

BEERTRESS

Projektrapport

Semesterprojekt 3, Gruppe 8

Peter Wann
201907121

August Hjerrild Andersen
201907251

Simon Phi Dang
201705957

Henry Pham
201606071

Alexander Flarup Wodstrup
201810602

Lucas Friis-Hansen
201811527

Jim Sørensen
201602614

Shynthavi Prithviraj
201807198

17. december 2020

Antal tegn med mellemrum: 63248

1 Abstract

The purpose of this project and the final system is to replace a waitress at a restaurant and minimize the risk of infection by Covid-19, which has affected the world in 2020. The project is called Beertress, as the system only will be able to handle beer for now.

Beertress consists of a PSoC and sensors, which makes it possible to navigate around a restaurant. The sensors that will be used are a color sensor used to navigate the track, a distance sensor used to avoid obstacles and a weight sensor that informs the staff to refill Beertress. A motor is used to make the robot drive. Beertress can receive orders from customers through a website, that will be implemented by a RPi and Websocket through this project.

The vision of a robot, that navigates around a restaurant is fulfilled to a certain amount. It has not been possible to accomplish all the goals, which were described in the beginning of the project, due to time pressure. Still, by an order of a customer, Beertress manages to navigate a track.

2 Resumé

Formålet med projektet og det endelige system er, at erstatte en tjener på en restauration, og dermed mindske risikoen for smittespredning af Covid-19, som har præget verden i 2020. Projektet er blevet kaldt Beertress, eftersom systemet i denne omgang kun vil servere øl.

Beertress indeholder en PSoC og sensorer, som skal gøre det muligt at navigere rundt i en restauration. Sensorerne der bliver benyttet, er bl.a. en farvesensor brugt til at navigere via en bane, en afstandssensor er brugt til at undgå forhindringer, en motor som får robotten til at køre, samt en vægt til at informere personale om genopfyldning af Beertress. Beertress kan modtage bestillinger fra kunderne via en hjemmeside, som også vil blive implementeret via en RPi og WebSocket i løbet af projektet.

Visionen med en robot, som kan navigere rundt i en restauration, bliver opfyldt til en vis grænse. Grundet udfordringer med tidspres har det ikke været muligt at opnå alle de opstillede mål, som opstod i starten af projektet, men formår at navigere rundt på en bane, ud fra en bestilling fra en kunde.

Indhold

1 Abstract	1
2 Resumé	2
3 Forord	7
3.1 Arbejdsfordeling	8
3.2 Ordforklaring	9
4 Indledning	9
4.1 Projektbeskrivelse	9
5 Projektafgrænsning	10
5.1 MoSCoW	10
6 Kravsspecifikation	12
6.1 Aktørbeskrivelse	12
6.2 Funktionelle krav	13
6.3 Ikke-funktionelle krav	14
6.3.1 Usability	14
6.3.2 Reliability	14
6.3.3 Performance	15
6.3.4 Supportability	15
6.3.5 Andre	15
6.3.6 Physical requirements	15
7 Metode og procesbeskrivelse	16
7.1 Metode	16
7.2 Proces	16
7.2.1 Udviklingsværktøjer	17
8 Analyse	17
9 Systemarkitektur	19
9.1 Hardware Arkitektur	19

9.2	Systemsekvensdiagrammer	21
9.2.1	Sekvensdiagram for UC1	21
9.2.2	Sekvensdiagram for UC3	22
9.2.3	Sekvensdiagram for UC4	23
9.3	Software Arkitektur	25
9.3.1	Domænemodel	25
9.3.2	Software allokering	25
9.3.3	Applikationsmodel for WaiterApp_SW	26
9.3.4	Applikationsmodel for MotorController_SW	28
9.3.5	Applikationsmodel for InterfaceController_SW	30
9.4	Beertress Protokol	31
10	Design, Implementering og Modultest	32
10.1	InterfaceController_SW	33
10.1.1	Software design	33
10.1.2	Software implementering	34
10.1.3	Kundegrænseflade Design	35
10.1.4	Kundegrænseflade Implementering	37
10.1.5	Modultest af RPI	37
10.2	WaiterApp_SW	38
10.2.1	Software Design	38
10.2.2	Software Implementering	38
10.3	MotorController_SW	39
10.3.1	Software design	39
10.3.2	MotorController_SW Implementering	41
10.4	i2cKommunikation	42
10.4.1	Software implementering	42
10.4.2	Modultest af I2C	43
10.5	Motor og Motorstyring	44
10.5.1	Hardware design	44
10.5.2	Hardware implementering	44
10.5.3	Software implementering	45
10.5.4	Modultest af Motor og Motorstyring	47

10.6 Farvesensor	48
10.6.1 Hardware design	48
10.6.2 Hardware implementering	48
10.6.3 Software implementering	50
10.6.4 Modultest af Farvesensor	51
10.7 Afstandssensor	52
10.7.1 Hardware design	52
10.7.2 Software implementering	52
10.7.3 Modultest	55
10.8 Vægtsensor	56
10.8.1 Hardware design	56
10.8.2 Hardware implementering	57
10.8.3 Software implementering	57
10.8.4 Modultest af Vægt	58
11 Test	58
11.1 Fremgangsmåde	58
11.2 Integrationstest	59
11.3 Systemtest	60
12 Resultater	61
12.1 Acceptttest af Use Case 1: Initialiser Beertress	61
12.2 Acceptttest af Use Case 2: Udskiftning af fustage	61
12.3 Acceptttest af Use Case 3: Bestilling	61
12.4 Acceptttest af Use Case 4: Betjening	61
12.5 Acceptttest af Use Case 5: Slukning af Beertress	61
12.6 Acceptttest af funktionelle krav	62
12.7 Acceptttest af ikke-funktionelle krav	62
13 Diskussion	62
14 Konklusion	63
15 Fremtidigt arbejde	64

3 Forord

Gruppe 8 består af 8 gruppemedlemmer, som alle læser Informations- og kommunikationsteknologi. Medlemmerne Peter Wann, Simon Phi Dang, Henry Pham, Lucas Friis-Hansen, Shyntavi Prithviraj, August Hjerrild Andersen, Alexander Flarup Wodstrup og Jim Sørensen. Gruppemedlemmerne læser hhv. på tredje og fjerde semester.

Projektet er blevet vejledt af Peter Høgh Mikkelsen, som er lektor på Ingenørhøjskolen, Aarhus Universitet. Projektet har forløbet fra d. 03/09-2020 til og med afleveringsfristen d. 18/12-2020. Projektet dokumenteres i rapporten samt i tilhørende bilag, hvoraf filerne indeholder detaljeret beskrivelser.

