

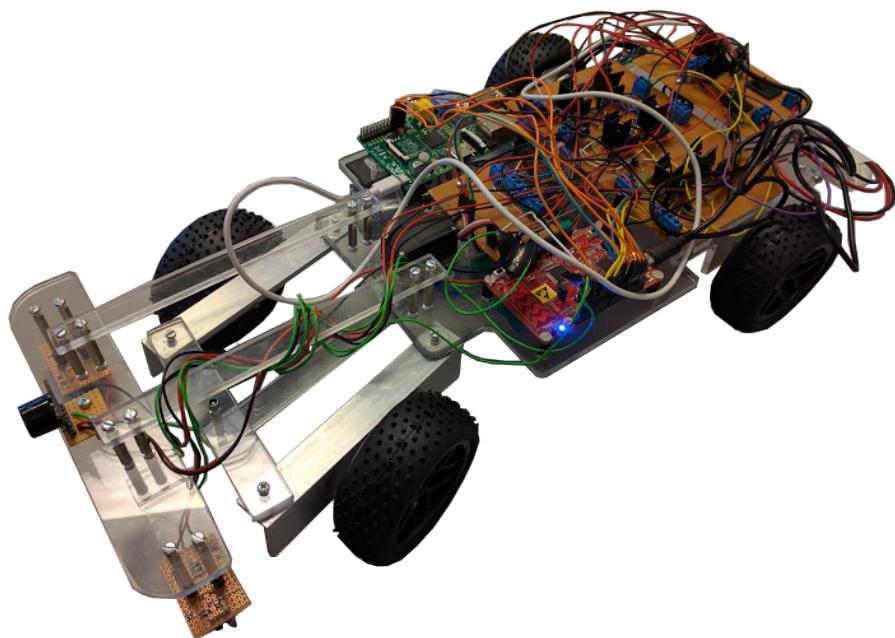
Projekt AutoCar

Rapport

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

Nicolai F

Nicolai H. Fransen
Studienr. 201404672

Nicolai Bonde

Nicolai Bonde
Studienr. 201404519

Emil Jepsen

Emil Jepsen
Studienr. 20092013

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 over ansvarsfordeling i projektet

Resumé

Denne rapport beskriver udviklingen af et 4. semester-projekt på IHA. Problemstillingen omhandler design og implementering af en selvkørende bil, AutoCar. AutoCar har en DC-motor til fremdrift, en servo-motor til retningsstyring, en sonar-sensor til afstandsbedømmelse, en vejbanesensor i hver side til detektion af vejbanestriber, to batterisensorer til at vise batteriniveauerne, et tachometer til at finde AutoCar's fart, og intern logik, regulering, og kommunikation for at få det hele til at fungere sammen. Brugeren kan via interfacet starte og stoppe AutoCar, få udskrevet status fra sensorne, sætte farten og anmode om overhaling. AutoCar skal selv sørge for at køre inden for vejbanestriberne.

AutoCar er udviklet med en PSoC 4 og en Raspberry Pi 2b, der tilsammen fungerer som kontrolenhed for AutoCar. GUI'et er designet med QT Creator i en Linux-terminal.

Udviklingsprocessen har båret præg af iterative værktøjer som SCRUM og V-modellen, og med fokus på design fra bunden og op. ASE-modellen for projektudvikling er også benyttet i den tidsmaessige planlægning. Brugen af disse modeller har hjulpet i udviklingen af projekt. Projektprocessen er mundet ud i en prototype hvor størstedelen af funktionaliteten er implementeret.

Abstract

This report describes the development of a 4th semester project at IHA concerning the design and development of a self-driving car, AutoCar. AutoCar has a DC-motor for propulsion, a servo for directional control, a sonar sensor used for ranging, a road sensor in each side to detect road markings, two battery sensors to detect and display battery levels, a tachometer to measure the speed of AutoCar, and a lot of internal logic, regulation, and communication to make it all work. The user can use the interface to start and stop AutoCar, print the status from the sensors, set the speed, and a request overtaking. AutoCar is itself responsible for driving within the road markings.

AutoCar is developed with a PSoC 4 and a Raspberry Pi 2b, which collectively function as the control unit. The GUI is designed with QT Creator in a Linux terminal.

The development process has been influenced by iterative tools such as SCRUM and the V model, and a strong focus on a bottom-up approach to the design process. The ASE Development Model has been used for chronological planning, and redmine along with SmartGit has been used to keep track of project files and resources. The use of these models and tools has ensured a competently developed system.

Indhold

Indhold	2
1 Indledning	3
2 Opgaveformulering	4
3 Projektafgrænsning	5
4 Systembeskrivelse	6
5 Krav	7
5.1 Kravspecifikation	7
6 Projektbeskrivelse	10
6.1 Projektgennemførelse	10
6.2 Metoder	11
6.3 Systembetragtning	13
6.4 Arkitektur	15
7 Hardware - design, implementering og test	19
7.1 Chassis	19
7.2 ESC	19
7.3 Effektkredsløb til motoren	20
7.4 Spændingsregulator	22
7.5 Servo	22
7.6 Sonar	22
7.7 Vejbanesensor	23
8 Software - design, implementering og test	25
8.1 SPI kommunikation	25
8.2 Tachometer	25
8.3 Sonar	26
8.4 Vejbanesensor	26
8.5 ESC	27
8.6 Raspberry Pi	28
8.7 TCP PC	29
8.8 GUI	30
9 Resultater og diskussion	32
9.1 Fremtidigt arbejde	33
10 Konklusion	34
Litteraturliste	35

Forord

Denne rapport omfatter projektarbejdet for gruppe 2 på 4. semester for Elektronikingeniøruddannelsen forår 2016. Rapporten afspejler gruppens arbejde i perioden fra projektstart d. 5. februar til projektaflevering d. 27 maj.

Afleveret elektronisk sammen med denne rapport findes projektdokumentationen samt bilag. Dybdegående forklaring af produktets specifikation, design og implementering kan findes i dokumentationen. I bilagsmappen findes al koden der ligger til grund for softwaren i systemet, samt diagrammer og kredsløbstegninger.

1 Indledning

Hel- og halv-automatiserede køretøjer er en teknologi der bliver mere og mere udbredt i dag[1]. Ved at eliminere menneskelige faktorer i trafikken er der opstået nye muligheder for komfort, trafiksikkerhed og miljøvenlige løsninger inden for trafikverdenen. Automatiske biler skaber et nyt domæne inden for komfort for bilisterne. Køretøjer der assisterer bilisten med at komme frem til sin destination i større omfang er ikke længere en fantasi[2]. Trafiksikkerhed kan også udvides i den forstand at menneskelige fejl vil kunne eliminieres eller mindskes. Ved at have biler udstyret med intelligente sensorer og alarmnetværk er det muligt for bilen at opfange de farer som bilisten vil overse[3]. Trafikforhold kan optimeres med intelligente biler. Sikkerhedsafstand og trafikkapacitet kan optimeres ved at lade det intelligente udstyr styre bilen[4].

Målet for dette projekt er at udvikle et demonstrator for et bilsystem med intelligente sensorer og styring. Der udvikles et model for en rigtig bil, som udstyres med sensorer der gør det muligt for den at følge vejens forløb samt at reagere på bestemte trafikbetingelser.

Brugerens interaktion med bilen modelleres på en grafisk brugergrænseflade på et system forbundet over internettet. Systemet skal have forbindelse med bilen og skal give brugeren mulighed for at monitorere bilens forhold samt at udstede kommandoer til hvordan bilen skal agere på vejbanen.

Projektet er udarbejdet med en iterativ udviklingsproces, hvor projektets kernelementer er prioriteret. Arbejdet er inddelt i forskellige faser over projektperioden, hvor hver fase har haft fokus på et specifikt aspekt.

Kilder er refereret med en numerisk reference omsluttet af firkantede parenteser eks. [1]. Referencen kan findes under afsnittet *Referencer*, hvor *forfatter*, *titel*, *årstal* er angivet. Refereres til et bestemt kapitel, afsnit eller figur er dette angivet før nummeret, [FIGUR 18, 1]. Henvises til en bestemt side er det anført på formen [1, s. 164]. Interne referencer til afsnit, figurer eller tabeller i dette dokument er vist med *Type af reference* og *afsnit.figurnummer*, eksempelvis *figur 5.3*. Henvisninger til dokumentationen er dette tydeliggjort i teksten, *ses i dokumentationen...*, og refereres på samme måde.

2 Opgaveformulering

I projektet skal der udvikles en model af et automatiseret bil-system. Bilen skal kunne styres trådløst og skal kunne følge en bane autonomt. Bilen skal kunne interagere med omverdenen vha. sensorer og aktuatorer. Der skal indgå pålidelig transmission af data mellem brugeren og bilen. Der skal også implementeres en brugergrænseflade.

Følgende punkter skal indgå i produktet:

- En eller flere sensorer til rumlig opfattelse
- Indlejret platform
- Interaktion til systemet igennem brugerflade på en computer
- Pålidelig trådløs kommunikation over WiFi
- Styring og regulering af motor. Hastigheden skal kunne indstilles præcist
- Styring af retning

Brugeren skal kunne interagere med systemet igennem en brugergrænseflade kørende på en computer. Ved denne interaktion skal det være muligt for brugeren at kunne regulere bilens fart, samt kontrollere sensor data. Computeren skal forbindes trådløst til bilen via WiFi. Kommunikationen skal være pålidelig.

Hastigheden skal kunne indstilles på bilen og skal kunne holdes konstant ved almindelig kørsel på jævn overflade. Bilen skal kunne rapportere sin status til brugeren.

3 Projektafgrænsning

I dette afsnit beskrives det egentlige udgangspunkt for hvilket produkt der skal udvikles, men også det egentlige arbejde der er gjort. Elementer af projektet der er specificeret, men ikke implementeret, er nævnt.

De vigtigste use cases samt projektets kerneelementer er prioriteret i udviklingen af produktet. Dette indebærer kernemoduler, som kræves udviklet, før andre dele af projektet kan fuldføres. En stor del af arbejdet er derfor lagt i at udbygge en ramme for produktet, hvorpå de andre detaljer vil kunne påføres.

Der er valgt blot at udvikle en demonstrator for en egentlig bil, da det er et væsentligt nemmere system at arbejde med. Bilen er udviklet så den kan bevæge sig på en modelbane. Banen består af vejbanestriber i lige strækning samt i sving. Vejbanestriberne er modelleret med et stykke tape med en anden farve end gulvet.

Bilens mekaniske opbygning er specialdesignet til produktet. Det er vanskeligt at finde en fjernstyret bil i en passende prisklasse, hvis størrelse skal stemme overens med projektets mål, så der er udviklet en løsning til dette.

Trådløs kommunikation over WiFi er implementeret i produktet. Det primære netværk der forbindes til er skolens eduroam netværk. Dette viste sig at volde problemer, da der bliver tildelt en ny IP-adresse hver dag og det skal der tages forbehold for. Det er desuden besværligt at oprette forbindelse til netværket igennem WiFi modulet, og det er en faktor der har voldt tidsmæssige problemer.

Der er valgt at bruge en brushless motor til projektet, som er monteret på bilen og der er udviklet regulering til. Motorstyringen afhænger af at der er monteret Hall-effekt sensorer i den. Motoren kom uden monterede sensorer, og viste sig at være en besværlig opgave på grund af motorens opbygning. Dette betød at der blev brugt lang tid på at montere sensorerne.

