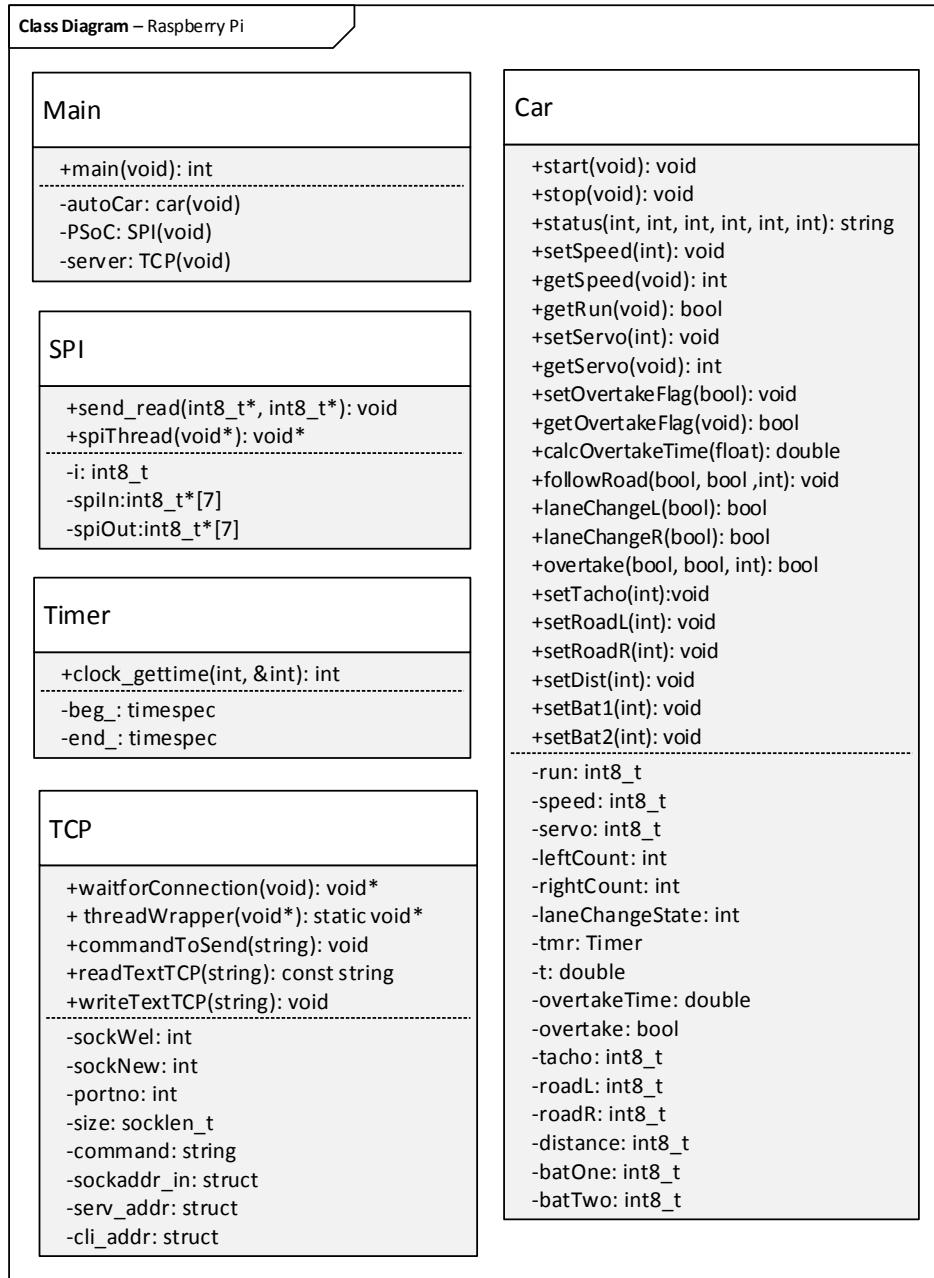
```

Yderst i funktionen køres arrayet igennem, og der ledes efter den første "-128" der ligger i SPI-arrayet. Når den er fundet, checkes den næste byte. Hvis den næste byte er "byte 7", som ikke eksisterer, sørger modulus for at arrayet behandles cirkulært i denne funktion. I hvert fald i den positive retning. Ud fra indholdet af byten efter den første "-128" kan positionen på Ønsket Fart findes, og dermed er de to næste hhv. Servo-vinkel og Start/stop-mode. Indholdet af disse tre bytes i arrayet, skrives til de tilhørende globale variable, der er defineret som extern i SPI.h-filen. På denne måde lykkes det, at finde dataene i et ellers uordnet array.

5.3.9 Print

Printfunktionen er brugt til at skrive RPM og hastighed ud i PuTTy-terminalen. Den fungerer over UART.

5.3.10 Raspberry Pi



Figur 5.50: Klassediagram for Raspberry Pi-software

SPI	send_read()
Prototype	void send_read(int8_t*, int8_t*);
Kort beskrivelse af funktionalitet	Sender og Læser SPI-data
SPI	spiThread()
Prototype	void* spiThread(void*)
Kort beskrivelse af funktionalitet	Opretter en SPI-thread
Timer	clock_gettime()
Prototype	int clock_gettime(int, &int)
Kort beskrivelse af funktionalitet	Timer-funktion
TCP	*waitForConnection()
Prototype	void *waitForConnection(void)
Kort beskrivelse af funktionalitet	Venter på forbindelse
TCP	threadWrapper()
Prototype	static void* threadWrapper(void*)
Kort beskrivelse af funktionalitet	ThreadWrapper
TCP	commandToSend()
Prototype	void commandToSend(string)
Kort beskrivelse af funktionalitet	Sender en kommando
TCP	readTextTCP()
Prototype	const string readTextTCP(string)
Kort beskrivelse af funktionalitet	Læser TCP besked
TCP	writeTextTCP()
Prototype	void writeTextTCP(string)
Kort beskrivelse af funktionalitet	Skriver TCP besked
TCP	start()
Prototype	void start(void);
Kort beskrivelse af funktionalitet	Sætter Run-flaget til 1
Car	stop()
Prototype	void stop(void);
Kort beskrivelse af funktionalitet	Sætter Run-flaget til 0
Car	status()
Prototype	string status(int, int, int, int, int);
Kort beskrivelse af funktionalitet	Læser status fra SPI og sender til GUI
Car	setSpeed()
Prototype	void setSpeed(int);
Kort beskrivelse af funktionalitet	Set-funktion til speed i car-objektet
Car	getSpeed()
Prototype	void getSpeed(int);
Kort beskrivelse af funktionalitet	Get-funktion til speed i car-objektet

Car	getRun()
Prototype	bool getRun(void);
Kort beskrivelse af funktionalitet	Set-funktion til run i car-objektet
Car	setServo()
Prototype	void setServo(int);
Kort beskrivelse af funktionalitet	Set-funktion til servo i car-objektet
Car	getServo()
Prototype	int getServo(void);
Kort beskrivelse af funktionalitet	Get-funktion til servo i car-objektet
Car	setOvertakeFlag()
Prototype	void setOvertakeFlag(bool);
Kort beskrivelse af funktionalitet	Set-funktion til overtake-flag i car-objektet
Car	getOvertakeFlag()
Prototype	bool getOvertakeFlag(void);
Kort beskrivelse af funktionalitet	Get-funktion til overtake-flag i car-objektet
Car	calcOvertakeTime()
Prototype	double calcOvertakeTime(float);
Kort beskrivelse af funktionalitet	Beregner tiden det tager at overhale objekt
Car	followRoad()
Prototype	void followRoad(bool, bool ,int);
Kort beskrivelse af funktionalitet	Følger vejbane
Car	laneChangeL()
Prototype	bool laneChangeL(bool);
Kort beskrivelse af funktionalitet	Skifter vejbane til venstre
Car	laneChangeR()
Prototype	bool laneChangeR(bool);
Kort beskrivelse af funktionalitet	Skifter vejbane til højre
Car	overtake()
Prototype	bool overtake(bool, bool ,int);
Kort beskrivelse af funktionalitet	Overhaler objekt
Car	setTacho()
Prototype	setTacho(int);
Kort beskrivelse af funktionalitet	Set-funktion til tachometer i car-objektet
Car	setRoadL()
Prototype	void setRoadL(int);
Kort beskrivelse af funktionalitet	Set-funktion til venstre vejbanesensor i car-objektet

Car	setRoadR()
Prototype	void setRoadR(int);
Kort beskrivelse af funktionalitet	Set-funktion til højre vejbanesensor i car-objektet
Car	setDist()
Prototype	void setDist(int);
Kort beskrivelse af funktionalitet	Set-funktion til sonar-afstand i car-objektet
Car	setBat1()
Prototype	void setBat1(int);
Kort beskrivelse af funktionalitet	Set-funktion til Motor-batteri i car-objektet
Car	setBat2()
Prototype	void setBat2(int);
Kort beskrivelse af funktionalitet	Set-funktion til Logik-batteri i car-objektet

I dette afsnit forklares de klasser som er lavet til Raspberry Pi'en. Dette inkludere funktionsbeskrivelser, klassediagram, og gennemgang af koden.

5.3.10.1 Overordnet Raspberry Pi arkitektur

5.3.10.2 SPI klassen

SPI klassens funktion er at kommunikere med PSoC'en og overholde SPI protokollen under kommunikationen.

SPI klassen er relativt simpel da den benytter bcm2835 biblioteket[33], som har en masse funktionalitet. Det kan styre Raspberry Pi'ens GPIO ben, hvilket betyder den også kan benytte SPI benene.

Constructoren bruges til at sætte de indstillinger SPI forbindelsen skal have. Det er bit ordenen, clock-divider og hvilken chip-select der bruges.