3.1 Arbejdsfordeling

Ansvarsområde	Ansvarshavende
RPi software	Alexander Flarup Wodstrup
I2C og grænsefladeprotokol	August Hjerrild Andersen
Personale- og kundegrænseflade	Jim Sørensen
Motor og PSoC main	Peter Wann
Line tracker sensor	Lucas Friis-Hansen og Shynthavi Prithviraj
Afstandssensor	Henry Pham
Vægtsensor	Simon Phi Dang

Tabel 1: Arbejdsfordeling

3.2 Ordforklaring

Ord	Betydning
RPi	Raspberry Pi Zero W.
PSoC	Programmable system-on-chip.
SW	Software
HW	Hardware
UML	Unified Modeling Language
SysML	Systems Modeling Language
I2C	Inter-Integrated Circuit
BDD	Block Definition Diagram
IBD	Internal Block Diagram
uc	Use case
sd	Sekvensdiagram
cd	Klassediagram
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
GPIO	General-Purpose Input/Output
AD	Analog Discovery
PSU	Power Supply Unit
TCP	Transmission Control Protocol

Tabel 2: Ordforklaring

4 Indledning

4.1 Projektbeskrivelse

I 2020 har COVID-19 præget verden og den måde vi omgås på. Dette ses blandt andet i restaurationsbranchen, hvor det er blevet mere besværligt at servere for kunderne. Risikoen for smitten forstørres, da afstandskravet på 1 meter ikke bliver imødekommet, når personalet skal servere mad og drikkevarer.

Dette problem har motiveret os til, at udvikle en servicerings-robot, som vil kunne servicere kunder

i en bar eller café med øl, så den fysiske kontakt med serveringspersonalet undgås.

Robotten skal kunne erstatte en tjener/bartender på eksempelvis en bar, natklub eller på en café, så en kunde vil kunne få skænket øl op, uden at skulle bevæge sig væk fra sit bord og dermed komme i fysisk kontakt med en bartender eller tjener.

Robotten skal fungere som et transportmiddel med en mindre øl fustage med en øl-skænker placeret oven på sig. Produktet vil blive kaldt Beertress og skal kunne køre fra sin standplads til barens borde.

Beertress skal finde rundt ved, at bordene enten skal have en fast placering i rummet, som Beertress kender i forvejen, eller også skal der dannes en form for sti fra standplads til de forskellige borde, i form af f.eks. farvet tape. Kunden kan enten anvende en knap eller bruge et website til kald af Beertress, hvorefter Beertress bevæger sig hen til det ønskede bord, udfører en skænkning af øl og derefter bevæge sig tilbage igen, når kunden har fortalt systemet, at aktionen er færdig.

Beertress skal ved hjælp af sensorer bremse, hvis der kommer nogle forhindringer på vejen til det bord, som har ønsket betjening, og skal generelt kunne befærde sig uden at være til gene for andre kunder.

5 Projektafgrænsning

I dette projekt vil projektgruppen prioritere, at få opbygget en funktionel robot, som kan køre rundt med en fustage. Beertress nedskaleres fra, at kunne køre med en stor fustage (>15L), til at kunne køre med en fustage på 5 L. Den automatiske opladning af Beertress afgrænses til manuel opladning. Såfremt der findes ekstra tid til udvidelser senere i projektforløbet, vil projektgruppen prioritere at få udvidelser som f.eks. udskænkning af øl via fustage, opskalering af robot samt brug af højttaler inkorporeret.

5.1 MoSCoW

Systemet vil bestå af flere funktioner, og ved hjælp af MoSCoW kan disse funktioner blive opdelt i ”must have”, ”should have”, ”could have” og ”won’t have” afhængig af, hvordan de er prioriteret i forbindelse med det endelige produkt. Igennem dette forløb vil der blive fokuseret på at implementere punkterne ”must have” og ”should have”, mens punkter som ”could have” vil blive udforsket, hvis der er tid tilovers. Punkterne ”won’t have” vil blive tænkt som fremtidig udvikling og vil derfor ikke blive implementeret i dette projekt.

Must have:

- Beertress **skal** kunne køre frem og tilbage
- Beertress **skal** have en sensor til at registrere forhindringer
- Beertress **skal** kunne bremse hvis nødvendigt
- Beertress **skal** kunne tilkaldes fra bordene ved hjælp af en knap eller website
- Beertress **skal** kunne dreje op til 360 grader

Should have:

- Beertress **bør** kunne ændre hastighed
- Beertress **bør** kunne skænke minimum en øl
- Beertress **bør** kunne tage imod mobilbetaling
- Beertress **bør** have et lager af plastikkrus

Could have:

- Beertress **kan** lyse op ved hjælp af LED's
- Beertress **kan** afspille lyd ved hjælp af højtalere
- Beertress **kan** køre tilbage og tilslutte lader efter betjening

Wont:

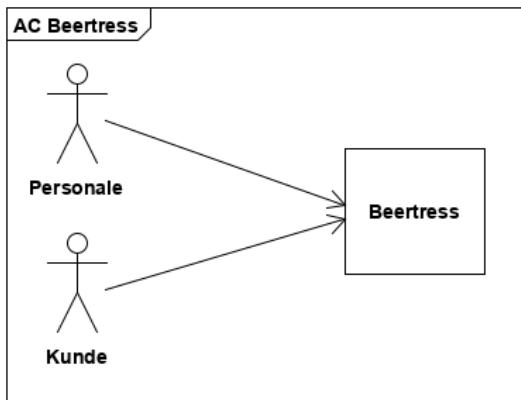
- Beertress **vil ikke** kunne skænke mere end 1 øl på samme tid
- Beertress **vil ikke** kunne skænke andet end øl

6 Kravsspecifikation

Ud fra projektbeskrivelsen og projektafgrænsningen har det været muligt at lave funktionelle krav til 'Beertress'. I dette afsnit beskrives systemet, så der er en præcis forståelse for hvad den skal kunne.

6.1 Aktørbeskrivelse

I dette afsnit beskrives systemet ved hjælp af et aktør-kontekst diagram, som vises på figur 1. Diagrammet viser, at Personale og Kunde er primær aktør. De overordnede beskrivelser af aktørerne findes i tabel 3 og tabel 4. Der beskrives det, at personalet kan starte og slukke Beertress efter behov via personalegrænsefladen, mens kunden kan vælge antal øl og hvilket bord, Beertress skal servere via kundegrænsefladen.



Figur 1: Aktør-Kontekst diagram for Beertress

Aktør	Personale
Type	Primær
Beskrivelse	Sørger for initiering samt nedlukning af Beertress via personalegrænseflade. Derudover sørger personalet for opfyldning af øl samt opladning af PSU.