Bilen er udstyret med optiske sensorer som kan detektere vejbanestriberne på banen. Vejbanesensorernes implementering har fungeret tilfredsstillende, men er muligvis ikke nok til at opfylde kravene til bilens rumlige opfattelse. Hertil skal der evt. udtænkes en mere robust løsning.

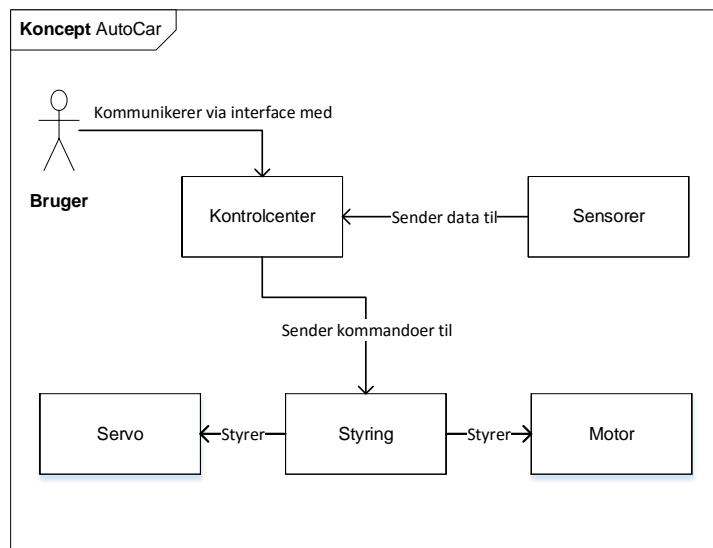
Det er blevet specificeret at bilen skal kunne monitorere batteriniveau og der skal være monteret forlys, baglys og blinklys, men ingen af disse er implementeret. Der er undersøgt strategier for batterimonitorering, men på grund af tidsnød er det blevet ned-prioriteret.

4 Systembeskrivelse

AutoCar består af tre hovedblokke – en række sensorer der monitorerer på vejens forhold, to aktuatorer i form af en servo og en motor, samt et kontrolcenter der står for alt logik. Kontrolcenteret består af en Raspberry Pi, der fungerer som beslutningstageren, og en PSoC, der fungerer som en forwarder, ved at sample data fra sensorerne og videoresende hastighed og retning til de to motorer. Der foregår en løbende kommunikation mellem Raspberry Pi og PSoC for at kunne kontrollere bilen.

Kommunikationen mellem brugeren og kontrolcenteret foregår via et GUI. Gennem det kan brugeren først og fremmest styre hastigheden, men også kontrollere de data der bliver sendt fra sensorerne.

På figur 9.1 ses et diagram over konceptet af AutoCar. Det giver et overblik, over hvad der er en del af AutoCar, og hvilken vej kommunikationen foregår.



Figur 4.1: Koncept for autoCar

5 Krav

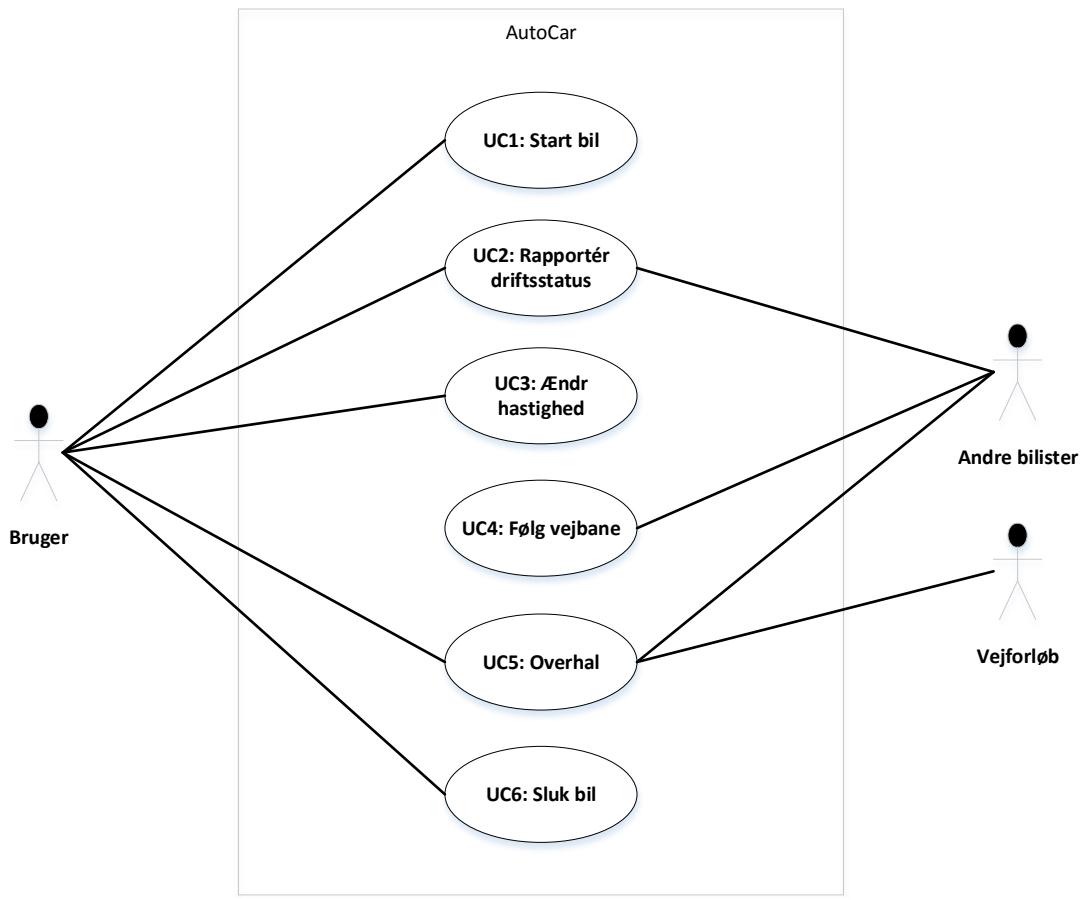
Kravene til produktet er analyseret vha. MoSCoW-metoden[5]. MoSCoW-metoden inddeles kravene til et system i fire kategorier. De fire kategorier er: Must, Should, Could og Won't.

- | | |
|---------------|--|
| 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- Trådløs kommunikation 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 kørellys- Undvige forhindring- Grafisk brugergrænseflade- Køre baglæns |
| Won't | <ul style="list-style-type: none">- Have navigation- Læse vejskilte- SmartPhone-applikation til styring |

Ud fra ovenstående MoSCoW-model er der fremstillet en række krav. Der er funktionelle krav, som defineres ved use cases, og ikke-funktionelle krav.

5.1 Kravspecifikation

Kravspecifikationen indeholder kravene til projektet. Use casene er funktionelle krav til systemet set fra brugerens perspektiv. I diagrammet nedenfor beskrives brugerens og systemets sammenhæng til de funktionelle krav for produktet. Brugeren kan via kommandoer på grænsefladen styre bilen, ved at initiere de forskellige use cases. De andre aktører ses til højre på figuren, og linjerne viser hvilke use cases de er involveret i.



Figur 5.1: Use case diagram

5.1.1 Use cases

De seks use cases forklares i dette afsnit. For yderligere beskrivelse af de forskellige use cases, se dokumentationens kravspecifikation.

Use case 1: Start bil

Use casen starter bilen når brugeren vælger "Start" via grænsefladen. Herefter vil bilen monitorere sensorinputs, udføre motortjek og rapportere status til brugeren via grænseflade.

Use case 2: Rapportér driftstatus

Use casen rapporterer om driftstatus når brugeren vælger "Status" via grænsefladen. Dette indebærer batteriniveau, bilens fart, status på vejbanesensorer og afstand til objekt foran.

Use case 3: Ændr fart

Brugeren indstiller den ønskede fart via grænsefladen og use casen justerer derefter farten til den ønskede værdi.

Use case 4: Følg vejbane

Use casen sørger for at bilen korrigerer sin retning i forhold til vejbanen. Dette er den

eneste use case, som ikke initieres af brugeren.

Use case 5: Overhal

Use casen overhaler et forankørende objekt når brugeren anmoder om en overhaling via grænsefladen.

Use case 6: Sluk bil

Use casen slukker bilen når brugeren vælger "Sluk" via grænsefladen. Her vil bilens fart sættes til nul samt slukke for sensorer, lys og motor. Dette udskrives på grænsefladen.

5.1.2 Ikke-funktionelle krav

Kravene for kvaliteten af bilen beskrives ved ikke-funktionelle krav. Disse krav er beskrevet under kravspecifikationen i dokumentationen.

Kravene indebærer bilens dimensioner i form af størrelse og vægt. Der stilles krav til bilens topfart og sonarens rækkevidde. Yderligere er der krav til, at brugeren skal kunne kommunikere med bilen via en grafisk brugergrænseflade.

6 Projektbeskrivelse

6.1 Projektgennemførelse

Udviklingen af AutoCar strakte sig over en periode på fire måneder. Sideløbende undervisning dannede grundlaget for meget af projektets indhold, som motorstyring, reguleringslæring, trådløs kommunikation med TCP, og andre relevante områder. Der er i projektet taget udgangspunkt i den fra tidligere projekter kendte ASE-model. Denne model samt aflevering af overordnet systemforslag tidligt i processen og et reviewmøde efter systemarkitekturen var udarbejdet, dannede rammerne for arbejdet på AutoCar. Projektmøder var igennem hele processen regelmæssige; som regel en gang om ugen, men ofte mere.

Til at fastlægge systemets overordnede arkitektur både hardware- og software-mæssigt er der arbejdet fælles, da dette gav alle i gruppen samme udgangspunkt når der senere skulle uddelegeres opgaver i mindre grupper. Ud fra dette arbejde med arkitekturen blev det vurderet hvor komplekst og tidskrævende hvert modul i systemet ville blive, og hvor mange resourcer der derfor skulle afsættes til det. Umiddelbart efter systemarkitekturen og reviewmødet påbegyndte gruppen designfasen, og projektgruppen blev opdelt.

Produktudviklingen tog udgangspunkt i Scrum, da gruppens medlemmer tidligere havde anvendt denne udviklingsmetode med gode resultater. På grund af sideløbende fag var det svært at anvende alle Scrum's elementer og arbejds metoder, som f.eks. product owner, daglige stand-up Scrum-møder og forventningskalibrering mellem Scrum-master og product owner. Derfor benyttedes en modificeret Scrum-model med mere flydende sprints og ugentlige Scrum-møder, hvor der både blev foretaget sprint-planlægning, udvikling, og sprint-review på samme dag. Da der i den første fase ikke blev udviklet meget ugentligt, men arbejdet var fokuseret på udarbejdelse af krav, arkitektur, og accepttest, blev det besluttet at sprint-retrospektiv ville fratake tid der i stedet kunne bruges på udvikling. Til at holde styr på backloggen og udvælge items til sprints anvendte gruppen Redmine's task board, som gav et godt overblik over hvilke opgaver der skulle færdiggøres hvornår, og gav mulighed for at introducere nye opgaver til hvert modul efterhånden som designfasen var undervejs og nye problemstillinger kunne identificeres. Til at holde styr på filer, versionshistorik, og merging blev Git anvendt.

Tidligt i processen var gruppen klar over at det var vigtigt at få bestilt de præfabrikerede dele så hurtigt som muligt, så de kunne testes og integreres. Det drejede sig primært om motoren, karosseriet til bilen, bilens hjul, og sonaren. Motoren, hjulene og sonaren blev bestilt og ankom relativt hurtigt, men det var svært at finde et chassis hvorpå hardwaren kunne placeres.