```

1 SPI::SPI()
2 {
3     if (!bcm2835_init())
4     {
5         printf("bcm2835_init failed. Are you running as root??\n");
6     }
7
8     if (!bcm2835_spi_begin())
9     {
10        printf("bcm2835_spi_begin failedg. Are you running as root??\n");
11    }
12    bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);
13    bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);
14    bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_256);

```

```

15     bcm2835_spi_chipSelect(BCM2835_SPI_CS0);
16     bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, LOW);
17 }
```

Den anden funktion i SPI klassen er `send_read()` som tager to `int8_t` arrays, som indeholder data der skal sendes og modtages.

```

1 void SPI::send_read(int8_t * spiOut_, int8_t * spiIn_)
2 {
3
4     for(int8_t i = 0; i < 7; i++)
5     {
6         spiIn_[i] = bcm2835_spi_transfer(spiOut_[i]);
7     }
8 }
```

5.3.10.3 TCP klassen

TCP klassens funktion er at oprette og kontrollere forbindelsen mellem GUI og Raspberry Pi. Den skal ligeledes kontrollere de beskeder der kommer fra GUI. Det betyder at den kalder de funktioner som sætter værdierne for bilens hastighed osv.

Constructoren i TCP klassen sørger for at oprette de rigtige sockets så der er en velkomst-socket til tilslutning og en socket som tildeles klienten der tilslutter. Desuden sættes socket-indstillinger som fx. at port 9000 kan genbruges, så der ikke skal vælges en ny socket under næste opstart.

Funktionen `void *TCP::waitForConnection()` er ansvarlig for at den tilsluttene klient bliver tildelt sin socket. Når klient er tilsluttet, lytter den efter beskeder, og reagerer med et svar og nogle funktionskald afhængig af den modtagne besked. Den modtagne besked skal være i overensstemmelse med TCP protokollen. Hvis der som eksempel modtages "01QstatusW" skal der kaldes en funktion som returnerer en streng. Strengen indeholder de informationer som forventes af protokollen, og strengen skrives så til klienten. Dette opnås ved at kalde funktionen `writeTextTCP()`. Den tager en streng og sender den til klienten. Derudover er der en funktion til at modtage fra klienten som hedder `readTextTCP` som returnerer beskeden fra klienten.

```

1
2 void *TCP::waitForConnection(void){
3     // Listening
4     cout << "Looking for clients..." << endl;
5     listen(sockWel, 5);
6     size = sizeof(cli_addr);
7
8     sockNew = accept(sockWel, (struct sockaddr *) &cli_addr, &size);
9     if (sockNew < 0)
10    {
11        cout << "Error on accept" << endl;
12    }
13 }
```

```

14     cout << "Client Connected" << endl;
15
16     while(1)
17     {
18         listen(sockNew, 5);
19         command = readTextTCP(command);
20         cout<<command<<endl;
21         if(command.compare("01QstatusW")==0)
22         {
23             string statusStr = autoCar.status(spiIn[0],
24                                             spiIn[1], spiIn[2], spiIn[3], spiIn[4], spiIn[5]);
25             cout << "status: " << statusStr << endl;
26             writeTextTCP(statusStr);
27         }
28         else if(command.compare("02QoverhalW")==0)
29         {
30             writeTextTCP("12QoverhalingQ0003W");
31         }
32         else if(command.substr(0,13).compare("03QhastighedQ")==0)
33         {
34             int speedInt = atoi(command.substr(13,3).c_str());
35
36             if(speedInt > 26){
37                 autoCar.setSpeed(26);
38                 writeTextTCP("13QhastighedQ2002W");
39             }else if(speedInt < -26){
40                 autoCar.setSpeed(-26);
41                 writeTextTCP("13QhastighedQ2002W");
42             }else{
43                 autoCar.setSpeed(speedInt);
44                 writeTextTCP("13QhastighedQ2001W");
45             }
46         }
47         else if(command.compare("04QstartW")==0)
48         {
49             autoCar.start();
50             cout << autoCar.getRun() << endl;
51             writeTextTCP("14QstartQ111119999W");//FIXME
52         }
53         else if(command.compare("05QstopW")==0)
54         {
55             autoCar.stop();
56             writeTextTCP("15QstopQ3001W");
57         }
58         else if(command.compare("06QtestW")==0)

```

```

59         {
60             writeTextTCP("16QtestW");
61         }
62
63         command.clear();
64     }
65     return 0;
66 }
```

5.3.10.4 Timer klassen

Timer klassen bruges til at bestemme tiden imellem to brugerbestemte tidspunkter. Denne funktionalitet bruges da der ikke ønskes delay i vores kode. Koden er afledt af et eksempel [34].

```

1 class Timer
2 {
3     public:
4         Timer() { clock_gettime(CLOCK_REALTIME, &beg_); }
5
6         double elapsed() {
7             clock_gettime(CLOCK_REALTIME, &end_);
8             return end_.tv_sec - beg_.tv_sec +
9                 (end_.tv_nsec - beg_.tv_nsec) / 1000000000.0;
10        }
11
12        void reset() { clock_gettime(CLOCK_REALTIME, &beg_); }
13
14     private:
15         timespec beg_, end_;
16 }
```

5.3.10.5 Car klassen

Car klassen udgør bilen, i den form at den indeholder alle de variable som bruges til at styrer bilen. Der er også de funktioner som implementerer bilens faktiske funktionalitet, som at den kan følge en vej. Den indeholder tre vigtige variable; run, speed og servo. De bestemmer hhv. om bilen er tændt, hvor hurtigt bilen skal køre, og hvordan servoen skal dreje. Den har også variable for den data der kommer fra sensorerne.

De fleste af de funktioner som Car indeholder er set- og get-funktioner. Disse funktioner sikrer at der ikke skrives noget forkert til de forskellige variable, og at der ikke læses hvor der ikke må.

Funktionen `void followRoad(bool infraLeft_, bool infraRight_, int distance_)`, bruges til at udregne hvor meget servoen skal dreje for at følge vejen. Dette afhænger af hvilken vejbanesensor der er aktiv. Funktionen reagerer også hvis afstanden til den forankørende er for lav. I denne implementering stopper vores bil hvis der er et objekt inden for en fastsat afstand. Dette kan i fremtiden implementeres, så bilen justerer sin hastig-

hed, og holder sig i en afstand af bilen foran. ro

```

1 void car::followRoad(bool infraLeft_, bool infraRight_, int distance_){
2     if(distance_ < MIN_DIST){
3         run = 0;
4         speed = 0;
5     }
6
7     if(infraLeft_ && !infraRight_){
8         if(leftCount > -100 + TURN_CONST){
9             leftCount -= TURN_CONST;
10            servo = leftCount;
11        }
12        servo = leftCount * TURN_CONST;
13    }else if(infraRight_ && !infraLeft_){
14        if(rightCount < 100 - TURN_CONST){
15            rightCount += TURN_CONST;
16            servo = rightCount;
17        }
18    }else{
19        rightCount = 0;
20        leftCount = 0;
21        servo = 0;
22    }
23 }
```

Funktionen `bool overtake(bool infraLeft_, bool infraRight_, int distance_)`, implementere den funktionalitet som muliggøre at bilen kan overhale den forankørende eller et objekt, dog kræves det i denne implementering at bilen eller objektet foran holder stille. Funktionen gør brug af flere hjælpefunktioner; `laneChangeL`, `laneChangeR` og `calcOvertakeTime`. Funktionen kører sekventielt igennem en række trin, først skifter den over i den venstre bane, det antages at der er plads i den venstre bane, når den højre vejbanesensor registrerer den midterste vejbanestribe, begynder bilen at følge den venstre vejbane i den tid det er udregnet det vil tage at overhale. Når tiden er gået, drejer bilen til højre ind i den højre vejbane. Ligesom tidligere er den i højre vejbane når den venstre vejbanesensor er blevet aktiveret.

```

1 bool car::overtake(bool infraLeft_, bool infraRight_, int distance_){
2     run = 1;
3     if(speed == 0){
4         speed = OVERTAKE_SPEED;
5     }
6     switch(laneChangeState){
7         case 0:
8             if(laneChangeL(infraRight_)){
9                 laneChangeState = 1;
10                overtakeTime = calcOvertakeTime(CAR_LENGTH);
11                tmr.reset();
12            }
13        }else if(laneChangeState == 1){
14            if(laneChangeR(infraLeft_)){
15                laneChangeState = 2;
16                overtakeTime = calcOvertakeTime(CAR_LENGTH);
17                tmr.reset();
18            }
19        }else if(laneChangeState == 2){
20            if(tmr.read() > overtakeTime){
21                turnRight();
22            }
23        }
24    }
25 }
```

```

12         }
13         break;
14     case 1:
15         if(tmr.elapsed() > overtakeTime){
16             laneChangeState = 2;
17         }
18         followRoad(infraLeft_, infraRight_, distance_);
19         break;
20
21     case 2:
22         if(laneChangeR(infraLeft_)){
23             laneChangeState = 0;
24             return 1;
25         }
26         break;
27     }
28     return 0;
29 }
```

5.3.10.6 Main

Main koden er som altid det sted hvor alle klasserne bliver kædet sammen. Der oprettes to tråde som kører parallelt, det er SPI og TCP kommunikationen. SPI tråden er også ansvarlig for at kalde de forskellige funktioner som `followRoad()`, og `overtake()` hvis et overhal flag er sat højt af TCP-tråden. Disse funktioner kaldes hver gang der bliver sendt information over SPI forbindelsen. Der sendes over SPI med en frekvens på 10Hz, dette sørger for at PSoC'en ikke belastes for meget. Det risikeres at den ikke kan udføre andre interrupts, eksempelvis reguleringen.