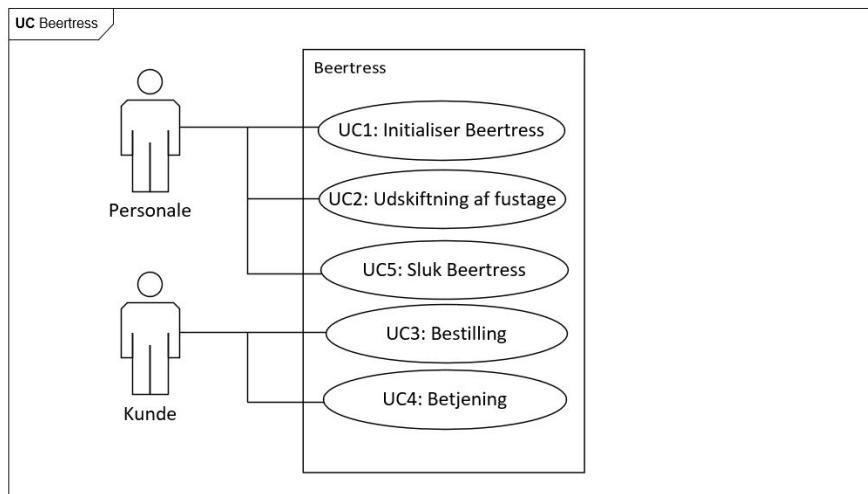
Tabel 3: Beskrivelse af aktør - Personale

Aktør	Kunde
Type	Primær
Beskrivelse	Kunden tilgår kundegrænsefladen. Her indtastes bordnummer samt antal øl, der ønskes serveret af Beertress.

Tabel 4: Beskrivelse af aktør - Kunde

6.2 Funktionelle krav

På figur 2 vises Use Case diagrammet for Beertress, som består af fem Use Case's. En brief beskrivelse af hver Use Case kan ses i tabel 5. De fully-dressed Use Cases kan findes i projektdokumentation under Kravspecifikationer.



Figur 2: UC diagram for Beertress

Use Case	Beskrivelse
UC1	Denne Use Case håndterer initialisering af Beertress via Personalegrænsefladen af Personale.
UC2	Denne Use Case håndterer udskiftning af fustage. Fortæller via Personalegrænsefladen, at Personale skal genopfylde fustage.
UC3	Denne Use Case håndterer bestilling af ordre fra Kunde via hjemmeside og sender denne ordre til Beertress.
UC4	Denne Use Case håndterer navigering af Beertress ned til bord og servering af øl. Sender information så log bliver opdateret.
UC5	Denne Use Case håndterer slukning af Beertress via Personalegrænseflade af Personale.

Tabel 5: Use Cases, brief form

6.3 Ikke-funktionelle krav

Nedenstående ikke-funktionelle krav er stillet op efter (F)URPS.

6.3.1 Usability

1. **U1.** Det skal maximalt tage 60 sekunder at skifte fustage på Beertress.
2. **U2.** 9 ud af 10 kunder skal kunne bestille øl, via hjemmesiden, uden brug for hjælp.
3. **U3.** Det skal maximalt tage 15 min at lære en ny kollega op i håndtering af Beertress.
4. **U4.** Personalet skal have mulighed for at kunne se 10 ordrer tilbage i loggen.

6.3.2 Reliability

1. **R1.** Beertress skal kunne holde strøm i 1 time ved brug, før opladning er nødvendigt.
2. **R2.** Beertress skal have en MTBF på 100 øl stænkninger.
3. **R3.** Beertress skal have en MTTR på 1.5 time.
4. **R4.** Beertress skal skænke 0,5 liter ($\pm 5\%$) når der trykkes på serveringsknappen.

6.3.3 Performance

1. **P1.** Beertress skal kunne opnå en hastighed på mindst 1 km/t.
2. **P2.** Beertress' kommunikations-rækkevidde skal være mindst 30 meter.
3. **P3.** Beertress bør kunne skænke en øl på minimum 10 sekunder.
4. **P4.** Beertress skal kunne tilsluttes en stikkontakt med 230 V.
5. **P5.** Beertress skal kunne veje fustagens vægt med 100% nøjagtighed $\pm 5\%$ i kg.
6. **P6.** Beertress skal kunne dreje 360 grader på stedet

6.3.4 Supportability

1. **S1.** Beertress skal kun være kompatibel med én 5 liter fustage.
2. **S2.** Beertress' dele skal maksimalt tage en time at udskifte hvis beskadiget.

6.3.5 Andre

1. **A1.** Personalegrænseflade skal være et program til linux eller windows.
2. **A2.** Kundegrænseflade skal være en hjemmeside.
3. **A3.** Beertress skal indeholde en Raspberry Pi Zero W.
4. **A4.** Beertress skal indholde en CY8CKIT-059 PSoC 5LP.
5. **A5.** Beertress skal indholde en sensor eller aktuator.

6.3.6 Physical requirements

1. **PR1.** Beertress må maksimalt veje 15 kg inkl. fyldt fustage.
2. **PR2.** Ølglas skal være mindst 75 cl.

7 Metode og procesbeskrivelse

7.1 Metode

Der er i system og hardware arkitekturen benyttet SysML for at give et overblik over komponenterne og deres kommunikation mellem hinanden. IBD, BDD og sekvensdiagrammer er alle lavet ud fra SysML.

Til softwarearkitekturen er der benyttet UML og SysML.

I analysedelen er der benyttet Risk Assessment, som har gjort det muligt for gruppen, at se på forskellige muligheder og løsninger for den optimale udgave af projektet. Dette er blevet udfærdiget i en tabel, som er vist i analyseafsnittet.

7.2 Proces

Projektets udviklingsproces har taget udgangspunkt i ASE-modellen[1]. I dette projekt er ASE-modellen blevet brugt til at skabe et overblik over projektforløbet. Forløbet blev indledt med projektformulering, specifikation og arkitektur, så der var et fundament at arbejde videre på, når designfasen, og dermed den iterative fase, blev indledt. På den måde var der en klar idé om hvad systemet skulle kunne, og der var en fælles vision for, hvor projektet skulle hen. Den iterative fase har gjort, at gruppen ikke var bange for at teste de forskellige komponenter, og har derfor fejlet i flere omgange, hvorefter man har kunnet lære af fejlene.

Som noget nyt for gruppen, er der forsøgt anvendt Scrum i udviklingsprocessen. Ved brug af Scrum gøres arbejdsgangene mere strømlinede da man har forskellige mål for hvert sprint, hvilket gør projektet mere overskueligt.

Sprint-backlog og product backlog blev sat op i et elektronisk task-board, således gruppen kunne få et overblik over de ting, der skal laves, er igangværende og er blevet færdiggjort i forhold til det sprint der blev sat op.

For at holde styr på kode-filerne blev GitHub anvendt. På denne måde kunne alle have adgang til al kode, alle diagrammer og alt dokumentation generelt.

Igennem projektforløbet blev der afholdt et review, hvor der blev kommenteret på reviewgruppens krav-, accepttestspecifikation og systemarkitektur. På samme måde fik gruppen konstruktiv kritik tilbage, som hjalp med at opdage fejl og mangler i denne fase.