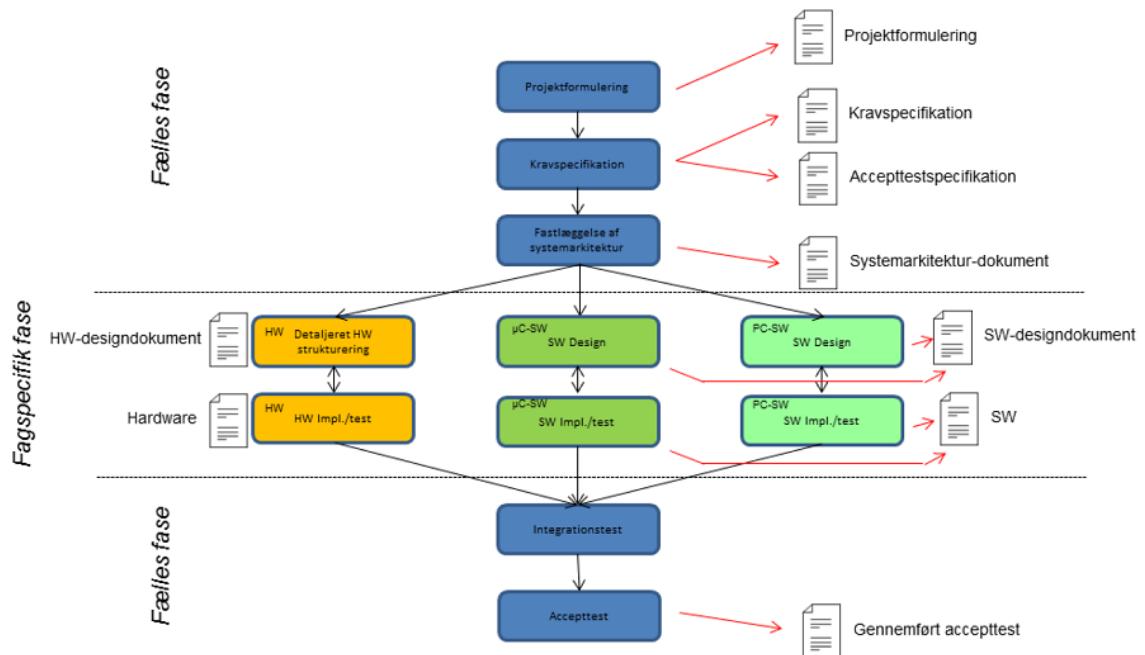
Efter systemets enkeltdele var færdigdesignet og testet blev der foretaget to integrationstests hhv. for PSoC'en og dens tilhørende elementer, og for kommunikationsvejen fra GUI til Raspberry Pi til PSoC. Gennem udviklingsprocessen blev der desuden skrevet dokumentation og test for hver af systemets enheder.

6.2 Metoder

I de følgende afsnit beskrives de anvendte metoder til udvikling af projektet.

6.2.1 ASE-modellen

På figur 6.1 ses ASE udviklingsmodellen, der er blevet brugt til at fastlægge den kronologiske udvælgelse af opgaver fra product backloggen. Desuden var processen underlagt aflevering af arkitektur til et review relativt tidligt.

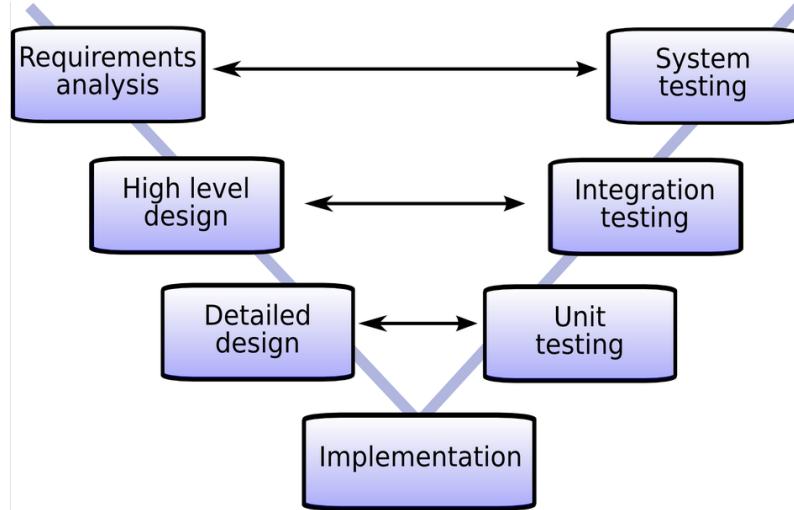


Figur 6.1: ASE udviklingsmodellen[6]

Modellen minder meget om vandfaltsmodellen hvor alle krav fastlægges i starten af projektet, derefter udvikles arkitekturen, så går projekt ind i designfasen, og til sidst i testfasen. Denne skarpe opdeling blev overholdt i starten af projektet, men afveg efterhånden som projektgruppen tog fat på deres ansvarsområder i løbet af designfasen. I denne fase blev modulerne brudt så langt ned som muligt, og blev derefter designet fra bunden op. Dette stemte overens med den næste metode der gjordes brug af i projektudviklingen.

6.2.2 V-modellen

V-modellen er en komponentbaseret udviklingsmodel der kan benyttes til hardware- og softwareudvikling, som beskriver hver udviklingsfase ud fra hvilke tests de er tilknyttet. Figur 6.2 giver et overblik over modellen.

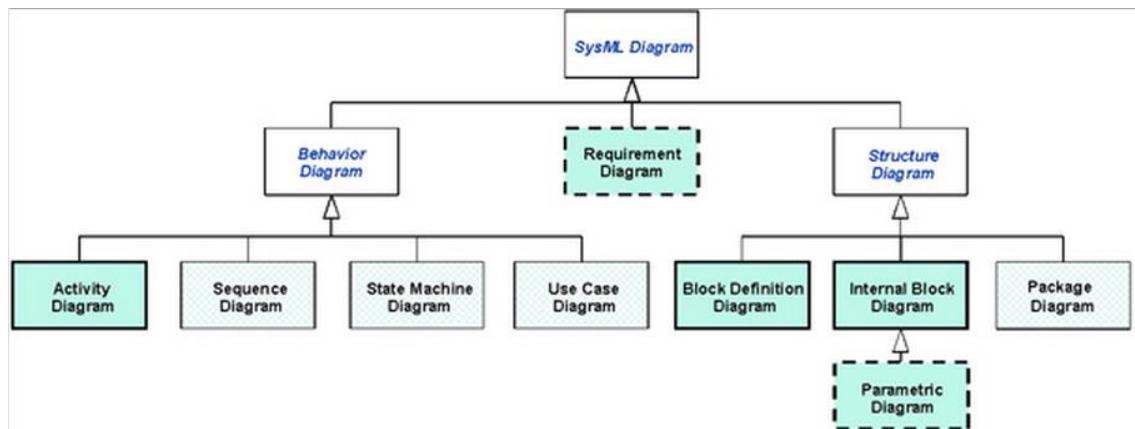


Figur 6.2: V-modellen[7]

Modellen viser, at der ud fra kravspecifikationen skrives en accepttest, ud fra det overordnede design skrives integrationstests, og for hver enkelt modul i systemet skrives enhedstests. Denne model blev benyttet til udviklingen af hele systemet og enkeltnoder. Modellen er utrolig brugbar til at nedbryde et komplekst problem til flere simple dele. Efterhånden som flere af disse mindre deles funktion kan verificeres i test, kan større, og mere omfattende, tests af det større problem foretages. Til sidst kan det samlede projekt, eller modul, testes og godkendes. Modellen blev anvendt meget under udviklingen af AutoCar.

6.2.3 SysML

Til at udarbejde systemarkitektur og design af systemet er SysML benyttet. SysML er en måde at modellere systemer med både software- og hardwarekomponenter, hvor UML primært beskriver software-systemer. Figur 6.3 viser et overblik over de forskellige diagramtyper i SysML.



Figur 6.3: SysML opbygning[8]

En af de store fordele ved at bruge SysML er at de fastlagte standarder gør at andre

udviklere hurtigt kan få et overblik over systemet, ved at se på SysML-diagrammerne i systemarkitekturen. Desuden gør det designfasen nemmere, når systemets virkemåde på forhånd er beskrevet i en vis detaljeringsgrad før design-fasen er påbegyndt.

De to strukturdiagrammer der blev anvendt i udarbejdelsen af systemarkitekturen er Internal Block Diagram(IBM), og Block Definition Diagram (BDD). BDD'erne bruges til at nedbryde systemerne i mindre blokke, hvilket især gør det nemmere at beskrive AutoCar, da denne har mange tilknyttede moduler. IBM'erne benyttes til at vise modulernes sammenhæng rent hardwaremæssigt. Til et IBM kan det ofte betale sig at tilknytte en signalliste, der beskriver præcis hvilke forbindelser diagrammet modellerer i den virkelige verden, og hvilke logiske niveauer de er på.

Adfærdsdiagrammerne der anvendes er Sequence Diagram (SD) og use case-diagram. Disse bruges primært til at beskrive hvordan softwaren opfører sig. Use case-diagrammer beskriver hvilke aktører der interagerer med systemet der designes, og hvilke use cases dette indeholder. Der er desuden brugt en applikationsmodel og en domænemodel i arkitekturen til at beskrive softwarens opførsel og overordnede struktur. Den overordnede struktur beskrives i flere detaljer i dokumentationsafsnittet for Overordnet Software-design med klassediagrammer og tilhørende funktionsbeskrivelser for hver af systemets softwaremoduler.

Hardwaren i systemet er designet ud fra det, der i SysML kaldes en "black-box approach", hvor de enkelte hardwareblokke i det overordnede IBM behandles som selvstændige enheder, men beskrives mere detaljeret i hver deres IBM senere i afsnittet.

6.3 Systembetragtning

Der er forud for projektarbejdet lavet betragtninger om hvordan projektet skal udformes. Der er vurderet 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.

6.3.1 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 er 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.

6.3.2 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 vejbanestriber. Det oplagte valg er optiske sensorer.

6.3.3 Kommunikation

Det er tiltænkt at der kommunikeres trådløst til bilen via WiFi. Dette kan ekstrapoleres til andre kommunikationsteknologier, eksempelvis GSM, og 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.