```

1 #include <pthread.h>
2 #include "global.h"
3
4 SPI PSoC = SPI();
5
6 int8_t spiOut[7] = {0,0,0,-128,-2,128,4};
7 int8_t spiIn[7];
8
9 car autoCar = car();
10 TCP server = TCP(9000);
11
12 void* spiThread(void* data){
13     while(1){
14
15         if( autoCar.getRun() == true)
16         {
17             if( autoCar.getOvertakeFlag() == false)
18                 {
```

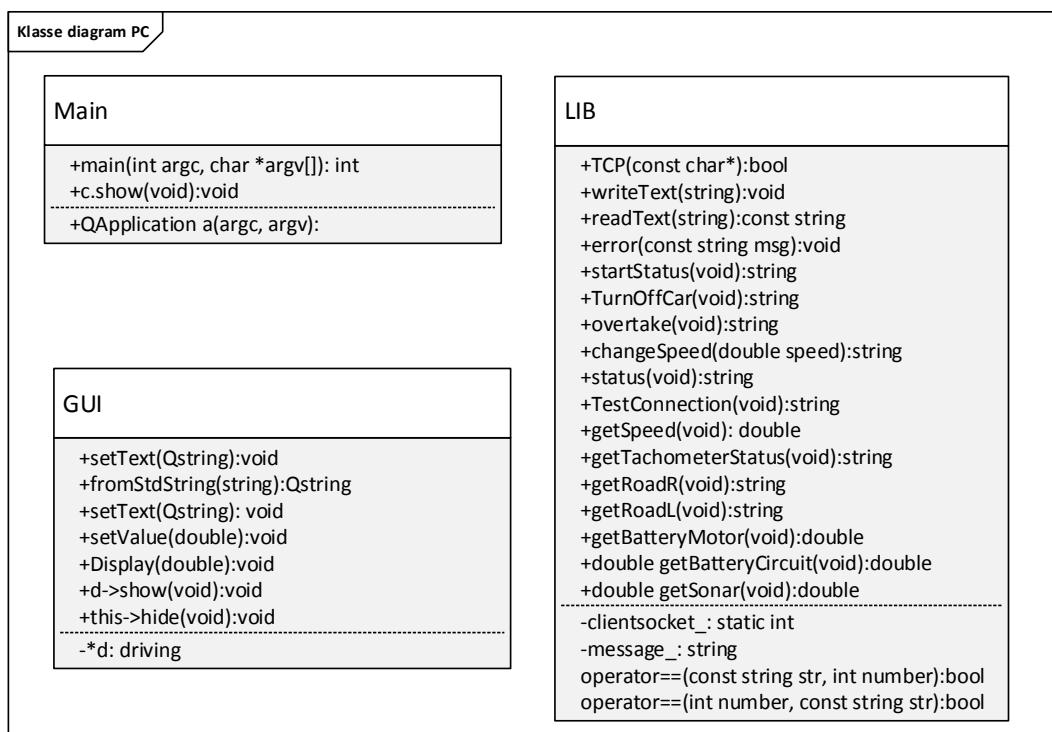
```
19             autoCar.followRoad(spiIn[1],spiIn[2],spiIn[3]);
20         }else if( autoCar.getOvertakeFlag() == true)
21     {
22         if(autoCar.overtake(spiIn[1],spiIn[2],spiIn[3]))
23     {
24         autoCar.setOvertakeFlag(false);
25         server.writeTextTCP("12QoverhalingQ0001W");
26     }
27 }
28
29
30     spiOut[0] = autoCar.getSpeed();
31     spiOut[1] = autoCar.getServo();
32     spiOut[2] = autoCar.getRun();
33
34     PSoC.send_read(spiOut,spiIn);
35
36     autoCar.setTacho(spiIn[1]);
37     autoCar.setRoadL(spiIn[2]);
38     autoCar.setRoadR(spiIn[3]);
39     autoCar.setDist(spiIn[4]);
40     autoCar.setBat1(spiIn[5]);
41     autoCar.setBat2(spiIn[6]);
42
43     usleep(100000);
44 }
45 }
46
47 int main(){
48
49     pthread_attr_t attr;
50     pthread_attr_init(&attr);
51     pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
52
53     pthread_t TCPserver;
54     pthread_t SPImaster;
55     int err;
56     string besked;
57
58     err = pthread_create(&TCPserver,&attr,&TCP::threadWrapper,&server);
59     if(err !=0){
60         cout << "Thread create error TCPserver!" << endl;
61     }
62
63     err = pthread_create(&SPImaster,&attr,spiThread,NULL);
64     if(err !=0){
```

```

65         cout << "Thread create error TCPserver!" << endl;
66     }
67
68     string tmp;
69     while(1);
70
71     return 0;
72 }
```

5.3.11 PC-software

Herunder beskrives kode til PC'en. Først kommer der et klassediagram og funktionsbeskrivelser, efterfølgende kommer der beskrivelse af udvikling af TCP og til sidst GUI.



Figur 5.51: Klasse diagram

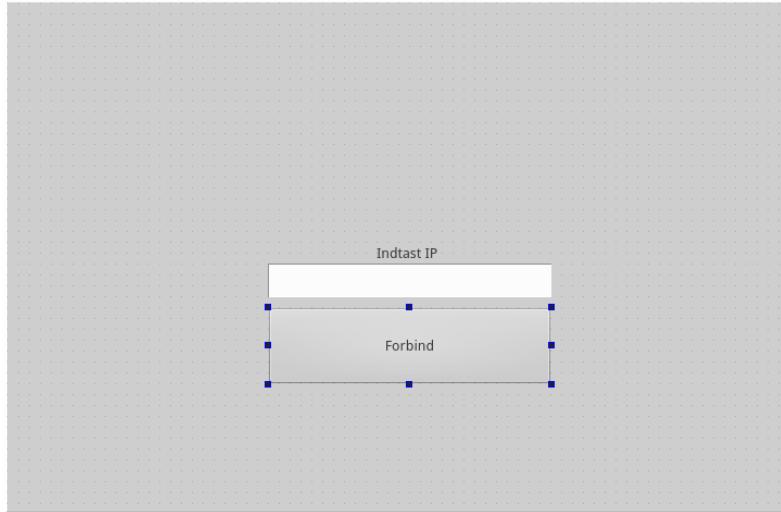
GUI	setText
Prototype	void setText(QString)
Kort beskrivelse af funktionalitet	sætter text på label
GUI	fromStdString
Prototype	QString fromStdString(string)
Kort beskrivelse af funktionalitet	konvertere string to QString

GUI	Display
Prototype	void setValue(double)
Kort beskrivelse af funktionalitet	sætter værdigen på et process bar
GUI	setValue
Prototype	void Display(double)
Kort beskrivelse af funktionalitet	sætter værdigen i et LCD display
GUI	show
Prototype	void d->show()
Kort beskrivelse af funktionalitet	viser et nyt vindue
GUI	hide
Prototype	void this->hide()
Kort beskrivelse af funktionalitet	skjuler det pågældende vindue
GUI	driving *d
Prototype	void driving *d
Kort beskrivelse af funktionalitet	opretter et nye vindue pointer objekt
main	main
Prototype	int main(int argc, char *argv[])
Kort beskrivelse af funktionalitet	main funktionen
main	c.show
Prototype	void c.show()
Kort beskrivelse af funktionalitet	viser et vindue
main	QApplication
Prototype	QApplication a(argc, argv)
Kort beskrivelse af funktionalitet	opretter et QT objekt
LIB	TCP
Prototype	bool TCP(const char*)
Kort beskrivelse af funktionalitet	opretter TCP forbindelse
LIB	writeText
Prototype	void writeText(string)
Kort beskrivelse af funktionalitet	skriver over TCP
LIB	readText
Prototype	const string readText(string)
Kort beskrivelse af funktionalitet	læser over TCP
LIB	error
Prototype	void error(const string msg)
Kort beskrivelse af funktionalitet	returnere fejl
LIB	startStatus
Prototype	string startStatus(void)
Kort beskrivelse af funktionalitet	anmode om start status

LIB	TurnOffCar
Prototype	string TurnOffCar(void)
Kort beskrivelse af funktionalitet	anmode om stop status
LIB	overtake
Prototype	string overtake(void)
Kort beskrivelse af funktionalitet	anmode om at overhale
LIB	changeSpeed
Prototype	string changeSpeed(double speed)
Kort beskrivelse af funktionalitet	anmode om at skifte hastigheden
LIB	status
Prototype	string status(void)
Kort beskrivelse af funktionalitet	anmode om status
LIB	TestConnection
Prototype	string TestConnection(void)
Kort beskrivelse af funktionalitet	test om der er forbindelse via TCP
LIB	getSpeed
Prototype	double getSpeed(void)
Kort beskrivelse af funktionalitet	hent hastighed fra hukommelsen
LIB	getTachometerStatus
Prototype	string getTachometerStatus (void)
Kort beskrivelse af funktionalitet	hent tachometer status fra hukommelsen
LIB	getRoadR
Prototype	string getRoadR (void)
Kort beskrivelse af funktionalitet	hent højre vejbane sensor status fra hukommelsen
LIB	getRoadL
Prototype	string getRoadL (void)
Kort beskrivelse af funktionalitet	hent venstre vejbane sensor status fra hukommelsen
LIB	getBatteryMotor
Prototype	double getBatteryMotor (void)
Kort beskrivelse af funktionalitet	hent motor batteri niveau fra hukommelsen
LIB	getBatteryCircuit
Prototype	double getBatteryCircuit (void)
Kort beskrivelse af funktionalitet	hent kredsløbs batteri niveau fra hukommelsen
LIB	getSonar
Prototype	double getSonar (void)
Kort beskrivelse af funktionalitet	hent sonar afstand fra hukommelsen

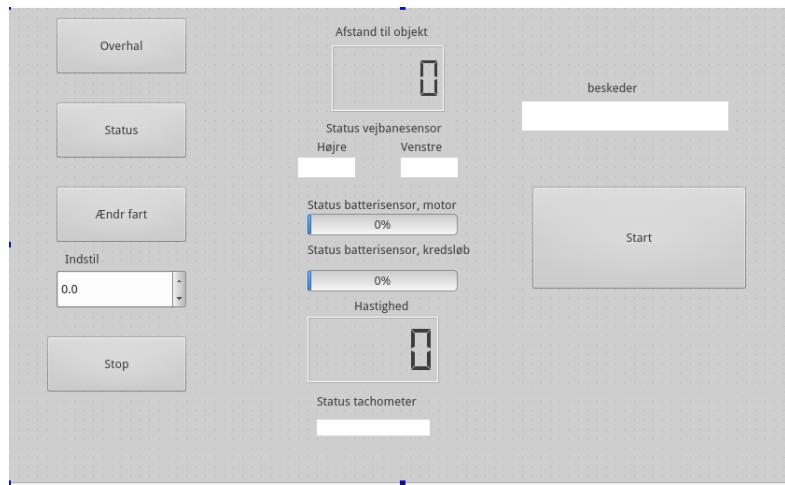
5.3.11.1 GUI

Der er lavet et GUI via QT creator. Det visuelle er designet med drag and drop, mens alt funktionaliteten er kodet i c++. Figur 5.52 og figur 5.53 viser de to vinduer der er i GUI'et.



Figur 5.52: Forbindelsesvindue i GUI

Figur 5.52 viser det vindue der kommer, når GUI'et startes. Her indtastes Raspberry Pi'en IP-adresse. Er IP'en korrekt, vises vinduet på figur 5.53. Hvis IP'en ikke er korrekt, udskrives en fejlbesked.



Figur 5.53: Kontrolvindue i GUI

Figur 5.53 viser kontrolvinduet, som bliver brugt til styringen af Raspberry Pi'en. I dette vindue er det muligt at starte og stoppe bilen samt ændre hastigheden. Desuden kan status på bilens sensorer udskrives. Det er kun muligt at ændre i hastighedsboksen. Her kan tastes tal i intervallet -13 til 13 i step af 0,5. Derefter trykkes på "Ændr fart", som vil hente dataen derfra og gemme værdien i en double. Når det er gemt som double, kan det sendes over TCP via `TCP.changeSpeed()`-funktionen. Denne funktion laver det om til

en streng, der skal sendes til Raspberry Pi.