7.2.1 Udviklingsværktøjer

Nedenstående tabel viser de udviklingsværktøjer, der er blevet brugt til udformning af projektet og en beskrivelse af hver.

Udviklingsværktøj	Beskrivelse
Trello	Elektronisk taskboard. Anvendt til sprint.
PSoC Creator	IDE til PSoC. Anvendt til at skrive kode og programmere til PSoC.
GitHub	Online Repository. Anvendt af gruppemedlemmerne til deling af kode, dokumentation osv.
VMPlayer	Virtual Machine. Anvendt til at køre Linux virtuelt og overføre til Raspberry Pi.
Visual Studio Code	Code Editor. Anvendt til at skrive kode til Raspberry Pi.
WaveForms	Analyseringsværktøj til AD. Anvendt til at undersøge I2C-kommunikation.
Overleaf	Online LaTeX Editor. Anvendt til udarbejdelse af projektdokumentation.
Discord	Kommunikationsværktøj. Anvendt til online kommunikation imellem gruppemedlemmer.
Zoom	Kommunikationsværktøj. Anvendt til vejledermøde når det ikke kunne gøres fysisk.
Realterm	Serial/TCP Terminal. Anvendt til at udskrive resultater til konsolvindue.

Tabel 6: Udviklingsværktøjer brugt i projektet

8 Analyse

Der blev, fra starten af projektforløbet, udarbejdet en Risk Assessment, hvor der blev undersøgt forskellige muligheder og problemstillinger til hver hardware- og softwaredel af systemet. Denne kan ses i bilaget under Risk Assessment, hvor der også kan ses en mere dybdegående redegørelse for valg af komponent. Ud fra denne er nedenstående tabel blevet opstillet, hvor man kan se, hvilke analytiske overvejelser der har været, til hver block i systemarkitekturen. Nogle af komponenterne, såsom f.eks. køretøj, begrænser mulighederne, så disse er ikke blevet analyseret dybere.

Komponent	Minimums krav	Hvilke muligheder findes der?	Argumenter for /imod?	Kandidat	Forsøg
PSU	Forsyne systemet med 9.6V, 5V og 3.3V	Utal af batterier ved Embedded Sto- ck	Skal lave strømregulator, da kun 9.6V batteri er dem man skal bruge	9.6V og batteri	Lave strømregulator og måle output med voltmeter
Kommuni- kation	Skal kunne kommunikere mellem RPi og PSoC	I2C, SPI, UART	UART er prøvet før, SPI mange pins men hurtig, I2C har 2 pins men knap så hurtig	I2C	Læse fra slave via I2C
Line-tracker sensor	Følge en linje og stoppe tre steder (Bord 1, Bord 2 og slutposition)	IR-sensor, TCS3200 og TCS230	TCS-sensorerne er mest optimalt, da de forskellige farver øger funktionaliteten af BeerTress	TCS230	Detekttere farverne rød, blå og grøn
Trådløs kommunikation	Skal kunne kommunikere mellem Beertress og telefon/PC trådløst	WebSocket og AdHoc	WebSocket: full-duplex, virker til hjemmeside, event-driven, kræver internet/hotspot forbindelse. AdHoc: direkte forbindelser, begrænset rækkevidde, kræver flere RPi'er	WebSocket	Forbind en RPi og hjemmeside via websocket og få den til at afhæse et input fra hjemmesiden
Afstandssensor	Skal kunne detektere et objekt foran sensoren	HC-SR04 og 2Y0A21YK	HC-SR04 påvirkes ikke af lys, og detektionsafstanden er 5 gange større end for 2Y0A21YK. 2Y0A21YK benytter JST connector, som ikke er tilgængelig via elektroniskærksted eller Embedded Stock	HC-SR04	Dekktere et objekt 50 cm væk
Vægtsensor	Vejning af øl	1 kg load cell, 10 kg load cell og køkkenvægt	Køkkenvægt ikke fleksibel nok da det player kun vægt, 10 kg load cell ikke nødvendig i forhold til den ekstra opsætning, 1 kg nemeste løsing med lidt simplicørsætning	1 kg load cell	Vejning af øldåse

Grundet pladsmangel i tabellen, er risk assessment for **Personalegrænsefladen** beskrevet herunder:

En af fordelene ved at benytte Microsoft Asp.net framework, er at man får stillet nogle værktøjer til rådighed, som gør implementeringen meget nemmere. Når man arbejder med websockets, kan der hentes ekstra pakker ind i projektet fra Microsoft NuGet, som har indbyggede funktioner, der blot skal implementeres. I dette tilfælde kunne en Nuget-pakke hentes ind, som hedder System.Net.WebSockets(4.3.0) og Websocket.Client(4.3.21). Med de to pakker gjorde det arbejdet med at lave forbindelsen nemmere.

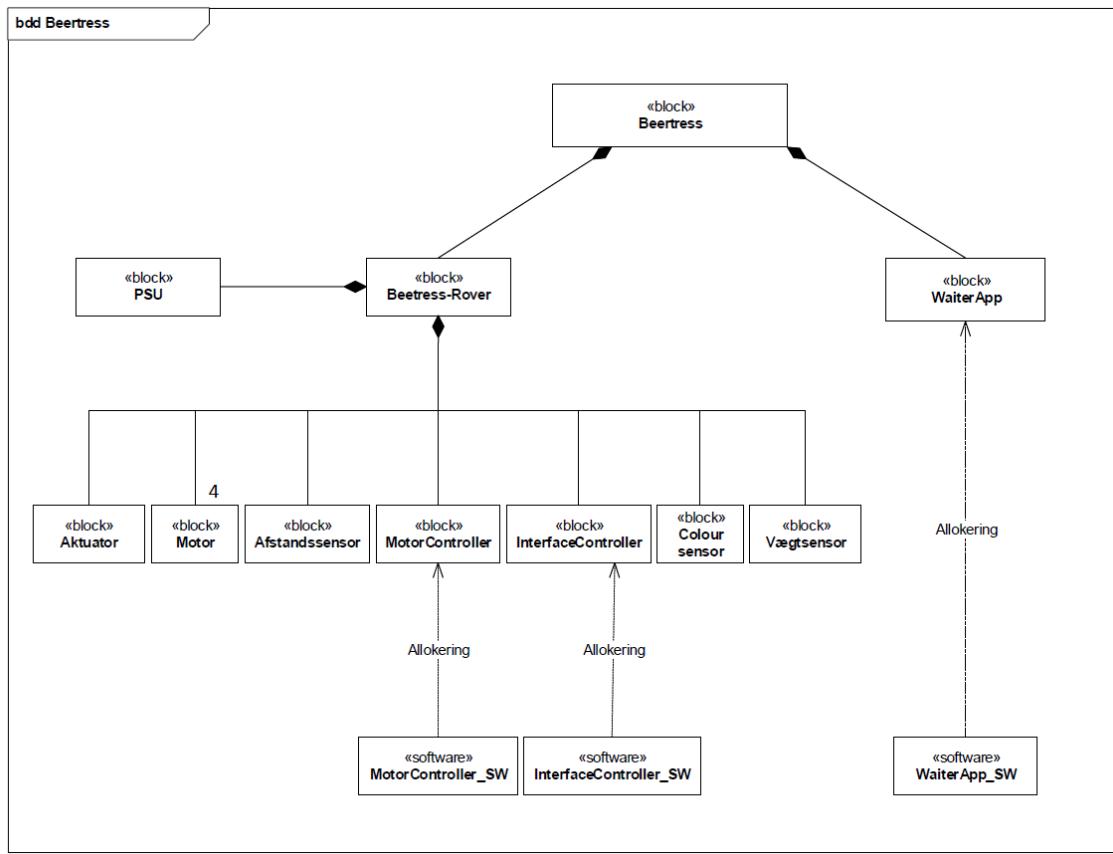
Derfor må det siges, at der ikke har været valgt at benytte et framework til personalet ud fra, at det er den bedste løsning, fordi det er det ikke til dette projekt. Valget er udelukkende fortaget, fordi man på det tidspunkt ikke havde kendskab til, de muligheder, som findes derude. At bruge en Windows App, som kører på pc'en er en fin nok løsning, men det ville være smartere at have noget, som kører online, så man kunne tilgå dette på en mere fleksibel måde.