6.4 Arkitektur

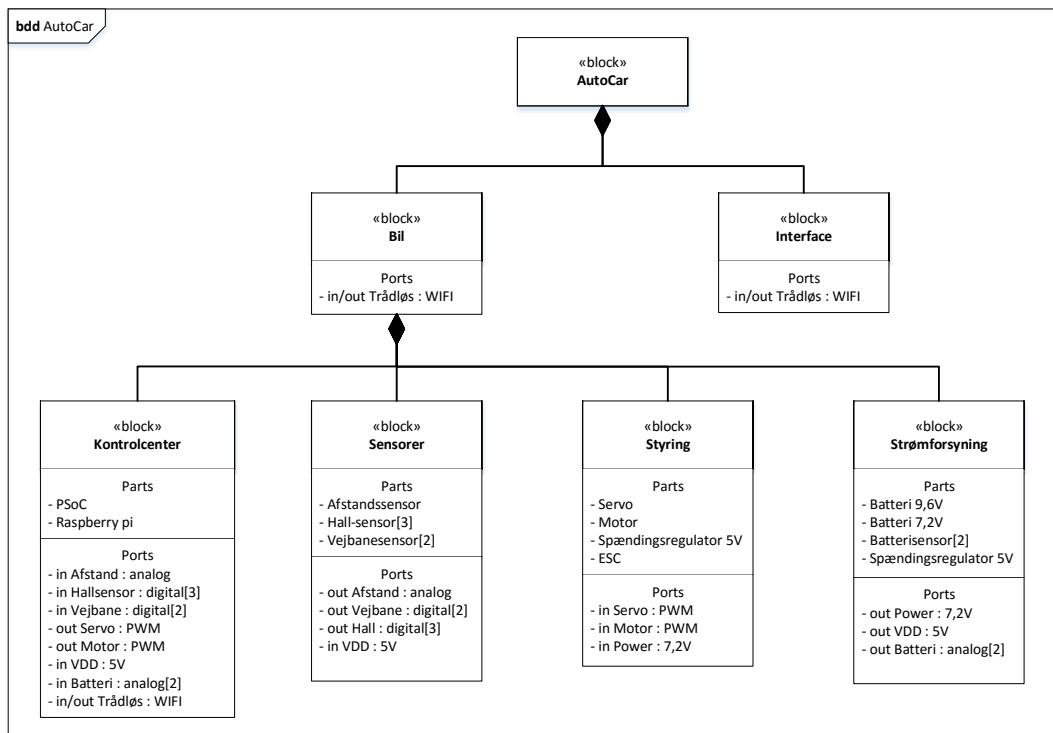
I det følgende afsnit beskrives hardware- og softwarearkitekturen, der er udarbejdet i forbindelse med projektet. Den ligger til grundlag for det videre arbejde det blev udført i designfasen.

6.4.1 Hardwarearkitektur

Hardwarearkitekturen beskrives vha. SysML's BDD og IBD. Der er udarbejdet diagrammer for det overordnede system, samt uddybende diagrammer for hver af systemets blokke.

6.4.1.1 Overordnet BDD

På figur 6.4 ses det overordnede BDD for det samlede system. Det giver et overblik og en grundlæggende forståelse for projektets hardware.



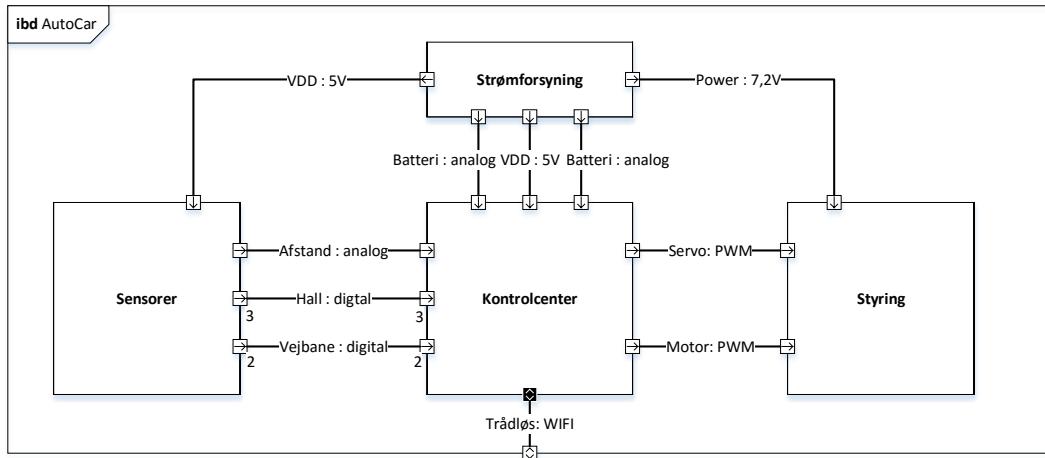
Figur 6.4: Overordnet BDD for AutoCar

AutoCar har to overordnede blokke; Bil, som indebærer kommunikation internt på bilen, sensorinputs samt regulering af servoen og motoren. Interface indebærer grænsefladen mellem systemet og brugeren. Kontrolcenter består af to dele: En Raspberry Pi og en PSoC. Raspberry Pi'en tager alle beslutningerne ud fra de informationer PSoC'en får fra sensorerne. Systemet har seks sensorer: En afstandssensor, tre Hall effekt-sensorer og to vejbanesensorer. Forhjulene bliver styret af en servo, mens motoren styrer farten på

bilen. Strømforsyningen forsyner systemets dele med angivne spænding.

6.4.1.2 Overordnet IBD

Ud fra BDD'et udarbejdes et IBD. Det overordnede IBD er vist på figur 6.5. Det giver et overblik over forbindelserne mellem systemets hardware-blokke.



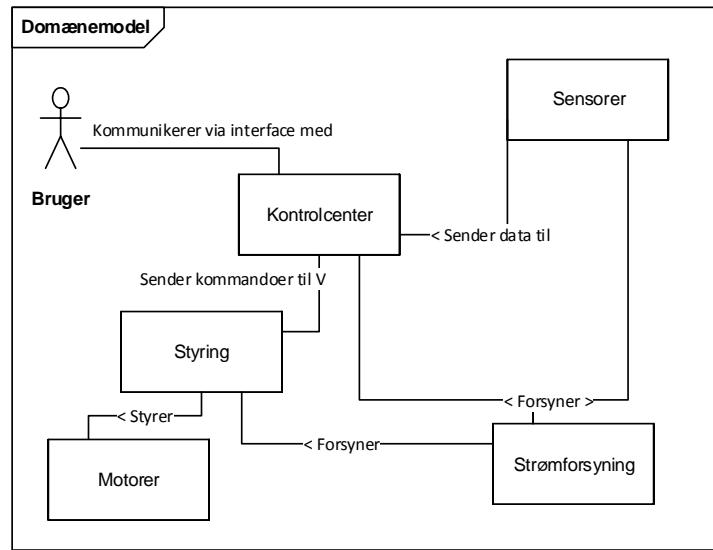
Figur 6.5: Overordnet IBD for AutoCar

6.4.2 Softwarearkitektur

Softwarearkitekturen beskrives vha. UML's domænemodel, applikationsmodel og sekvensdiagram.

6.4.2.1 Domænemodel

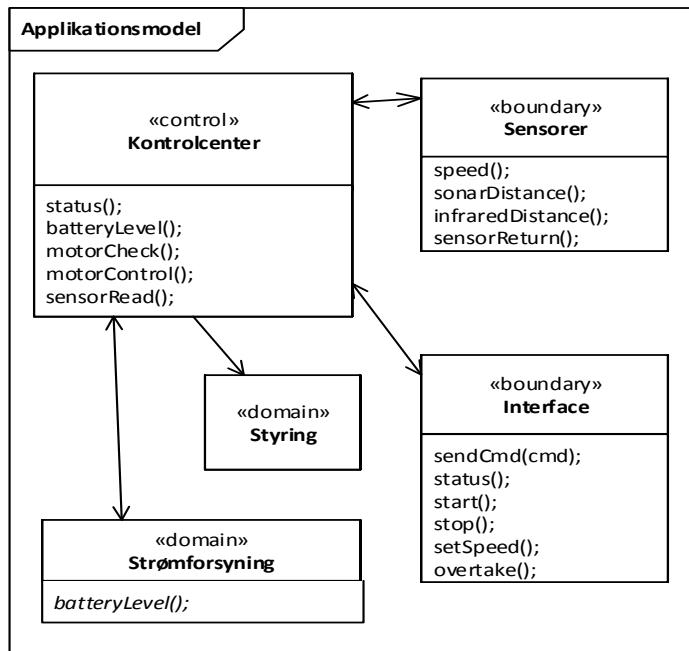
Domænemodellen viser hvordan systemets blokke kommunikerer, og hvilken vej kommunikationen foregår. Domænemodellen for systemet ses på figur 6.6.



Figur 6.6: Domænemodel

6.4.2.2 Applikationsmodel

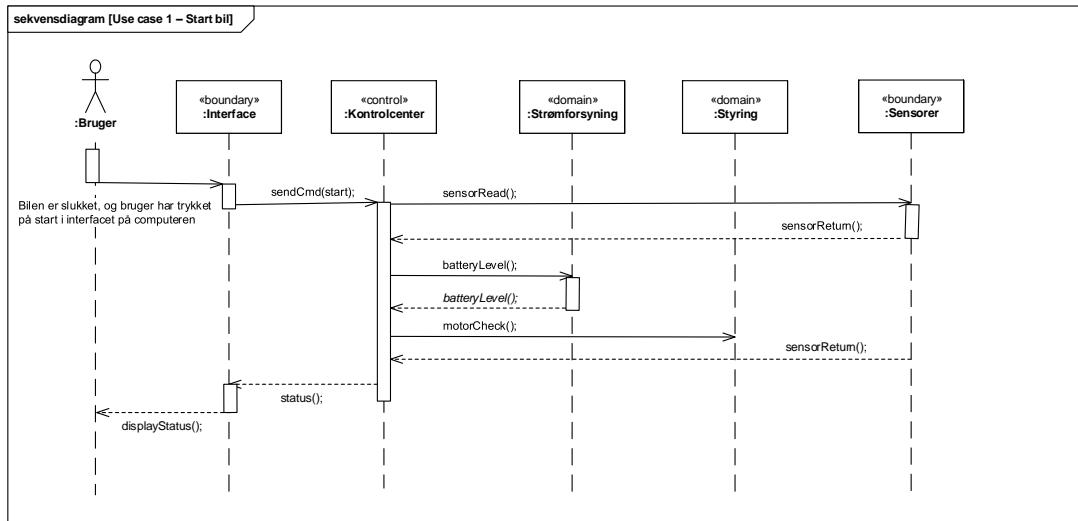
Ud fra domænemodellens klasser, udarbejdes en applikationsmodel for det samlede system. Der opereres med tre forskellige klasser – boundary, control og domain – og det angives hvilket lag de opererer på. Det er henholdsvis grænseflade-, logik- og datalaget. På figur 6.7 ses applikationsmodellen for systemet.



Figur 6.7: Applikationsmodel

6.4.2.3 Sekvensdiagram

Sekvensdiagrammet laves for hver use case, med blokkene fra domæne- og applikationsmodellen. Det beskriver forløbet og kommunikationen mellem blokkene med funktioner, for at overskueliggøre software udviklingen. På figur 6.8 ses sekvensdiagrammet for use case 1.

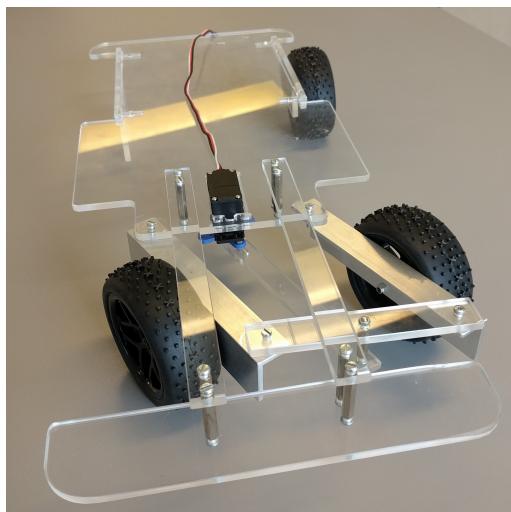


Figur 6.8: Sekvensdiagram - Use case 1 - Start bil

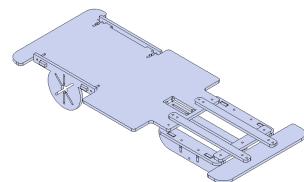
7 Hardware - design, implementering og test

7.1 Chassis

Det blev besluttet at der skulle udvikles et chassis til vores bil, frem for at købe et, da køb chassis lå uden for budget.