```

1 LIB TCP;
2 string message;
3 double speed = ui->speedBox-> value();
4 message = TCP.changeSpeed(speed);

```

Herunder ses koden til "Overhal" knappen. Først oprettes et TCP objekt og en string, som skal bruges til at hente data fra overhalingsfunktionen. TCP.overtake() sender koden for overhaling ifølge TCP protokollen, og returnerer en streng med bekræftelsesmeddelsen. Denne udskrives på messageLabel. Da alle labels i QT skal bruge en QString for at udskrive, skal denne først konverteres. QString er en string-standard som bruges i QT. Den gemmer datene i Unicode som 16bit og giver mange behandlingsmuligheder. GUI'et er den eneste del, der er udviklet i QT og derfor er det kun når der skrives til labels, at QString bliver brugt.

```

1 void driving::on_pushButton_2_clicked()
2 {
3     LIB TCP;
4     string message;
5     message = TCP.overtake();
6     ui->messageLabel -> setText(QString::fromStdString(message));
7 }

```

5.3.11.2 TCP

TCP er en kommunikationsstandard, som ofte bruges i forbindelse med kommunikation over internettet. TCP er en pålidelig kommunikationsprotokol, da den indeholder flow-control, sekvensnumre, acknowledgements og timers.

TCP er en forbindelsesorienteret protokol, dvs. at det er kommunikation mellem to enheder, en server og en client. Før der bliver sendt data over forbindelsen udveksler server og client et håndtryk som indeholder IP og portnummer. Forbindelsen er full duplex, dvs. at der kan sendes data begge veje på samme tid.[35]

Det er valgt at den trådløse kommunikation skal være TCP, da den har været en del af undervisning i IKN, og TCP indeholder pålidelig kommunikation.

5.3.11.3 TCP PC

Der er implementeret TCP via socket, med de funktioner der er udgivet i det tilhørende bibliotek <sys/socket.h>. Derudover bruges der structs, som er defineret i <netinet/in.h>. Forklaring til socket-implementeringen følger herunder.

```

1
2     clientsocket = socket(AF_INET, SOCK_STREAM, 0);
3     if (clientsocket < 0)
4     {
5         error("Error opening socket");
6         return false;
7     }

```

Først gemmes socketen i en int. Denne består af AF_INET der er adressens type, og SOCK_STREAM som angiver at dataene sendes som en strøm. I modsætning til SOCK_STREAM kan der bruges SOCK_DGRAM, hvor dataen vil blive sendt i pakker. Der testes om der er oprettet en socket. I tilfælde af at den ikke oprettes vil der blive gemt 0 i clientsocket.

```

1 server = gethostbyname(IP);
2
3 if (server == NULL)
4 {
5 error("ERROR, no such host");
6 return false;
7 }
```

gethostbyname er en socket-funktion, som ud fra en IP-adresse henter en række data på serveren og gemmer det i en struct. I tilfælde af at der ikke findes en server med den ønskede IP-adresse, returneres der NULL fra gethostbyname. Dette bruges til fejlkontrol på om der er en server.

```

1 if (connect(clientsocket,(struct sockaddr *) &serv_addr,
2             sizeof(serv_addr)) == -1)
3 {
4     error("Error establishing connection");
5     return false;
6 }
7 clientsocket_=clientsocket;
```

Til sidst køres der en kontrol på forbindelsen med connect som returnerer 0 hvis forbindelsen er oprettet og -1 hvis den ikke er.

```

1 void writeText(string line);
2 const string readText(string inText);
3 void error(const string msg);
```

De ovenstående tre funktioner er en modificeret version af den kode der blev givet i forbindelse med øvelse 8 IKN³. writeText har til opgave at sende over TCP-forbindelsen 0-termineret, mens read har til opgave at modtage de data der kommer over TCP-forbindelsen. error bliver brugt til at læse de fejlbeskeder, der bliver skrevet i perror, og returnere dem. perror er en funktion som returnerer værdien af errno, som er en variabel der fortæller noget om fejsituationen.

```

1 string status();
2 double getSpeed();
```

De resterende funktioner i TCP client er get- eller send-funktioner. send funktionerne sender en kode i overensstemmelse med protokollen og venter derefter på svar fra serveren. Når de har modtaget svar, gemmer de det i string-attributten message_. get-funktionerne henter den data stump der hører til den enkelte get-funktion. Dette sker

³Denne kode er skrevet af Lars Mortensen, IHA, ifm. øvelse 8, I4IKN

```

Terminal - root@ubuntu:~/Desktop/Test TCP/build/host
File Edit View Terminal Tabs Help
root@ubuntu:~/Desktop/Test TCP/build/host# ./prog 10.0.0.1
intast besked :
test
vent på retur besked :
test
intast besked :
abc
vent på retur besked :
abc
intast besked :
123456
vent på retur besked :
123456
intast besked :
+
vent på retur besked :
+
intast besked :
...
vent på retur besked :
...
intast besked :