9 Systemarkitektur

I dette afsnit vil arkitekturen for systemet blive beskrevet ved hjælp af SysML diagrammer. Formålet med systemarkitektur afsnittet er, at skabe kendskab til hele systemet og hvorledes det hænger sammen. Der laves derfor beskrivelser af hele systemet og de enkelte moduler, systemet består af. Herunder vil deres funktion og kommunikationen mellem modulerne også blive beskrevet.

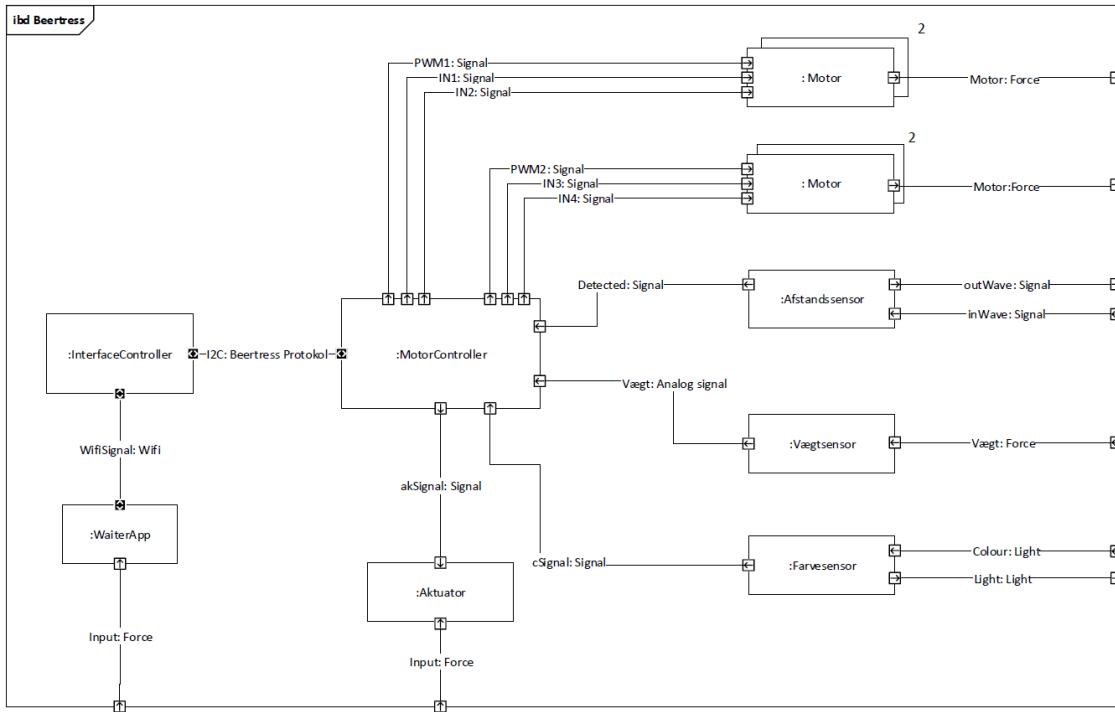
9.1 Hardware Arkitektur

I følgende afsnit vises den overordnede hardware arkitektur, og hvorledes de forskellige hardware komponenter kommunikerer med hinanden. Med den viden, der er blevet indsamlet for de forskellige moduler, er det blevet muligt at udvikle et BDD for Beertress. Det udviklede BDD, som ses på figur 3, viser de blokke systemet består af. På BDD'et kan man bl.a. også se, at der er allokeret software til 3 af blokkene, nemlig MotorController, InterfaceController og WaiterApp. Et mere beskrivende BDD, med porte, parts og en mere beskrivende forklaring kan findes under projektdokumentation systemarkitektur.



Figur 3: BDD for Beertress

Det overordnede IBD af Beertress ses på figur 4. IBD’et er med til at illustrere de enkelte blokkes interne kommunikation og hvorledes de hænger sammen. PSU er blevet udeladt i IBD’et, da det ikke vil give mening at vise, at den forbindes til begge controllers, motor og alle sensorer. Der er blevet udviklet et separat IBD for PSU’en, hvor dens forbindelse til modulerne kan ses. Dette kan findes i projektdokumentationen under hardware arkitektur.