(a) Den færdige bil



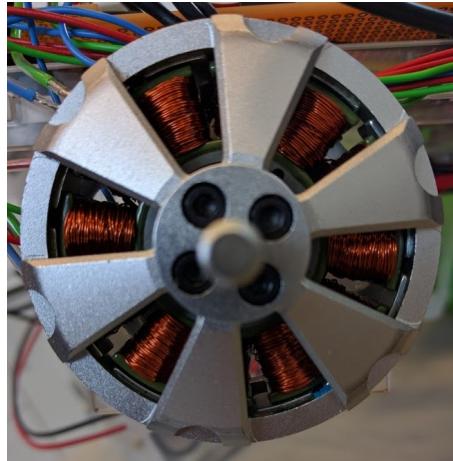
(b) Bilen som designet på computer

Figur 7.1

Der er gjort overvejelser om hvorvidt bilen skal kunne dreje ved hjælp af en servo, uden alt for meget mekanisk kompleksitet. Der er gjort plads til at der på det ene baghjul kan placeres en motor direkte på hjulet, igen for at undgå mekanisk kompleksitet. Der er desuden sørget lavet en bundplade i chassiset hvor der er plads til at montere diverse kredsløb, det samme er der gjort plads til på den plade som er monteret i fronten.

7.2 ESC

Der er valgt en Brushless DC-motor (BLDC-motor) til fremdrift af bilen. Den er valgt da BLDC-motorer har højere effektivitet og højere holdbarhed. Dette skyldes at BLDC-motorer ikke har brushes. I stedet er spolerne monteret radialt og er omgivet af permanentmagneter med skiftende pol-orientering. Det er permanent-magneterne der roterer[9].

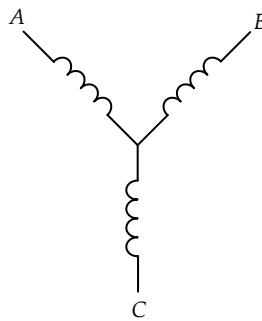


Figur 7.2: Billede af den valgte motor til bilen

Motorens position skal være kendt for at sikre jævn omdrejning af motoren. Motorpositionen findes med tre Hall-effekt sensorer. Fordelen ved at bruge disse sensorer er at positionen altid er kendt. Ulempen er at hver sensor skal have tre ledninger, men tachometeret er implementeret som en Hall-effekt sensor, så kun to yderlige er nødvendige. Ved at bruge disse sensorer er positionen altid kendt og der er ikke brug for en speciel startprocedure. Alternativet er back-EMF, men dette kræver en open-loop start og fungerer ikke så godt ved lavt omdrejningstal[10]. Hall-effekt sensorerne giver højt output halvdelen af tiden og lavt resten af tiden pga. de skiftende pol-orienteringer. Sensorerne er monteret 120° fra hinanden imellem spolerne da dette giver seks unikke kombinationer af input. Ved test af output fra sensorerne ved omdrejning af motoren er en af sensorne altid høj. Det betyder at der ikke er seks unikke positioner, hvilket skaber problemer for softwaredelen af ESC. Denne motor er valgt da den fungerer ved lavt omdrejningstal for at undgå brug af gear. Der er derfor valgt en motor med en lav Kv-værdi svarende til en lav omdrejningshastighed.

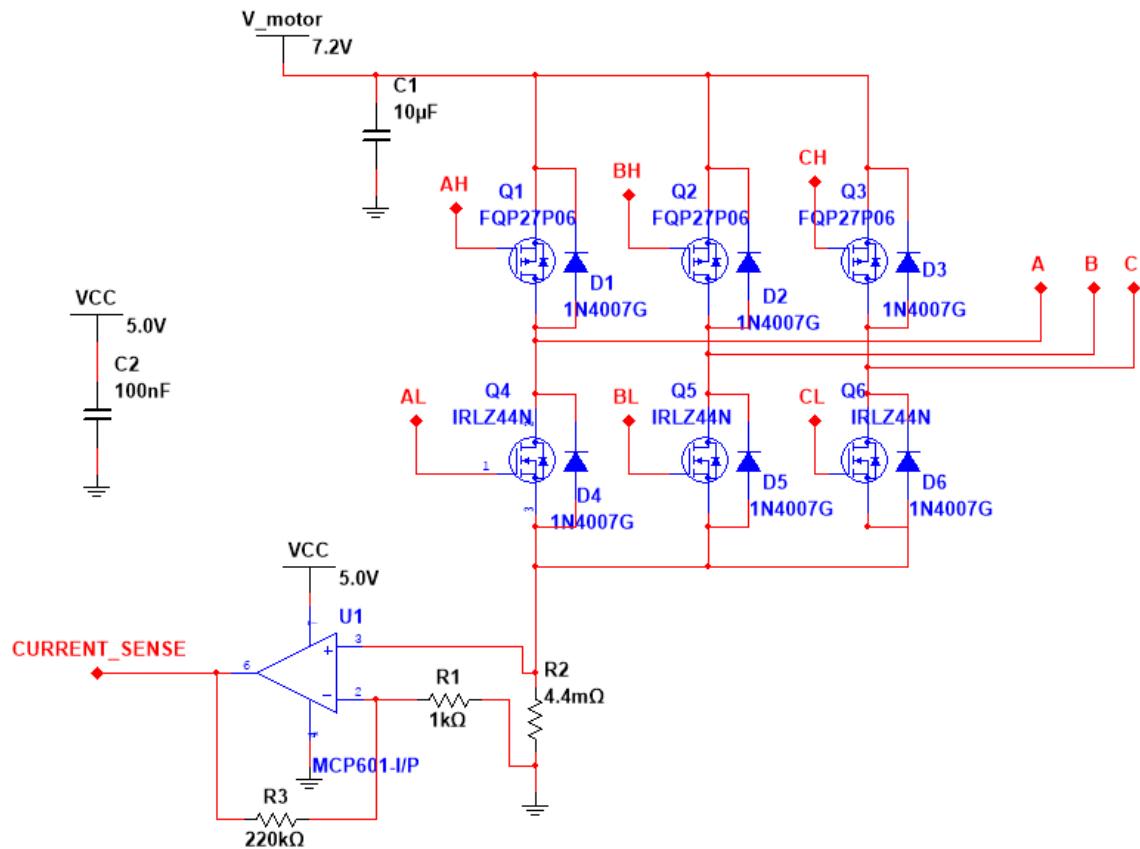
7.3 Effektkredsløb til motoren

Den valgte motor indeholder tre faser, hvorpå polariteten bestemmer hvilken fase er aktiv. Der er således udviklet et kredsløb, som kan skifte polariteten over motorens fase samt leverer den nødvendige strøm.



Figur 7.3: Princippet bag motoren. A, B og C er de forskellige faser.

Figur 7.3 viser en funktionel tegning over motoren. Det skal være muligt for hver enkelt fase at kunne sætte positiv eller negativ polaritet, og det være muligt for en fase at være svævende. Når der løber en strøm gennem en spole vil der induceres et magnetfelt, som vil gøre at motoren retter sig ind efter de påmonterede permanent-magneter. Kredssløbet der er udviklet virker i principippet som en 3-faset inverter. Det er dimensioneret efter at kunne leve 5 A.



Figur 7.4: Effektkredssløb

Figur 7.4 viser effektkredssløbet. I udgangstrinnet til motoren er der valgt N-kanal og P-kanal MOSFET-drive. Der er monteret kølefinner på udgangstransistorerne, således de vil kunne leve den ønskede strøm og stadig være indenfor et sikkert termisk om-

råde. Der er monteret en shunt-modstand i kredsløbet (R2), som gør det muligt at måle strømmen der løber gennem motoren. Dette anvendes i styringen til at begrænse hvor meget motoren bliver drevet.

Idet MOSFET-transistorernes gate i principippet er en kondensator som skal op- og aflades, så er der implementeret et gate driver kredsløb¹. Formålet med kredsløbet er at forbedre tiden det tager at oplade og aflade gaten og derved tænde og slukke transistoren. Dette er gjort med et push-pull BJT-forstærkertrin, således at der både kan oplades og aflades ladninger på kort tid.

Der er blevet gjort nogle EMC-overvejelser i forbindelse med kredsløbet. Idet der er switches på motoren, som i principippet er opbygget af spoler, vil der opstå transiente i forbindelse med spolernes følsomhed over for ændring i strøm ($\frac{di}{dt}$). Der er monteret dioder, som begrænser størrelsen af denne transient. Idet der kan løbe forholdvis stor strøm igennem kredsløbet, er der gjort overvejelser om hvordan man kan minimere strømsløjfer i kredsløbet. Strømsløjfer udsender elektromagnetisk støj, som er proportionel med arealet af strømsløjfen. EMC-relaterede overvejelser er beskrevet nærmere i EMC-rapporten som kan findes i bilaget.

7.4 Spændingsregulator

Der er lavet reguleringer på de to batteriers spændinger. Der skal bruges 5V til PSoC, Raspberry Pi og sensorer. Servoen skal også bruge 5V. Der er valgt et 7.2V batteri til servo- og brushless motor. Reguleringen til 5V sker ved brug af en LM7805, da der max skal bruges 700mA til servoen. LM7805 kan leve op til 1A[11]. Til at forsyne alt logikken, er der brugt en switch-mode regulator, da den valgte kan leve op til 1.5A[12]. Det forventes, at Raspberry Pi'en kan bruge op til 1 A. Samtidigt skal der være nok strøm til PSoC og sensorerne.

7.5 Servo

Servoen bruges til at dreje bilens forhjul. Der er valgt en servo, fordi den drejer mellem to yderpunkter, hvilket gør det muligt for AutoCar at dreje. Den styres ved at sende et PWM-signal på signalledningen. PWM-signalet skal have en frekvens på 50Hz, med en duty-cycle mellem 5% og 10%. Ved en duty-cycle på 5% drejer servoen til venstre og ved en duty-cycle på 10% drejer den til højre.

7.6 Sonar

Til afstandsbestemmelse for selvkørende biler vælges der typisk lidar- eller radar-teknologi[13], men grundet budgetbegrænsninger for AutoCar-projektet blev sonar valgt som afstandssensor. Kort sagt udsender en sonar en lydbølge, og denne bølge reflekteres af objekter i dens vej. Sonaren mäter hvor lang tid der er gået før den hører ekkoet fra sin udsendte lydbølge, og beregner afstanden ud fra følgende ligning:

¹Kredsløbet ikke vist i rapporten. Kredsløbet kan findes i dokumentationen under ESC afsnittet.

$$d = \frac{\Delta t \cdot V_{lyd}}{2}$$

Der divideres med to da bølgen skal både frem og tilbage fra objektet. Den valgte sonar er en LV-MaxSonar-EZ MB1010[14], som kan afstandsbestemme med høj præcision fra 20 cm til 6.45 m. Sonaren har forskellige muligheder for data-transmission, og analog spænding blev valgt, da den virkede mest præcis og interessant ift. EMC. Der er flere forskellige EMC-overvejelser med sonaren, primært da data-ledningen er modtagelig over for de høje strømændringer der kommer på motorledningerne. I EMC-rapporten blev det vist, at hvis data-ledningen fra sonaren var bare 7 cm fra motorledningen under en strømændring ville sonarresultatet læse 1 cm forkert, hvilket ikke er ubetydeligt. Derfor er det vigtigt, at adskille sonarens analoge output fra motorledningerne, når systemet samles på AutoCar. Databladet for sonaren angiver en konverteringsfaktor fra spændingen på outputtet til centimeters afstand på

$$d = \frac{V_{ADC}}{3.86 \frac{mV}{cm}}$$

Denne konvertering ikke var helt præcis. Derfor blev der lavet en lineær regression til, at give et bedre og mere præcist estimat. Denne regression kan ses i sonarens hardware-afsnit i dokumentationen.