Terminal - root@ubuntu:~/Desktop/TCPServer/build/host
File Edit View Terminal Tabs Help
root@ubuntu:~/Desktop/TCPServer/build/host# ./prog
Setting up TCP server
port: 9999
Socket opened
sock opt set
Binded
Looking for clients...
l
l
Client Connected
test
abc
123456
+
....
```

Figur 5.54: Test af TCP Client

ved at lave en substring af message_, og så returnere dataen i den format der er behov for.

5.3.11.4 Modul test

For at teste om TCP clienten virker, sættes en TCP server op som Echo-server. Echo-serveren returnerer alle værdier som sendes. Dermed kan både write- og read-funktionen testes samtidig.

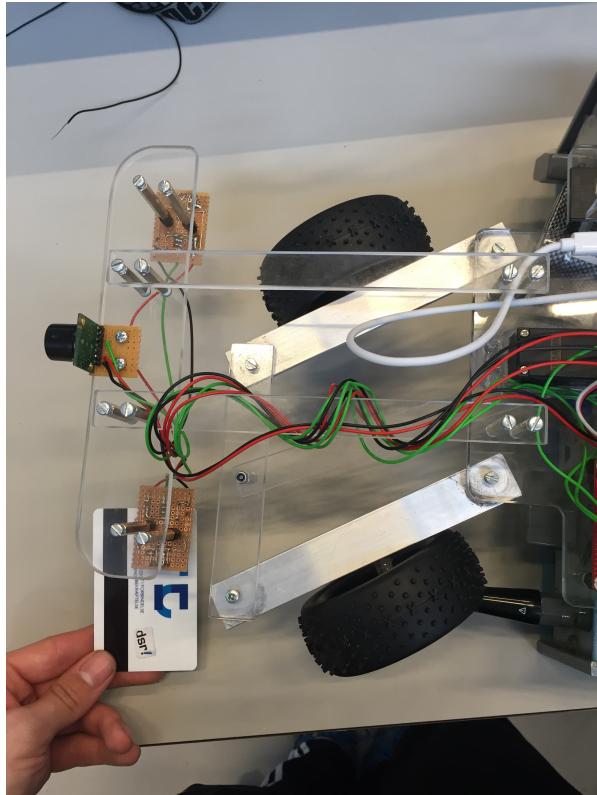
På figur 5.54 ses klienten til højre og Echo-serveren til venstre. Clienten udskriver både det den modtager og sender. På serveren udskrives der kun 1 gang da det, er samme data som sendes og modtages. Det kan ses at klienten både sender og modtager det forventede. Test af TCP i følge protokol kan ses under integrationstest, hvor det testes sammen med Raspberry Pi.

5.4 Integrationstest

5.4.1 Sensorer, Servo og PSoC

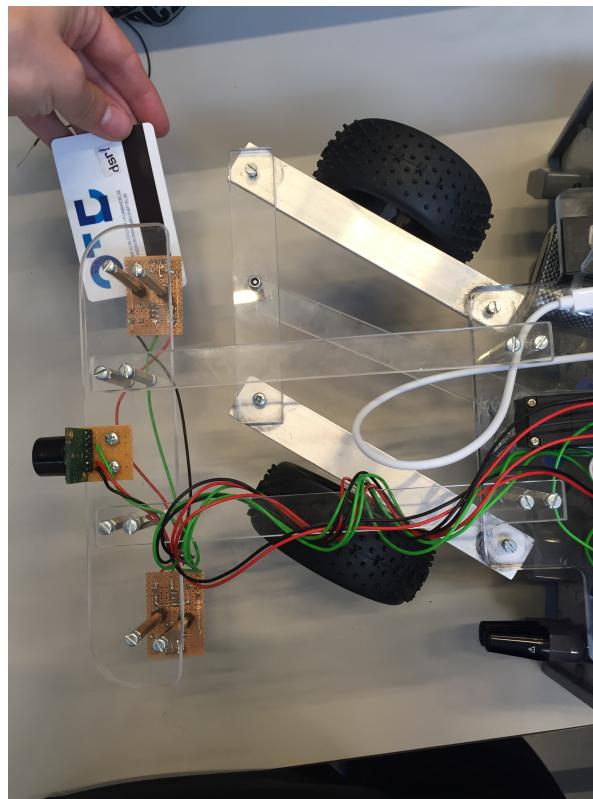
Denne integrationstest indeholder testen af grænsefladen mellem sensorene, servo'en og PSoC'en. For at udfører denne test er der skrevet et testprogram i PSoC'en. Dette program drejer servo'en helt til venstre hvis den venstre vejbanesensor registrer noget, og modsat med den højre. Derudover termineres programmet hvis sonaren registrer noget inden for en given grænse.

På figur 5.55 testes den venstre vejbane sensor med servo'en og PSoC'en. Det ses at når sensoren registrer noget hvidt, drejer servo'en helt til venstre.



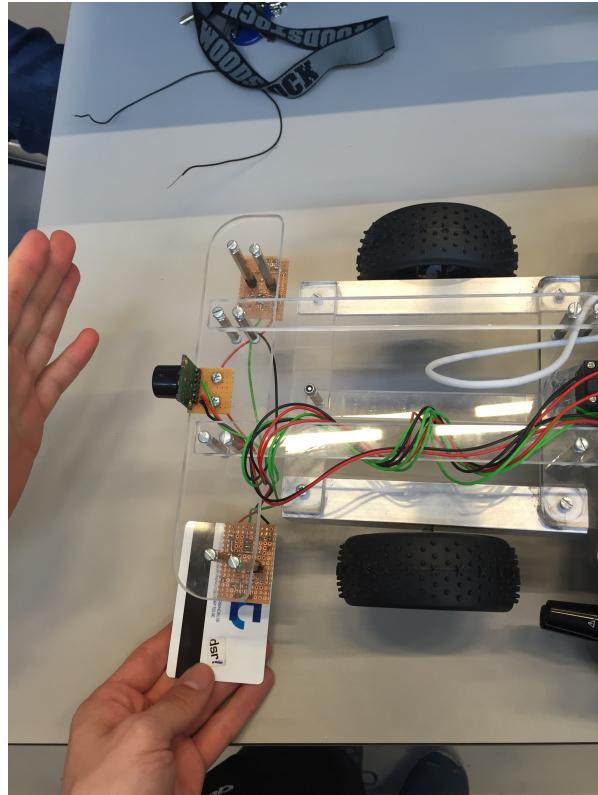
Figur 5.55: Integrationstest af venstre vejbanesensor og servo

På figur 5.56 ses integrationstesten for den højre vejbanesensor, servoen og PSoC'en. Modsat før drejer servoen nu til højre når sensoren registrer noget hvidt.



Figur 5.56: Integrationstest af højre vejbanesensor og servo

På figur 5.57 ses integrationstesten af sonaren og PSoC'en. Her termineres programmet og servoen drejer til midtpunktet, når sonaren registrer et objekt. Det ses at selvom den ene vejbanesensor aktiveres sker der ikke noget med servoen.

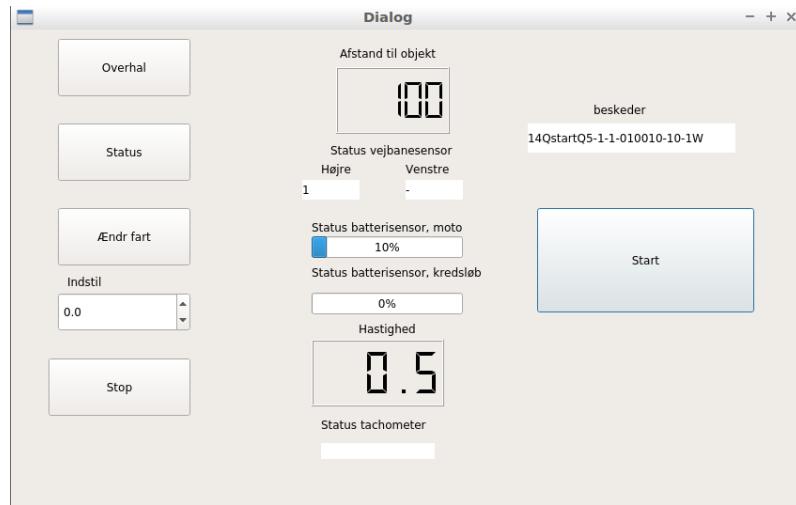


Figur 5.57: Integrationstest af sonaren og PSoC'en

5.4.2 GUI, Raspberry Pi, PSoC

Denne test er lavet for at teste kommunikationen imellem de tre platforme, det vil sige TCP og SPI. Yderligere testes der også hvorvidt PSoC'en bliver styret på den måde det forventes ud fra designet. For at teste sendes der en kommando fra GUI, såsom Start og så kigges der på Raspberry Pi's terminal og ses om det rigtige er blevet sendt, PSoC'en indstilles til debugging mode og der kigges på den buffer som SPI skriver til. Desuden tjekkes det også om den rigtige return-besked sendes tilbage til GUI'et.

Under test af Start-knappen forventes det at Run-variablen på PSoC'en bliver sat til 1. Desuden forventes det også at der printes en status for hver sensor på GUI'et.



Figur 5.58: GUI efter start er valgt

Efter start er valgt modtages en besked fra Raspberry Pi. Sensordataen på billedet er ikke korrekt, da de ikke var tilsluttet under testen.