Figur 4: Overordnet IBD for Beertress

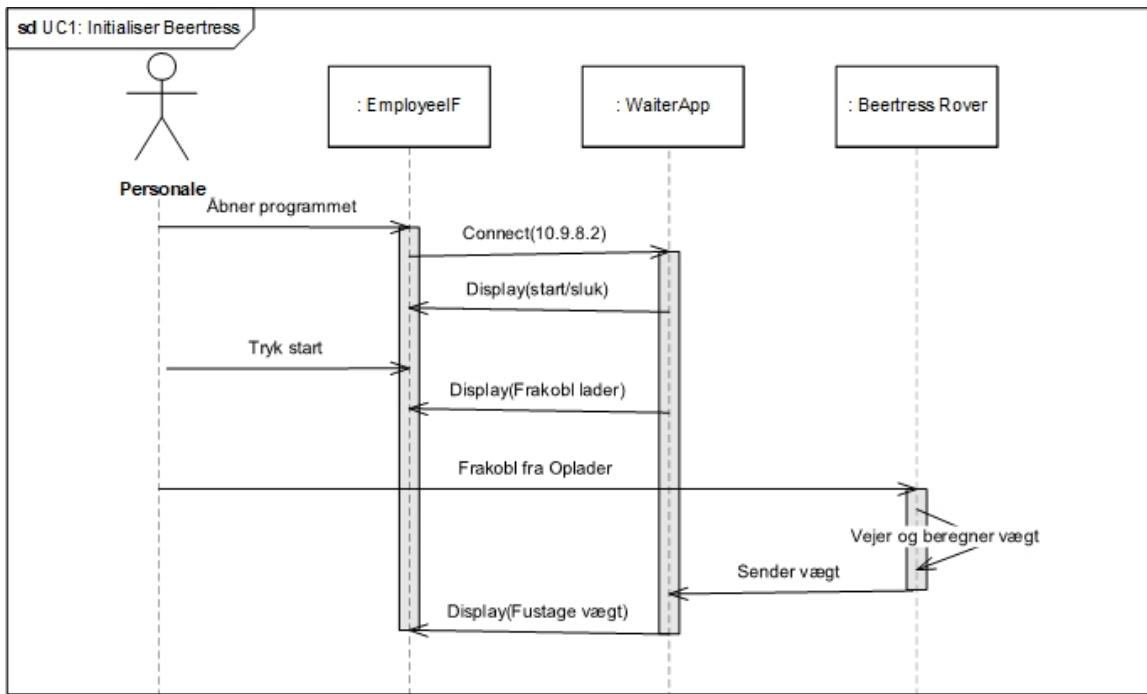
Derudover kan tabeller af blokbeskrivelse og signalbeskrivelse også findes i projektdokumentationen under Systemarkitektur.

9.2 Systemsekvensdiagrammer

For at give et overblik over systemet, vil sekvensdiagrammerne for UC1, UC3 og UC4 gennemgås, da liget netop disse tre Use Cases, giver det store overordnede overblik over kommunikationen mellem komponenterne i systemet.

9.2.1 Sekvensdiagram for UC1

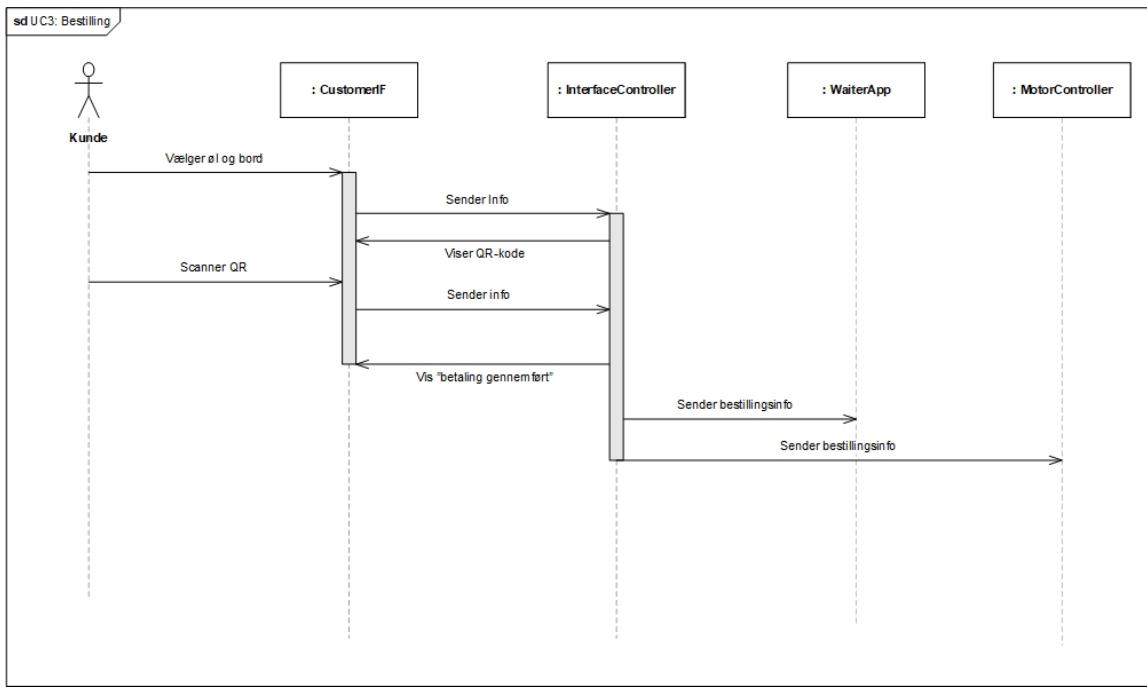
Figur 5 viser sekvensdiagrammet for UC1, som handler om opstart af Beertress. Dette sker fra WaiterApp som er en PC applikation.