7.7 Vejbanesensor

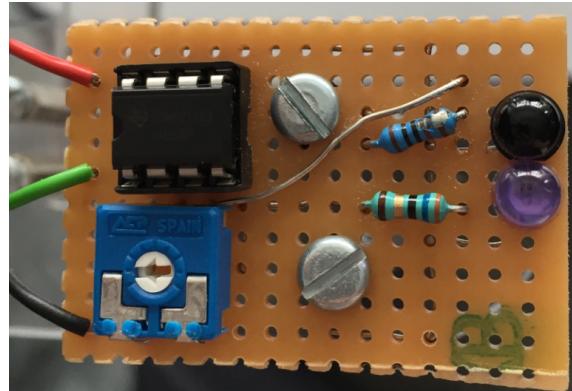
Til at få bilen til at holde sig inden for vejbanestribene, er der valgt IR-sensorer. Det er valgt ud fra principippet om "follow line robot"[15], da de minder en del om hinanden. Computer vision er en anden mulighed, som var oppe at vende. Det blev dog fra starten vurderet, at der ikke ville være tid nok til gruppen kunne nå at udvikle det sammen med de andre moduler.

Ved IR-sensorer bruges en infrarød LED, hvilket passer godt til en vej i virkeligheden. Dens stråle vil blive absorberet af mørke farver som asfalt, mens den bliver reflekteret af lyse farver som vejstriben. Samtidig er der brugt en fotodiode der skærmer for sollys, hvilket er essentielt på en rigtig motorvej[16].

Derudover er det valgt, at lave outputtet fra sensoren om til et digitalt signal. Det er smartest, da der kun ønskes to forskellige værdier. 1 når sensoren opfanger en vejbane, og 0 når den ikke gør. Det er altså ikke nødvendigt at kende den præcise spænding fra kredsløbet, som vil være tilfældet ved et analogt output. Derfor ses et digitalt signal som den naturlige løsning.

Der er forholdsvis stor forskel på hvor tæt sensoren skal være på underlaget, før den kan detektere forskellige lyse farver. Testunderlaget der har været brugt er heller ikke helt mørkt, og derfor vil sensoren også kunne opfange det, hvis den er indstillet forkert. Derfor er der indsat et potentiometer, som kan kalibrere sensoren til lige netop at opfange den vejbanefarve, der arbejdes med, men samtidig ikke detektere selve vejen.

Den færdige IR sensor ses på figur 7.5.



Figur 7.5: Implementeret vejbanesensor

8 Software - design, implementering og test

8.1 SPI kommunikation

Til kommunikation mellem PSoC og Raspberry Pi er der valgt SPI. Dette er gjort for at udnytte SPI's mulighed for at sende full duplex. Der er kun en modtager og en sender, hvilket betyder at de ekstra ledninger sammenlignet med I2C ikke er et problem. Raspberry Pi er valgt som master og PSoC som slave, da det er Raspberry Pi der sender besked til PSoC'en om hvad bilen skal gøre, og Raspberry Pi skal modtage sensordata.

Der er udviklet en SPI-protokol hvor Raspberry Pi skal bruge PSoC'ens seks datainputs fra sensorer og en afslutningsbit. Raspberry Pi sender fartindstilling, servoretning og start/sluk til PSoC. De fire resterende bits er altid ens og bruges af PSoC'en i en sorteringsalgoritme, så rækkefølgen på PSoC'en altid er rigtig og kommandoerne fra Raspberry Pi kan findes det rigtige sted. Raspberry Pi er master og sender alle syv SPI-beskeder 10 gange hvert sekund.

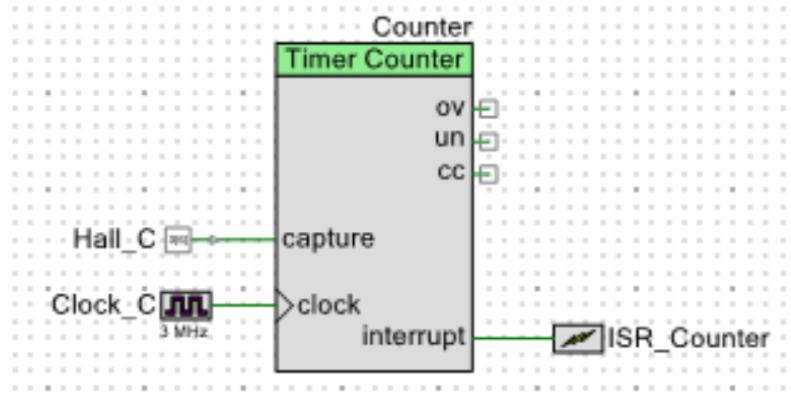
På PSoC'en er SPI implementeret vha. det indbyggede SPI-modul og sat op til at gemme beskeder når alle syv beskeder er modtaget fra Raspberry Pi. På Raspberry Pi er SPI master implementeret vha. bcm2835 bibliotek der styrer Raspberry Pi's GPIO ben. Der er implementeret en duplex send_read funktion som samtidig læser fra og skriver til SPI.

8.2 Tachometer

Til at måle bilens fart bruges en Hall effekt-sensor som tachometer. Det implementeres i PSoC'en vha. en timer counter. Den bruges til at beregne frekvensen af signalet fra Hall effekt-sensoren. Det bliver behandlet i funktionen:

```
1 float speedMeasure();
```

Den returnerer farten i en float. Timer counteren bestemmer perioden ved at gemme de compare-værdier, der skaber et interrupt. Ved at finde forskellen mellem to følgende compare-værdier, kan man bestemme periodetiden ud fra clock-frekvensen, der er brugt som sample-frekvens.



Figur 8.1: Top design for tachometer

8.3 Sonar

Softwaredesignet tilknyttet sonaren er implementeret på PSoC'en. Det analoge output er forbundet til en ADC-indgang på PSoC'en. Ved at sætte ADC'en til at tage gennemsnitsværdier af mange målinger, som er muligt på grund af den høje sample rate, er der en højere præcision og færre fejlmålinger. ADC'en har tilknyttet et interrupt, så sonaren værdi opdateres med en frekvens på 1kHz i koden. ADC'en har en opløsning på 12 bits. Da sonaren skal kunne måle mellem 20 cm og 250 cm, er den højeste spænding der forventes at blive målt på ADC'ens indgang

$$250\text{cm} = \frac{V_{\text{ADC},\text{max}}}{3.787 \frac{\text{mV}}{\text{cm}}} - 2.561\text{cm} \implies V_{\text{ADC},\text{max}} \cong 956.45\text{mV}$$

12 bit er dermed mere end nok til at behandle dataen inden for de angivne afstandsgrenser. Ved hver SPI-exchange med Raspberry Pi oversættes sonarens afstandsværdi til en enkelt byte. Den har stadig en opløsning på 1 cm, hvilket er fem gange højere end kravspecifikationen angav. Det er altså lykkedes, at få sonarafstanden til at have en høj opdateringsfrekvens og præcision.

8.4 Vejbanesensor

Softwaredelen til vejbanesensoren er delt op i to dele – monitorering og regulering. Det er delt op i to funktioner, en der videresender data fra sensorene, og en der omregner signalet fra Raspberry Pi til et brugbart PWM signal. Der er opstillet to formler for beregning af PWM signalet, en for at dreje til højre og en for at dreje til venstre. Det blev gjort fordi servoens midterpunkt ikke er lineært i forhold til de to yderpunkter.

Nedenfor ses formlen der bruges, hvis bilen skal dreje til venstre. "dir" er den værdi der modtages fra Raspberry Pi, som omregnes til PWM-signalet. Det ligger i intervallet -100 til 100, hvor -100 er helt til venstre og 100 er helt til højre.

```

1 if (dir <= 0 && dir > -100)
2     PWM = 3*dir + 1300;

```

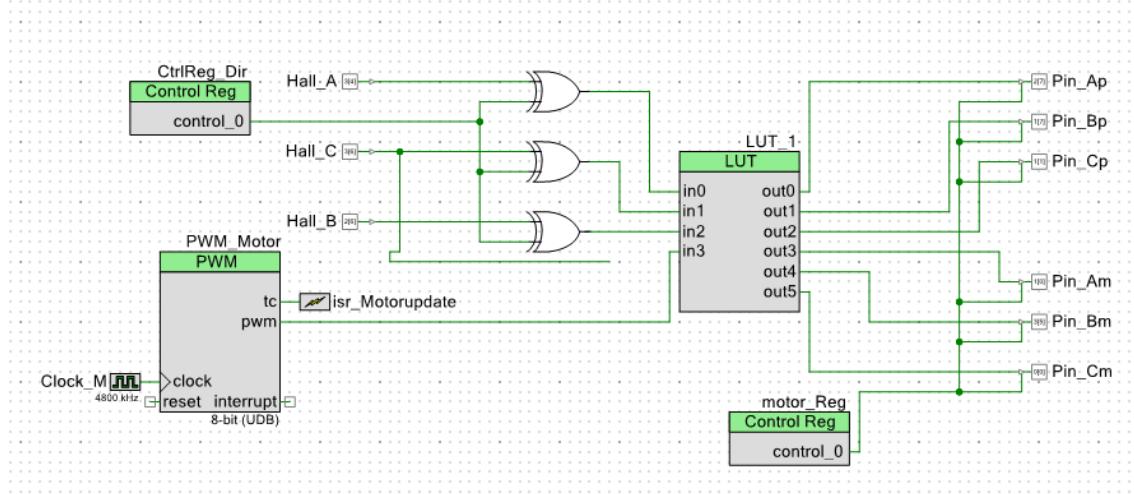
Til sidst udskrives PWM-signalet til servo'en med PSoC funktionen:

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

Funktionen ændrer compare-værdien i PWM-blokken, som bestemmer hvilken værdi der sker en falling edge ved.

8.5 ESC

Motorstyringen på PSoC er baseret på regulering af motorens position vha. Hall effekt-sensorer. Ved brug af tre sensorer skal der kunne måles på seks unikke positioner. Ud fra de seks positioner kan outputtet bestemmes, som er input til ESC-kredsløbet.



Figur 8.2: Motorstyring implementeret med lookup-tabel og PWM-styring af hastighed. Input er Hall effekt-sensorerne og PWM til hastighedsstyring. De tre øverste outputs er for at sætte faserne til høj spænding og de tre nederste er for at sætte faserne til lav spænding.

Da output skal følge input er det implementeret med en lookup-tabel. For at styre hastigheden er input til lookup-tabellen også et PWM-signal som genererer et PWM-signal på de positive udgange til ESC-kredsløbet. Perioden på PWM-signalet styrer hastigheden på motoren. For at implementere en bakfunktion er sensorinputtet xor'et med et register så inputtet kan inverteres.