```

sock SO_KEEPALIVE set
Binded
Looking for clients...
^Cpi@raspberrypi:~$ sudo ./Car/bin/host/prog
Setting up TCP server
port: 9000
Socket opened
sock SO_REUSEADDR set
Error on keepalive SO_OPT
sock SO_KEEPALIVE set
Binded
Looking for clients...
Client Connected
05QstopW
Wrote to TCP: 15QstopQ3001W
04QstartW
Tacho: -1
RoadL: -1
RoadR: -1
Distance: -1
Bat1: -1
Bat2: -1
Wrote to TCP: 14QstartQ5-1-1-010010-10-1W

```

Figur 5.59: Raspberry Pi's terminaludskrift

Raspberry Pi'en udskriver at den modtager en start-besked. Den udskriver derefter sensor-værdierne, og sender så en besked tilbage over TCP som forventet.

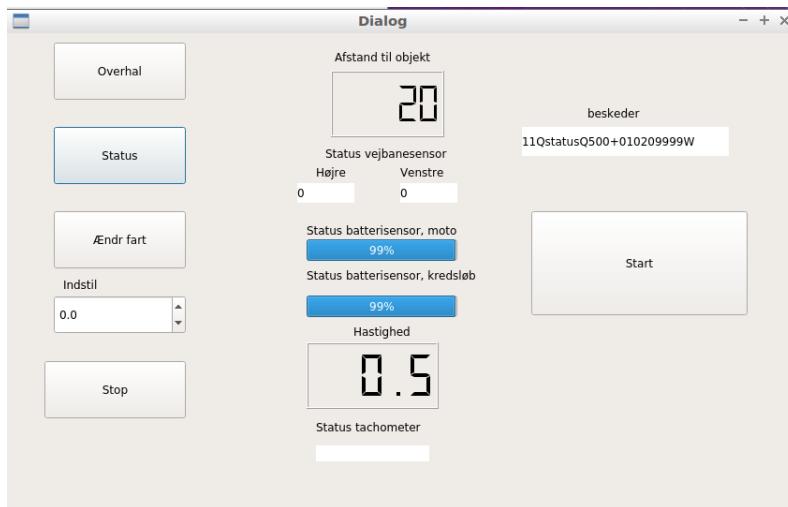
Name	Value	Address	Type	Radix
Run	0x01 '\001'	0x200000CC (All)	unsigned char	Default
setSpd	0x00 '\000'	0x2000013C (All)	signed char	Default
Click here to add				

Locals
 Watch 1
 Registers
 Memory 1

Figur 5.60: Watch på Run-variablen på PSoC

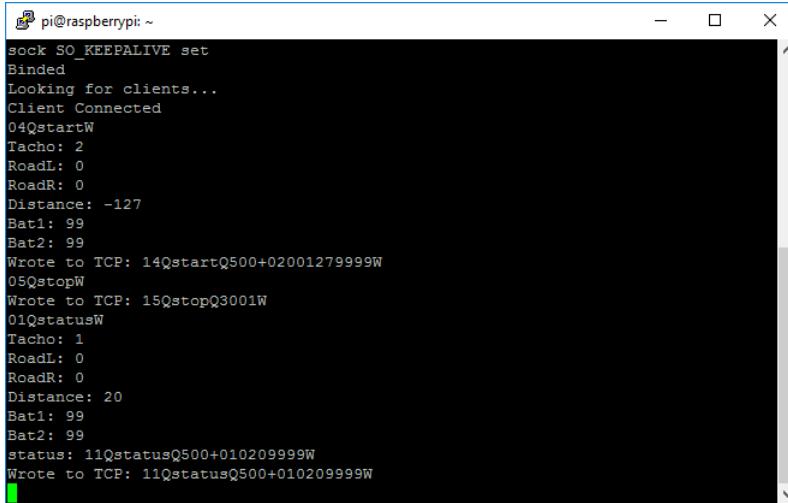
På PSoC'ens watch kan det ses at Run-variablen er blevet opdateret til 1.

Statusknap på GUI testes. Her forventes i store træk det samme som under start. Det er meningen af der skal udskrives en TCP-besked på Raspberry Pi-terminalen. Det forventes også at data bliver skrevet ud i terminalen. Det forventes ikke at der sker noget på PSoC'en, da der læses fra det data som den sender. Det sidste der forventes er at de passende felter på GUI'et opdaterer så det korresponderer med den data som blev modtaget og udskrevet på Raspberry Pi-terminalen.



Figur 5.61: GUI efter status er valgt

Nu bedes der om status, og det ses at de felter hvor sensor-værdierne skal udskrives bliver opdateret med den værdi der er modtaget. Beskeden der er modtaget passer også.



```

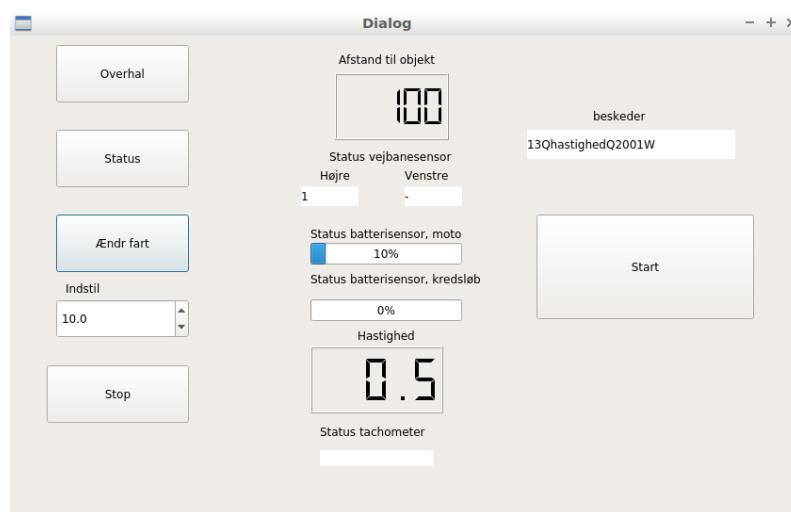
pi@raspberrypi: ~
sock SO_KEEPALIVE set
Binded
Looking for clients...
Client Connected
04QstartW
Tacho: 2
RoadL: 0
RoadR: 0
Distance: -127
Bat1: 99
Bat2: 99
Wrote to TCP: 14QstartQ500+02001279999W
05QstopW
Wrote to TCP: 15QstopQ3001W
01QstatusW
Tacho: 1
RoadL: 0
RoadR: 0
Distance: 20
Bat1: 99
Bat2: 99
status: 11QstatusQ500+010209999W
Wrote to TCP: 11QstatusQ500+010209999W

```

Figur 5.62: Raspberry Pi's terminaludskrift af status

Ligesom under test af start udskrives sensor-værdierne i terminalen efter at der er blevet udskrevet at der er modtaget en status-besked. Til sidst skrives der tilbage over TCP som før.

Det testes om ændr fart-funktionaliteten virker ved at vælge en hastighed og trykke på ændr fart. Det forventes at der ses en hastighed som er dobbelt så høj som den valgt, udskrevet på terminalen på Raspberry Pi'en, og tilsvarende tjekkes det om den værdi som ses i terminalen på Raspberry Pi'en også ses i debugging watches på hastighedsvariablen. Det tjekkes også hvad der kommer tilbage på GUI'et, og det forventes at den rigtige besked modtages i GUI'et.



Figur 5.63: GUI efter fartændring

Der er blevet valgt en fart på 10.0 km/h, og så trykkes der ændr fart. Det ses at der modtages en besked fra Raspberry Pi'en om at farten er blevet indstillet korrekt.

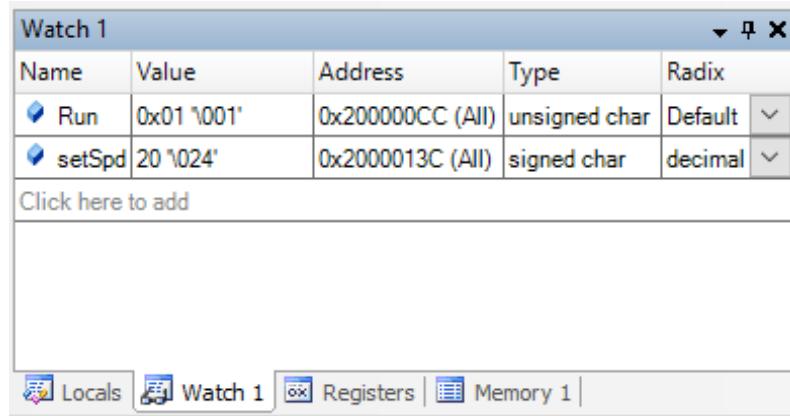
```

Looking for clients...
^Cpi@raspberrypi:~$ sudo ./Car/bin/host/prog
Setting up TCP server
port: 9000
Socket opened
sock SO_REUSEADDR set
Error on keepalive SO_OPT
sock SO_KEEPALIVE set
Binded
Looking for clients...
Client Connected
05QstopW
Wrote to TCP: 15QstopQ3001W
04QstartW
Tacho: -1
RoadL: -1
RoadR: -1
Distance: -1
Bat1: -1
Bat2: -1
Wrote to TCP: 14QstartQ5-1-1-010010-10-1W
03QhastighedQ20W
Wrote to TCP: 13QhastighedQ2001W

```

Figur 5.64: Raspberry Pi's terminaludskrift

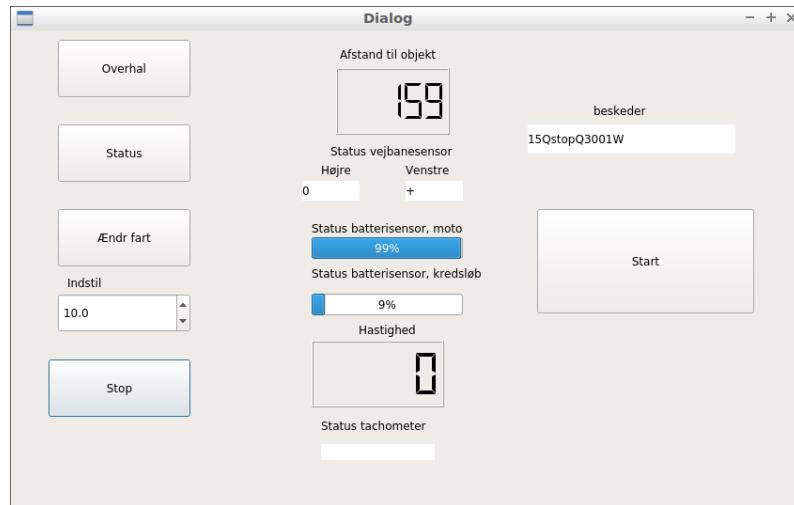
Raspberry Pi-terminalen udskriver at hastigheds-beskeden er modtaget og skriver derefter tilbage til GUI'et.



Figur 5.65: Watch på setSpd-variablen på PSoC

Det ses at setSpd-variablen bliver sat til 20 som forventet, da hastigheder bliver gemt dobbelt så høje som deres faktiske værdi.

Til sidst tjekkes det om stop-knappen opfører sig som forventet. Det forventes at Run-variablen på PSoC'en bliver sat til 0. Dette skal sørge for at bilen er "slukket". Der kontrolleres om den korrekte besked vises på Raspberry Pi-terminalen, og Run-variablen faktisk er sat til 0 på PSoC'en vha. debugging.



Figur 5.66: GUI efter at stop er trykket

Der er blevet trykket stop, og der modtages en besked om at stop er sket korrekt fra Raspberry Pi.