Figur 5: Sekvensdiagram for UC1

9.2.2 Sekvensdiagram for UC3

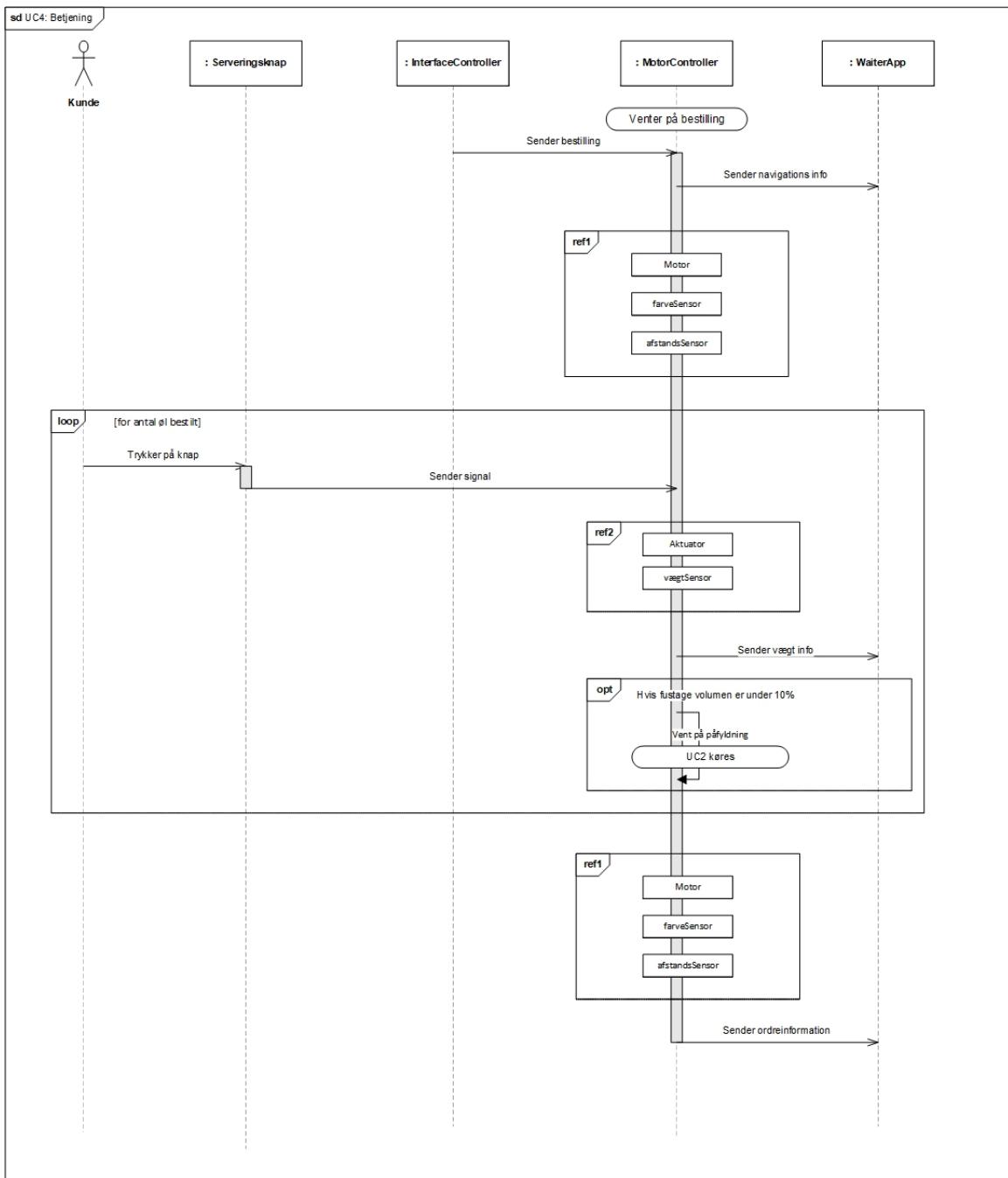
Figur 6 viser processens udfoldelse for UC3, som omfatter bestilling af øl. Kunden vælger antal øl samt hvilket bordnummer de sidder ved. Når CustomerIF har registreret disse oplysninger, sendes de til interface controlleren. Der sendes derefter en QR-kode ud på CustomerIF, hvorefter kunden scanner denne. Nå kunden har scannet QR-koden, sendes der oplysninger tilbage til interface controlleren om at betaling er gennemført. Der sendes herefter en besked ud på websiden, at betalingen er gennemført. Use Casen afsluttes ved at interface controlleren sender bestillingsinfo til hhv. WaiterAppen og MotorControlleren. Under WaiterApp findes en log som lagrer disse data.



Figur 6: Sekvensdiagram for UC3

9.2.3 Sekvensdiagram for UC4

Figur 7 viser sekvensdiagrammet for UC4. MotorController'en modtager først en bestilling fra InterfaceController'en, som har modtaget en bestilling fra kunden, hvilket kunne ses i sekvensdiagrammet for UC3. Af overbliksmæssige hensyn er der lavet yderligere sekvensdiagrammer, som viser forbindelserne mellem MotorController'en og de forskellige sensorer, som systemet består af. Se i projekt-dokumentationen under systemarkitektur for at læse mere om disse. Til sidst sendes informationerne om ordren til WaiterApp, som gemmer disse i en log.



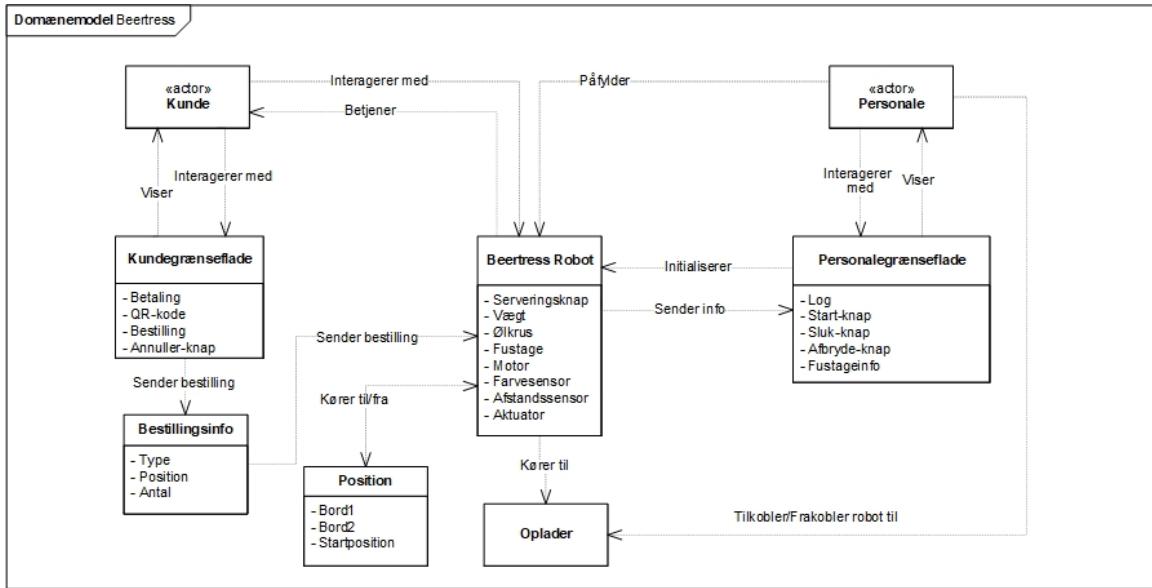
Figur 7: Sekvensdiagram for UC4

9.3 Software Arkitektur

I de følgende afsnit vil software arkitekturen blive beskrevet. Først indledes der med en domænemodel for at vise et overblik over systemet. Derefter vises applikationsmodeller for hver block i BDD, på figur 3, der skal have software allokerings.

9.3.1 Domænemodel

Figur 8 viser en domænemodel for systemet Beertress. Denne har til formål, at give et overblik over systemet. Denne er blevet udarbejdet på baggrund af indholdet i Use Cases, hvor man samler alle navne- og udsagnsord.



Figur 8: Domænemodel for Beertress

9.3.2 Software allokering

Som det kan ses i BDD'et på figur 3, skal der software allokeres til tre blokke: InterfaceController, MotorController og WaiterApp. En beskrivelse af hver allokering kan ses i tabel 8.