Til regulering af motorhastigheden er der implementeret en PI-controller til at regulere PWM-signalet. Proportionsleddet giver en statisk fejl så der er implementeret et I-led for at fjerne den stationære fejl fra P-leddet.

```
1 ...
2 pTerm = pGain*error;
3 ...
4 iTerm = iGain*integral;
5 ...
6 control = pTerm + iTerm; // + dTerm;
7 ...
```

Motoren er testet med PI-controlleren. Motoren har svært ved at dreje rundt ved lave hastigheder. Dette kan skyldes at input fra Hall effekt-sensorerne ikke er som ønsket.

8.6 Raspberry Pi

For at styre bilen og for at kommunikere med GUI, er der brugt en Raspberry Pi. Raspberry Pi'ens primære formål er at fortælle PSoC'en hvad den skal få bilen til at gøre. Raspberry Pi'en skal beslutte hvad der skal ske ud fra hvad den får fra GUI'et. Det er selvfølgelig en bruger der på GUI'et i sidste ende har bestemt hvad bilen skal gøre.

Vi har valgt at bruge en Raspberry Pi 2b, da den bruger mindre strøm end de nyere og større Raspberry Pi's. Vi har også valgt den frem for andre producenter, da vi så sikrer os at der er god dokumentation. Vi kunne også have valgt en Raspberry Pi Zero, men den var ikke tilgængelig da vi skulle bruge den.

Der er forskellige funktionaliteter som Raspberry Pi'en står for. Den kan ud fra sensordataen sørge for at bilen f.eks. følger vejen eller overhaler. For at implementere denne funktionalitet er der lavet tre klasser: En TCP-klasse, en SPI-klasse og en Car-klasse.

Raspberry Pi'ens TCP-klasse står for kommunikation til GUI'et. Det som bliver komunikaret bliver vurderet af Raspberry Pi, og der bliver udført de funktioner eller opdateret de variable som passer til den pågældende besked.

Listing 8.1: Eksempel på hvordan der vurderes på besked over TCP

```

1 ...
2 else if(command.substr(0,13).compare("03QhastighedQ") == 0)
3 {
4     int speedInt = atoi(command.substr(13,3).c_str());
5
6     if(speedInt > 26){
7         autoCar.setSpeed(26);
8         writeTextTCP("13QhastighedQ2002W");
9     }else if(speedInt < -26){
10        autoCar.setSpeed(-26);
11        writeTextTCP("13QhastighedQ2002W");
12    }else{
13        autoCar.setSpeed(speedInt);
14        writeTextTCP("13QhastighedQ2001W");
15    }
16 }
17 ...

```

I koden 8.1 kan man se hvordan der bliver trukket de værdier ud som betyder noget for bilen når der modtages en hastighedsbesked fra TCP-klienten. Desuden bliver der også lavet tjek på om den værdi, som er modtaget, er inden for de opsatte grænser.

Som det også kan ses er der et autoCar-objekt som er af klassen car. car-klassen beskriver alt den funktionalitet bilen har, som ikke indebærer kommunikation. Det vil sige at det er car-klassen hvor følg vejbane-funktionaliteten er placeret og det er også der

overhalings-funktionen ligger. De variabler som car-klassen sætter læses af SPI-klassen og sendes til PSoC'en.

Vi har valgt at implemente koden på Raspberry Pi'en med to tråde som kører parallelt, en med SPI og en med TCP. SPI-tråden står desuden for at kalde car-klassen, og det er derfor SPI-tråden der bestemmer hvor hurtigt der variablerne opdateres, da der er sat en begrænsning på hvor hurtigt der sendes over SPI. Vi har valgt at sende med 10Hz for at sikre at PSoC'en ikke bliver overvældet.

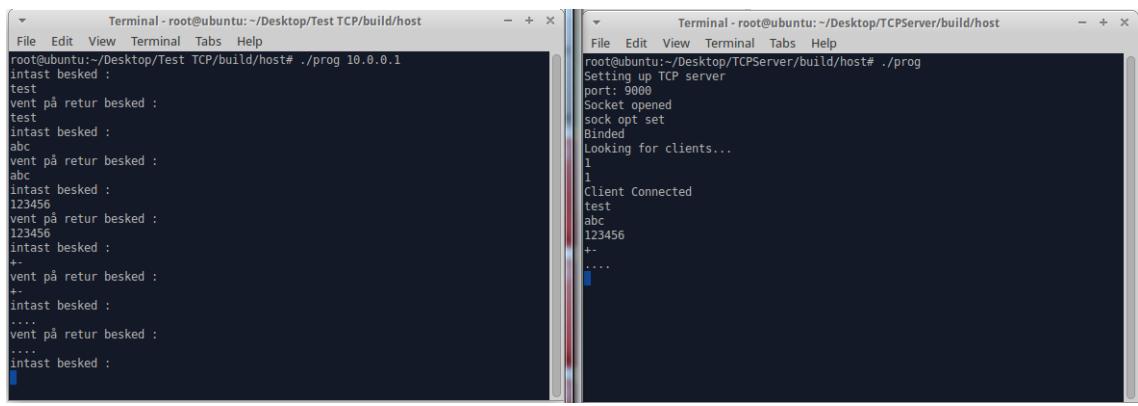
8.7 TCP PC

TCP på PC'en har, som det er implementeret, to funktioner. Dens primære opgave er at sende og modtage data fra Raspberry Pi'en og at sende de korrekte statuskoder derfra. Statuskoderne har til opgave at formidle data til og fra GUI'et, og det skal også kunne dekode beskederne sendt fra Raspberry Pi'en jævnfør TCP-protokollen.

TCP er valgt fordi det er en almen brugt internet protokol som indeholder fejltjek og kontrol af mistede data. Dette gør det til en yderst pålidelig protokol at bruge til trådløs kommunikation.

Det er valgt at bruge socket-programmering til at designe TCP, og dette betyder at der designes et applikationslag og kun til dels transportlaget. Dette er valgt da vores viden om design af linklaget og transportlaget først er blevet fyldestgørende hen mod slutningen af projektet, og socket fuldt ud opfylder de krav vi har til programmering af kommunikation. Design af dette kan ses i dokumentationsafsnittet om TCP PC.

Figur 8.3 viser en test af TCP-forbindelsen mellem to virtuelle Linux-maskiner, som tester at selve forbindelsen virker uden protokol. På højre side ses client-koden, som ligger bag GUI'et på PC'en, og til venstre ses serveren som er sat op som en Echo-server dvs. at alt den modtager sendes tilbage. På denne måde kan både send og modtag testes på klienten på en gang, uden at brugerne skal interagere med server-siden.



```
Terminal - root@ubuntu:~/Desktop/TestTCP/build/host
File Edit View Terminal Tabs Help
root@ubuntu:~/Desktop/TestTCP/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: 9000
Socket opened
sock opt set
Binded
Looking for clients...
1
1
Client Connected
test
abc
123456
+-
....
```

Figur 8.3: Test af TCP PC

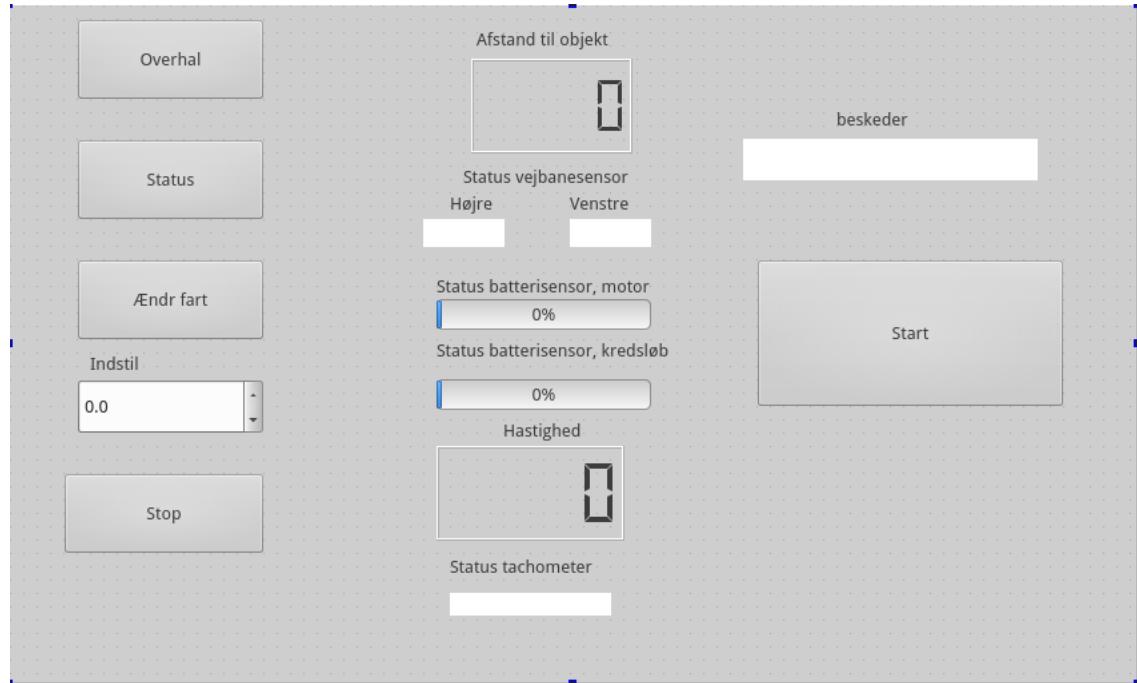
TCP'ens sekundære funktion er at sende beskeder til og dekode beskeder modtaget fra serveren.

Send er implementeret med send-funktioner som sender en kode jævnfør protokollen alt efter opgaven. Send-funktionen venter også på svar fra Raspberry Pi og gemmer dette

i en attribute. For at dekode beskederne er der udviklet get-funktioner som ud fra attributen returnerer en værdi alt efter dens opgave. Mere om implementeringen af dette kan læses i dokumentationen under TCP PC.

8.8 GUI

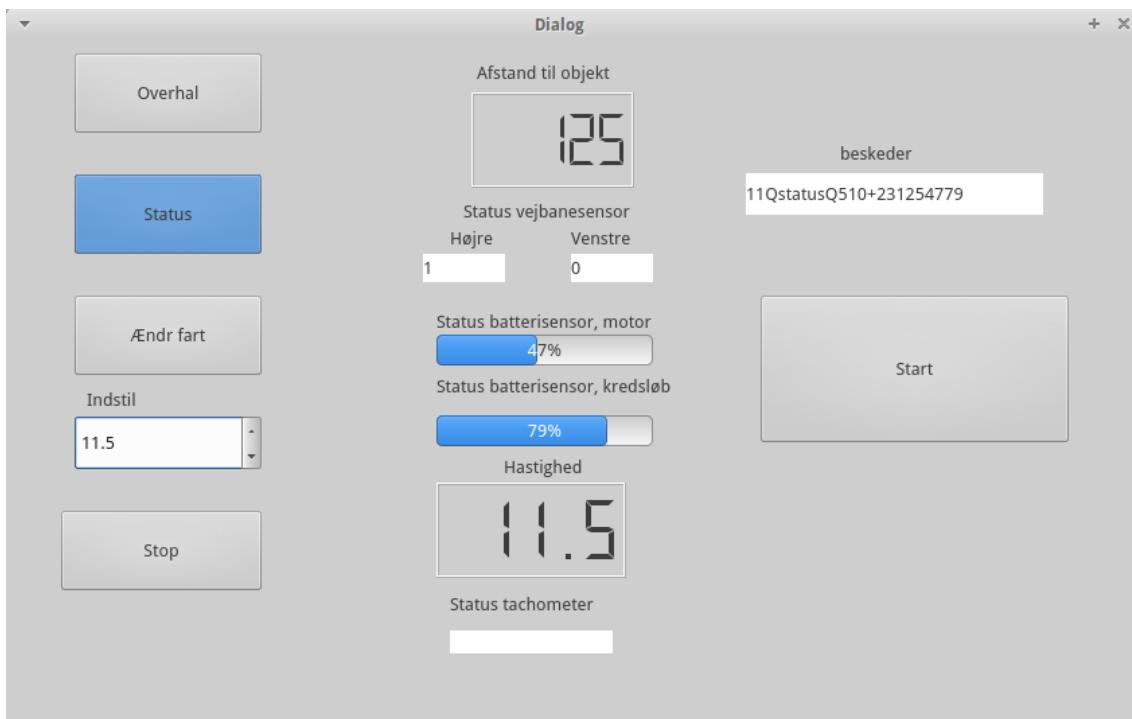
Den grafiske brugergrænseflade (GUI) er udviklet i QT Creator og skrevet i C++. Det har til opgave at stå for alt interaktion med brugeren.