```

PuTTY (inactive)
Bat1: 99
Bat2: 99
Wrote to TCP: 14QstartQ510+040149999W
03QhastighedQ26W
Wrote to TCP: 13QhastighedQ2001W
03QhastighedQ00W
Wrote to TCP: 13QhastighedQ2001W
05QstopW
Wrote to TCP: 15QstopQ3001W
04QstartW
Tacho: 0
RoadL: 0
RoadR: 0
Distance: 15
Bat1: 99
Bat2: 99
Wrote to TCP: 14QstartQ500+000159999W
03QhastighedQ14W
Wrote to TCP: 13QhastighedQ2001W
03QhastighedQ20W
Wrote to TCP: 13QhastighedQ2001W
05QstopW
Wrote to TCP: 15QstopQ3001W

```

Figur 5.67: Raspberry Pi's terminaludskrift

Raspberry Pi'en udskriver at den har modtaget stop, og den svarer at den korrekt har sat Run til 0 på PSoC'en.

Name	Value	Address	Type	Radix
Run	0x00 '000'	0x200000CC (All)	unsigned char	Default
setSpd	20 '\024'	0x2000013C (All)	signed char	decimal
Click here to add				

Locals Watch 1 Registers Memory 1

Figur 5.68: Watch på run variablen på PSoC

Til sidst ses det at Run er blevet ændret til 0 på PSoC'ens watch.

6 Udført accepttest

Her ses den udførte accepttest for produktet.

Use case under test		Use case 1 - Start bil		
Scenarie		Hovedscenarie		
Prækondition		Bilen er slukket, brugerens er forbundet til bilen		
Step	Handling	Forventet	Faktisk	Vurdering
1	Vælg "Start bil" via interface	Statusrapport udskrives. For- og baglys tændes, blinklys blinker	Statusrapport udskrives. Lys tændes ikke.	Ikke godkendt

Tabel 6.1: Test for Use case 1 - Start bil - Hovedscenarie

For- og baglys er ikke blevet implementeret.

Use case under test		Use case 1 - Start bil		
Scenarie		Extension 1: Status ikke OK		
Prækondition		Vejbanesensor er disconnected		
Step	Handling	Forventet	Faktisk	Vurdering
1	Vælg "Start bil" via interface	Interfacet melder fejl på vejbanesensor, og bilen er ikke startet	Bilen starter. Vejbanesensorerne viser 0.	Ikke godkendt

Tabel 6.2: Test for Use case 1 - Start bil - Extension 1: Status ikke OK

Der er ikke implementeret tjek på om sensorerne virker.

Use case under test		Use case 2 - Rapporter driftsstatus		
Scenarie		Hovedscenarie		
Prækondition		Brugerens er forbundet til bilen, bilen er tændt		
Step	Handling	Forventet	Faktisk	Vurdering
1	Vælg "driftsstatus" via interface	Data vises på interface	Data vises på interface	Godkendt

Tabel 6.3: Test for Use case 2 - Rapporter driftsstatus - Hovedscenarie

Use case under test	Use case 2 - Rapporter driftsstatus			
Scenarie	Extension 1: Bilen er slukket			
Prækondition	Bilen er slukket, brugerne er forbundet via interface			
Step	Handling	Forventet	Faktisk	Vurdering
1	Vælg "driftsstatus" via interface	Batteriniveau udskrives. Bilen rapporterer at den er slukket via interface	Data vises på interface	Ikke godkendt

Tabel 6.4: Use case 2 - Rapporter driftsstatus - Extension 1: Bilen er slukket

Hvis bilen ikke har kørt før, sendes default værdier på -1. Ellers vises de sidst gemte værdier.

Use case under test	Use case 3: Ændr fart			
Scenarie	Hovedscenarie			
Prækondition	Use case 1 er fuldført (Bilen er i drift)			
Step	Handling	Forventet	Faktisk	Vurdering
1	Angiv fart via interface og vent	Bilen justerer sin fart	Bilen justerer sin fart	Godkendt
2	Vælg "Driftsstatus" via interface	Bilen rapporterer fart svarende til den angivne værdi	Bilen rapporterer fart svarende til den angivne værdi	Godkendt
3	Angiv fart på 0 km/t via interface	Bilen stopper	Bilen stopper	Godkendt

Tabel 6.5: Test for Use case 3 - Ændr fart - hovedscenarie

Use case under test	Use case 3 - Ændr fart			
Scenarie	Extension 1: Ønsket fart er højere end tilladt			
Prækondition	Use case 1 er fuldført (Bilen er i drift)			
Step	Handling	Forventet	Faktisk	Vurdering
1	Angiv en fart på 15 km/t via interface	Bilen advarer om at den angivne fart er for høj. Bilen justerer til højest tilladte fart.	Farten kan ikke sættes højere end det tilladte.	Ikke godkendt
2	Vælg "Driftsstatus" via interface	Bilen rapporterer fart tilsvarende den højest tilladte værdi	Bilen rapporterer fart tilsvarende den højest tilladte værdi	Godkendt

Tabel 6.6: Test for use case 3 - Ændr fart

GUI'et er implementeret således, at det ikke er muligt at sætte farten højere end de 13km/t.

Use case under test	Use case 4 - Følg vejbane			
Scenarie	Hovedscenarie			
Prækondition	Bilen er i drift. Brugeren er forbundet via interface			
Step	Handling	Forventet	Faktisk	Vurdering
1	Angiv en fart på 5 km/t	Bilen bevæger sig og holder sig inden for vejbanen	Bilen bevæger sig ikke	Ikke godkendt
2	Vælg "Driftsstatus" via interface	Bilen rapporterer fart tilsvarende den angivne værdi	Bilen rapporterer fart tilsvarende den angivne værdi	Godkendt

Tabel 6.7: Test for Use case 4 - Følg vejbane - Hovedscenarie

Bilen kører ikke når farten er så lav. Ved højere fart kører bilen for hurtigt til at den kan nå at dreje.

Use case under test		Use case 4 - Følg vejbane			
Scenarie		Extensions 1: Forhindring på vejbane			
Prækondition		Bilen er i drift og holder stille. Bruger forbundet via interface			
Step	Handling	Forventet	Faktisk		Vurdering
1	Angiv en fart på 5 km/t via interface	Bilen bevæger sig og holder sig inden for vejbanen	Bilen bevæger ikke	be-sig	Ikke godkendt
2	Testobjekt bevæges i samme retning foran bilen med lavere fart	Bilen indhenter testobjektet og justerer sin fart ned til testobjektets	Bilen bevæger ikke	be-sig	Ikke godkendt
3	Testobjektet standses	Bilen stopper inden testobjektet rammes	Bilen har ikke været igang		Ikke godkendt

Tabel 6.8: Test for Use case 4 - Følg vejbane - Extension 1: Forhindring på vejbane

Bilen kører ikke når farten er så lav. Ved højere fart vil bilen indhente testobjektet og stopper.

Use case under test		Use case 5 - Overhal			
Scenarie		Hovedscenarie			
Prækondition		Bilen er i drift. Brugeren er forbundet via interface			
Step	Handling	Forventet	Faktisk		Vurdering
1	Angiv en fart på 5 km/t via interface	Bilen bevæger sig	Bilen bevæger ikke	be-sig	Ikke godkendt
2	Testobjekt bevæges i samme retning foran bilen med lavere fart	Bilen indhenter testobjektet og justerer sin fart ned til testobjektets	Bilen bevæger ikke	be-sig	Ikke godkendt
3	Vælg "Overhal" via interface	Bilen trækker ud og accelererer til den er passeret testobjektet. Bilen trækker derefter ind og fortsætter med den angivne fart.	Bilen væger ikke	be-sig	Ikke godkendt

Tabel 6.9: Test af Use case 5 - Overhal - Hovedscenarie

Bilen kører ikke når farten er så lav. Ved højere fart stopper bilen når sonaren opfanger objektet.

Use case under test		Use case 5 - Overhal			
Scenarie		Extension 1: Overhaling ikke mulig			
Prækondition		Bilen er i drift. Brugeren er forbundet via interface			
Step		Handling	Forventet	Faktisk	Vurdering
1		Angiv en fart på 5 km/t via interface	Bilen bevæger sig	Bilen bevæger sig ikke	Ikke godkendt
2		Testobjekt bevæges i samme retning foran bilen med lidt lavere fart end bilen	Bilen indhenter testobjektet og justerer sin fart ned til testobjektets	Bilen bevæger ikke	Ikke godkendt
3		Vælg "Overhal" via interface	Bilen informerer brugeren at farten ikke er tilstrækkelig. Bilen fortsætter med samme fart.	Bilen bevæger ikke	Ikke godkendt

Tabel 6.10: Test af Use case 5 - Overhal - Extension 1: Overhaling ikke mulig

Use case under test		Use case 6 - Sluk bil			
Scenarie		Hovedscenarie			
Prækondition		Bilen er i drift. Brugeren er forbundet via interface			
Step		Handling	Forventet	Faktisk	Vurdering
1		Angiv en fart på 5 km/t via interface	Bilen bevæger sig	Bilen bevæger sig ikke	Ikke godkendt
2		Vælg "Sluk bil" via interface	Bilen stopper. Bilens lys slukkes. Der udskrives på interfacet at bilen er slukket.	Bilen er stoppet.	Ikke godkendt

Tabel 6.11: Test af Use case 6 - Sluk bil - Hovedscenarie

Der kommer en protokol meddeelse om stop, men ikke en direkte "stop"besked.

6.1 Test af ikke-funktionelle krav

Krav	Test	Forventet resultat	Resultat	Vurdering
Bilens længde og bredde må ikke overskride 50cm x 30cm	Længde og bredde måles	Længden og bredden overskider ikke 50cm x 30cm	Længden er 51.7cm og bredden er 27.5cm	Ikke godkendt
Brugeren skal have mulighed for at kommunikere med bilen via tekst-terminal og / eller grafisk brugergrænseflade	Visuel test	Der er en grafisk brugergrænseflade	Der er en grafisk brugergrænseflade	Godkendt
Bilens skal detektere objekter på en afstand i intervallet 20cm til 2,5m i spring af 5cm	Der placeres et objekt 20cm foran bilen. Driftsstatus vælges og aflæses via interface. Objektet føres ud i en afstand 2,5m i skridt af 5cm og driftsstatus vælges og aflæses via interface.	Detektion i intervallet fra 20cm til 2,5m med oplosning på 5cm	Ved 20cm detekterer sonaren objektet korrekt. Ved 150cm detekterer sonaren gulvet.	Ikke godkendt.
Bilen skal kunne køre med en fart på mindst 13 km/t \pm 0,5	Der afmåles et vejbanestykke på 20m og et på 10m i forlængelse af hinanden. Der angives en fart på 13 km/t. Idet bilen passerede de første 20m startes et stoppur og stoppes efter bilen har bevæget sig yderligere 10m	Bilen kan køre 13 km/t \pm 0,5 km/t	Bilen kan køre 13 km/t \pm 0,5 km/t	Godkendt

Krav	Test	Forventet resultat	Resultat	Vurdering
Bilen skal have en maksimal vægt på 2 kg	Bilen placeres på en vægt	Vægten er under 2 kg	Bilen vejer 2.15 kg	Ikke godkendt.
Bilen skal måle sin fart med en opløsning på 0,5 km/t og bilen skal kunne indstille sin fart med en opløsning på 0,5 km/t	Der afmåles et vejbanestykke på 20m og et på 10m i forlængelse af hinanden. Bilens indstilles til forskellige hastigheder fra 0 km/t til 13 km/t i trin af 0,5 km/t. Idet bilen passerer de første 20m startes et stopur og stoppes idet den har bevæget sig yderligere 10m. Der læses fart ud fra driftsstatus i løbet af de sidste 10m.	Bilen kan aflæse og indstille sin fart med en opløsning på 0,5 km/t	Farten bilen kører er forkert.	Ikke godkendt

Litteraturliste

- [1] Orr Hirschauge. "Are Driverless Cars Safer Cars?" I: *Wall Street Journal* (2015). URL: <http://www.wsj.com/articles/are-driverless-cars-safer-cars-1439544601>.
- [2] Chris Urmson. "How a driverless car sees the road". I: 2015. URL: <https://www.youtube.com/watch?v=tiwVMrTLUWg>.
- [3] Sparkfun. *Rover 5 Robot Platform*. URL: <https://www.sparkfun.com/products/10336>.
- [4] Robot Living. *Two Wheeled Robot*. URL: <http://www.robotliving.com/building-robots/two-wheeled-robot-2/>.
- [5] Teflon. *Disc brakes and power steering!!!* 23. jun. 2007. URL: <http://forum.lro.com/viewtopic.php?f=24&t=3806>.
- [6] automobiledimension. *Dimensions of new Audi cars showing length, width and height*. 2016. URL: <http://www.automobiledimension.com/audi-car-dimensions.html>.
- [7] Infineon Technologies. *TLE4905 datasheet*.
- [8] MaxBotix inc. *LV-MaxSonar-EZ™.pdf*. 26. maj 2016.
- [9] Cadie Thompson. "There's one big difference between Google and Tesla's self-driving car technology". I: *TechInsider* (2015). URL: <http://www.techinsider.io/difference-between-google-and-tesla-driverless-cars-2015-12>.
- [10] Various. *Density of air*. 2016. URL: https://en.wikipedia.org/wiki/Density_of_air#Composition.
- [11] Various. *Molecular Weight - Gases and Vapors*. 2016. URL: http://www.engineeringtoolbox.com/molecular-weight-gas-vapor-d_1156.html.
- [12] Siemens. *SFH485.datasheet.pdf*.
- [13] OSRAM. *SFH203FA Datasheet.pdf*.
- [14] National semiconductor. *LM358M.pdf*.
- [15] Vishwam Aggarwal. *How to build an IR Sensor*. 4. aug. 2013. URL: <http://maxembedded.com/2013/08/how-to-build-an-ir-sensor/>.
- [16] Wikipedia. *Electronic speed control — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Electronic_speed_control&oldid=716831813.
- [17] Microchip Technology Inc Padmaraja Yedamale. *AN885 - Brushless DC (BLDC) Motor Fundamentals*. 2003. URL: <http://ww1.microchip.com/downloads/en/AppNotes/00885a.pdf>.
- [18] hobbyking.com. *Quanum MT Series 5208 360KV Brushless Multirotor Motor Built by DYS*. 2016. URL: http://www.hobbyking.com/hobbyking/store/_67037_Quanum_MT_Series_5208_360KV_Brushless_Multirotor_Motor_Built_by_DYS.html.

- [19] Tore Skogberg. *Analogteknik - T-005*. 2016.
- [20] Cypax. KØLEFINNE TO220 CLIP-0N 13x19x13. URL: <http://www.cypax.dk/vare/40.081.0011>.
- [21] Jon Klein Fairchild Semiconductor. *AN-6005 - Synchronous buck MOSFET loss calculations with Excel model*. 21. nov. 2014. URL: <https://www.fairchildsemi.com/application-notes/AN/AN-6005.pdf>.
- [22] International Rectifier Vrej Barkhordarian. *Power MOSFET Basics*. URL: <http://www.infineon.com/dgdl/mosfet.pdf?fileId=5546d462533600a4015357444e913f4f>.
- [23] Peter Markowski. "Estimating MOSFET switching losses means higher performance buck converters". I: *EETimes* (12. nov. 2002). URL: http://www.eetimes.com/document.asp?doc_id=1225701.
- [24] Cypax. *LEDN BLØD 0,75 mm² SORT H07ZK rl=200m (PRODUKTSIDE)*. URL: <http://www.cypax.dk/vare/60.010.0750>.
- [25] Lars Mandrup et al. Keld Skov. *T-306. Grundlæggende EMC I*. Dec. 2001.
- [26] Steven Keeping. *Controlling Sensorless, BLDC Motors via Back EMF*. URL: <http://www.digikey.com/en/articles/techzone/2013/jun/controlling-sensorless-bldc-motors-via-back-emf>.
- [27] Frances Rees. *How Do Servo Motors Work*. URL: <http://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html>.
- [28] Murata Power Solutions. *Fixed Output 1.5 Amp SIP DC/DC Converters*. 2014.
- [29] FAIRCHILD. *3-Terminal 1 A Positive Voltage Regulator*. Sep. 2014.
- [30] Microchip Technology Inc. Ward Brown. *AN857 - Brushless DC Motor Control Made Easy*. 2011. URL: <http://ww1.microchip.com/downloads/en/AppNotes/00857B.pdf>.
- [31] Jed Storey. *Hall Effect Sensor Placement for Permanent Magnet Brushless DC Motors*. 20. aug. 2011. URL: <http://mitrocketscience.blogspot.dk/2011/08/hall-effect-sensor-placement-for.html>.
- [32] G.C.Y. Chong og Y. Li K.H. Ang. "PID control system analysis, design, and technology". I: *IEEE Trans Control Systems Tech* (2005). URL: <http://eprints.gla.ac.uk/3817/1/IEEE3.pdf>.
- [33] Mike McCauley. *C library for Broadcom BCM 2835 as used in Raspberry Pi*. 9. feb. 2016. URL: <http://www.airspayce.com/mikem/bcm2835/>.
- [34] gongzhitao. *How to calculate a time difference in c++*. 5. nov. 2013. URL: <http://stackoverflow.com/questions/728068/how-to-calculate-a-time-difference-in-c>.
- [35] James F. Kurose og Keith W. Ross. *Computer Networking - A Top-Down Approach*. 5. mar. 2012.