Figur 8.4: Forbindelsesvindue i GUI

Figur 8.4 viser GUI'ets primær-vindue, det er udviklet så alt funktionelt ligger i dette vindue. Det er muligt at få udskrevet data fra alle sensorer, starte og stoppe bilen, og ændre hastigheden. Alle knapper og menuer kalder en funktion i TCP-klassen.

Det er valgt at ligge al funktionalitet i samme vindue for at gøre design-processen simplere. Konsekvensen af dette er at det muligt at vælge både start og stop selv om bilen er i denne tilstand.



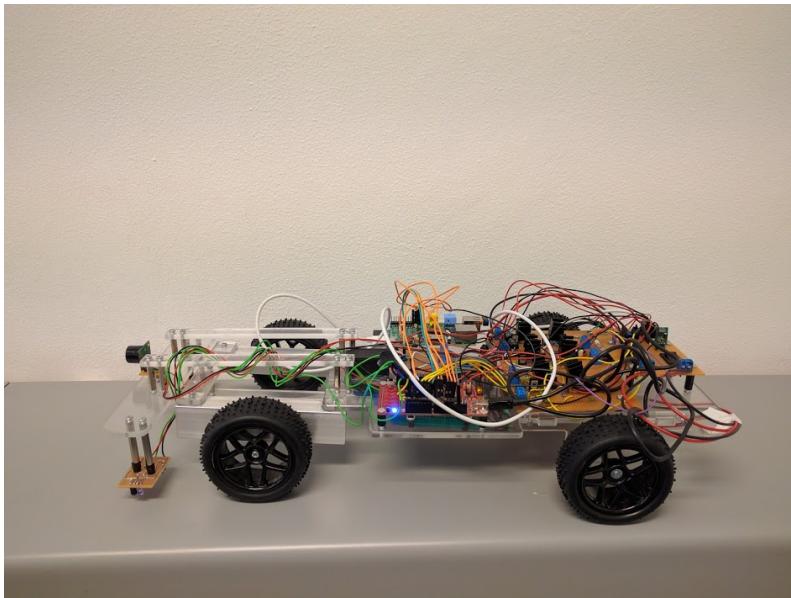
Figur 8.5: Aktivt GUI-vindue

Figur 8.5 viser GUI'ets udskrift med simulerede data fra en virtuel Linux-maskine, for at demonstrere den fulde funktionalitet af GUI'et.

9 Resultater og diskussion

Projektarbejdet med AutoCar er mundet ud i en funktionel demonstrator, hvor alle enheder fungerer.

Nogle af kravene er overholdt og opfyldt. Systemet har sensorer og aktuatorer, et GUI, og trådløs kommunikation over WiFi. Elementer fra semestrets fag er i høj grad integreret i projektet, især ift. ESC'en og TCP.



Figur 9.1: Koncept for AutoCar

Blinklys og kørelyser er ikke implementeret, da det vurderes at andre moduler var vigtigere for systemets funktion. Ligeledes er batterisensorerne ikke implementeret, selvom al koden er skrevet dertil, og er med i klassediagrammet. Ingen blev det vurderet at anden funktionalitet havde højere prioritet end disse.

Der er implementeret en brushless DC-motor der, trods besværligheder i designfasen, til sidst fungerede godt. Det blev valgt tidligt at gruppen selv ville implementere Hall effekt-sensorerne til motorstyring og tachometer, da det er en større udfordring og giver mere erfaring end at købe sig til en færdig løsning.

PI-controlleren til regulering af motoren hastighed fungerede også til sidst, trods lidt uforudsigelig opførsel og begrænsninger. Controlleren ramte ofte sine poler og blev ustabil under designfasen, især ved lave hastigheder. Det var svært at opstille modeller for motorkredsløbet eller brute-force kalibrere controlleren, men ved at prøve sig frem blev der alligevel fundet en fungerende løsning ved hastigheder over 8 km/t med bestemte gain-værdier.

Der er implementeret et fungerende GUI som giver brugeren adgang til de fem use cases der initieres af brugeren. GUI'et kommunikerer med AutoCar's Raspberry Pi kontrollenhed over TCP med WiFi, og en protokol med data og statusmeddelelser. Disse pro-

tokolmeddelelser kan bl.a. fortælle om fejl på systemet. Raspberry Pi er bindeleddet mellem GUI'et og PSoC'en, der styrer AutoCar's moduler direkte og holder styr på dataen fra begge sider. Kommunikationen mellem Raspberry Pi og PSoC over SPI fungerer også som ønsket efter indsættelse af en sorteringsalgoritme på PSoC'en.

Der er implementeret sonar- og tachometer-sensorer der holder styr på afstanden til objekter og motorens hastighed ud fra Hall-effekten. Begge moduler opfylder kravene, og sonaren har endda højere oplosning og præcision end specificeret.

Der er implementeret en servomotor og to vejbanesensorer, der sammen har til opgave at holde bilen inden for vejbanerne. Begge moduler er velfungerende, men det viser sig at designet af vejbanesensorne ikke har den effekt vi ønskede fra systemarkitekturen. Det er svært at regulere på to relativt små områder foran bilen, og derfor svært at holde bilen inden for en vejbane uden at den slingrer for meget. Desuden var det pga. motor-designet og controlleren meget svært at opnå en lav hastighed på motoren, hvilke gjorde det endnu sværere for vejbanesensorne at regulere bilens position i realtid.

9.1 Fremtidigt arbejde

Havde gruppen haft mere tid til udvikling, så kunne mange af problemstillingerne været behandlet mere detaljeret. Lys og batteri-sensorer kunne bl.a. være implementeret. Den oplagte løsning til at holde AutoCar inden for vejbanerne er computer vision, som kunne have outputtet en afstand til vejen i begge sider der så kunne reguleres med en dertil egnet controller. Computer vision er en meget kompliceret og tidskrævende proces, men ville kunne løse mange af problemerne. Med mere tid kunne man også implementere nogle mekaniske forbedringer for at reducere den interne friktion i hjulene, og lave motorgearing. Friktionen kunne reduceres med f.eks. kuglelejer og ved at forbedre motor-styring og introducere gearing kunne dette modul være mere stabilt ved lave hastigheder og nemmere at styre. En reconnect-funktion på GUI'et og en timeout ville også gøre PC-siden mere robust.

10 Konklusion

Målet med projektet var at udvikle en demonstrator for et bilsystem med intelligente sensorer og aktuatorer. Bilen skulle give mulighed for en bruger at indstille farten samt at bestemme bilens adfærd på en vejbane.

Der er udviklet på en demonstrator, som udfylder nogle af de specificerede krav. Kravene for brugerens interaktion for systemet er ikke implementeret, men de vigtigste krav for bilens operation er implementeret i et hvis omfang. Der er udviklet et distribueret system, hvorpå der er mulighed for regulering af motor, sensorering og styring af modelkøretøj samt pålidelig kommunikation.

Det er lykkedes at implementere reguleringen af en brushless motor. Det er muligt at indstille en fart, som systemet regulerer sig ind efter. I systemets nuværende tilstand er reguleringen dog ustabil i forhold til ønsket operation.

Optisk sensorering samt en strategi for styring er implementeret. Det er blevet erfaret, at den nuværende opstilling med to enkelte optiske vejbanesensorer ikke er tilstrækkelig. Videre udvikling med eventuel computer vision er tiltænkt.

Der er implementeret funktionel og pålidelig kommunikation fra computer til bilens indlejrede platform. Det er muligt for brugeren at kommunikere med systemet samt at indstille ønskede parametre, med enkelte dele af funktionaliteten ikke implementeret.

Litteraturliste

- [1] Mads Elkær. "Audi-chef: Vi har selvkørende biler om få år". I: *Computerworld* (18. jun. 2015). URL: <http://www.computerworld.dk/art/234175/audi-chef-vi-har-selvkørende-biler-om-faa-aar>.
- [2] Evan Ackerman. "Tesla Model S: Summer Software Update Will Enable Autonomous Driving". I: *IEEE Spectrum* (23. maj 2015). URL: <http://spectrum.ieee.org/cars-that-think/transportation/self-driving/tesla-model-s-to-combine-safety-sensors-to-go-autonomous>.
- [3] Orr Hirschauge. "Are Driverless Cars Safer Cars?" I: *Wall Street Journal* (2015). URL: <http://www.wsj.com/articles/are-driverless-cars-safer-cars-1439544601>.
- [4] Troels Karlakov. "VIDEO Delvist selvkørende lastbiler drøner gennem Danmark". I: *DR Nyheder* (30. mar. 2016). URL: <https://www.dr.dk/nyheder/indland/video-delvist-selvkørende-lastbiler-droener-gennem-danmark>.
- [5] DSDM. *MoSCoW Prioritisation*. 25. maj 2016. URL: <https://www.dsdm.org/content/moscow-prioritisation>.
- [6] Kim Bjerge. "Vejledning til udviklingsprocessen for projekt 2". I: (20. feb. 2015).
- [7] Bruyninc. *V-model-en*. 7. mar. 2008. URL: <https://upload.wikimedia.org/commons/b/b6/V-model-en.png>.
- [8] omgsysml.org. *SysML*. 15. mar. 2009. URL: <http://www.omg.sysml.org/images/Figure-2.jpg>.
- [9] Mat Dirjish. *What's The Difference Between Brush DC And Brushless DC Motors?* 16. feb. 2012. URL: <http://electronicdesign.com/electromechanical/what-s-difference-between-brush-dc-and-brushless-dc-motors>.
- [10] Thomas Freitag. *The end of sensored BLDC control?* 16. apr. 2012. URL: <http://www.ecnmag.com/article/2012/04/end-sensored-bldc-control>.
- [11] FAIRCHILD. *3-Terminal 1 A Positive Voltage Regulator*. Sep. 2014.
- [12] Murata Power Solutions. *Fixed Output 1.5 Amp SIP DC/DC Converters*. 2014.
- [13] 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>.
- [14] MaxBotix inc. *LV-MaxSonar-EZ™.pdf*. 26. maj 2016.
- [15] Buildcircuit. *Easy steps for making a line following robot using Infrared LED, Photodiode, ArduMoto and Arduino*. 26. maj 2016. URL: <http://www.buildcircuit.com/easy-steps-for-making-a-line-following-robot-using-infrared-led-photodiode-ardumoto-and-arduino/>.
- [16] OSRAM. *SFH203FA Datasheet.pdf*.