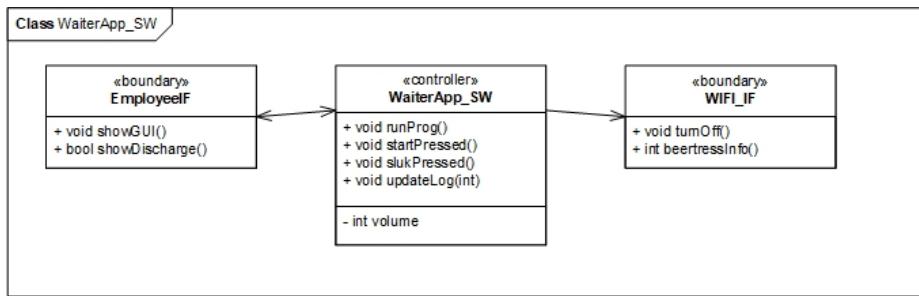
Allokering	Beskrivelse
WaiterApp_SW	Denne software skal være et program, der kan bruges på f.eks. personalets PC, så personalet kan starte og slukke Beertress Rover, samt se en log over bestillinger og volume på fustage som bliver opdateret af InterfaceController.
InterfaceController_SW	InterfaceController er i dette tilfælde en Raspberry Pi, som skal have software, så en hjemmeside kan tilgås af kunden via WiFi hotspot, hvor kunden kan bestille til sit ønskede bord. Dernæst skal den kunne kommunikere med MotorControlleren via I2C, så MotorControlleren kan navigere ned til bordet og servere bestillingen, samt opdatere loggen, som WaiterApp kan tilgå.
MotorController_SW	MotorController_SW er i dette tilfælde en PSoC, som skal have software så den kan håndtere motor, farvesensor, afstandssensor, vægtsensor og aktuator, så den kan navigere hen til det korrekte bord og servere bestillingen. Den skal kunne modtage ordrer fra InterfaceController og sende information tilbage når ordrer er færdig og om fustage eventuelt skal genopfyldes.

Tabel 8: Software allokeringer

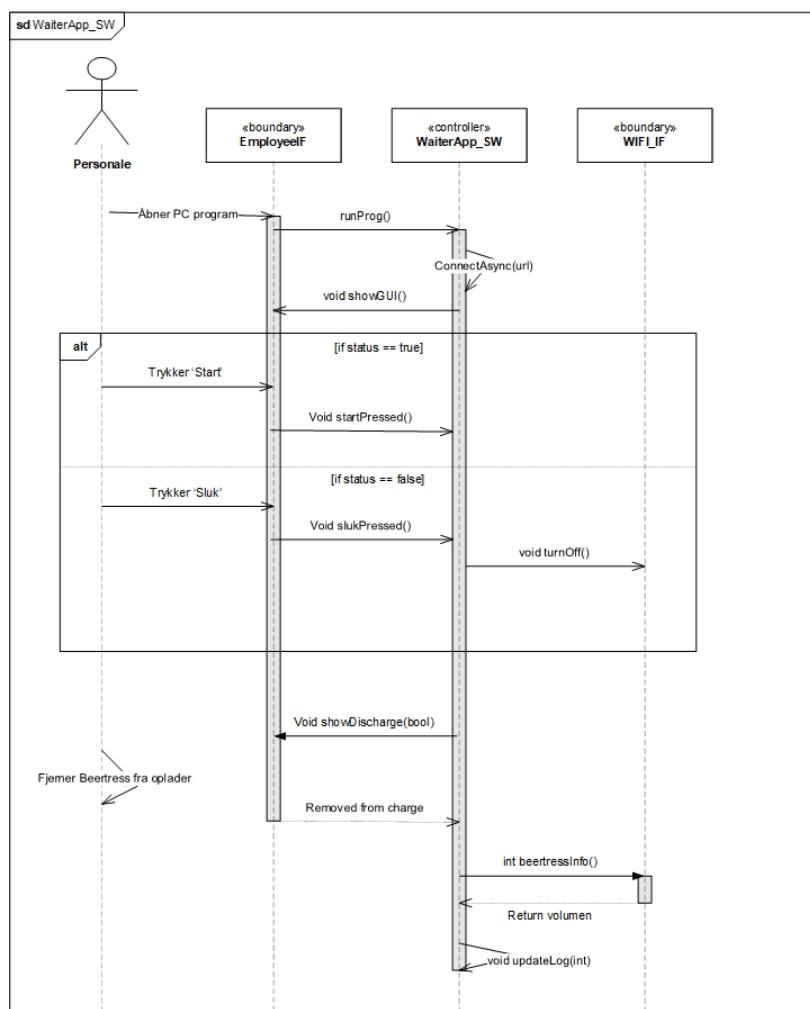
9.3.3 Applikationsmodel for WaiterApp_SW

I det følgende afsnit beskrives den overordnede software arkitektur for WaiterApp_SW ved brug af klassediagrammer og sekvensdiagrammer. Disse er blevet udarbejdet med udgangspunkt i Use Case 1 samt domænemodellen.

Klassediagrammet med metodekald kan ses på figur 9 og har en controllerklasse der aktiveres når personalet tænder programmet. Boundaryklassen EmployeeIF sørger for at vise det grafiske interface til personalet hvorpå de kan starte, slukke og se logbeskeder. Boundaryklassen WIFI_IF sørger for kommunikationen til InterfaceController via WiFi, så InterfaceController kan hente information om volumen samt opstarte og nedlukke MotorController (og dermed Beertress Rover). Sekvensdiagrammet med metodekald kan ses på figur 10.



Figur 9: Klassediagram for WaiterApp_SW med metodekald

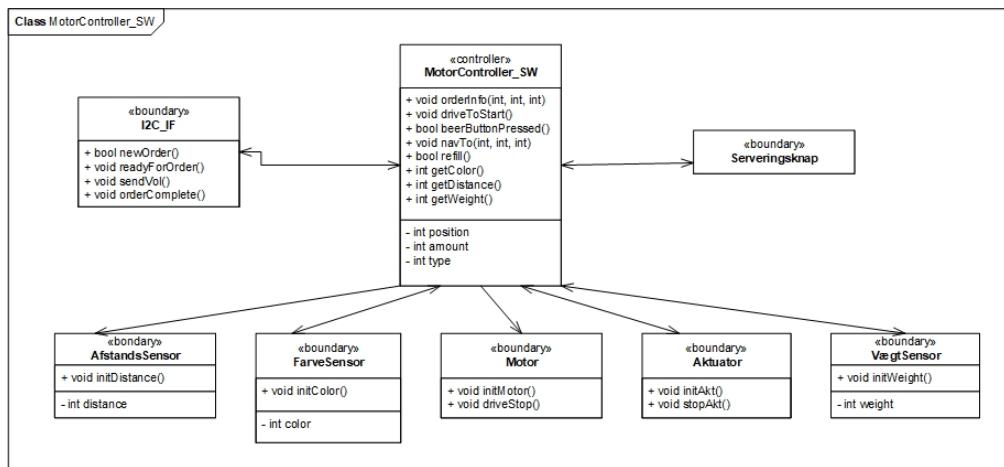


Figur 10: Sekvensdiagram for WaiterApp_SW

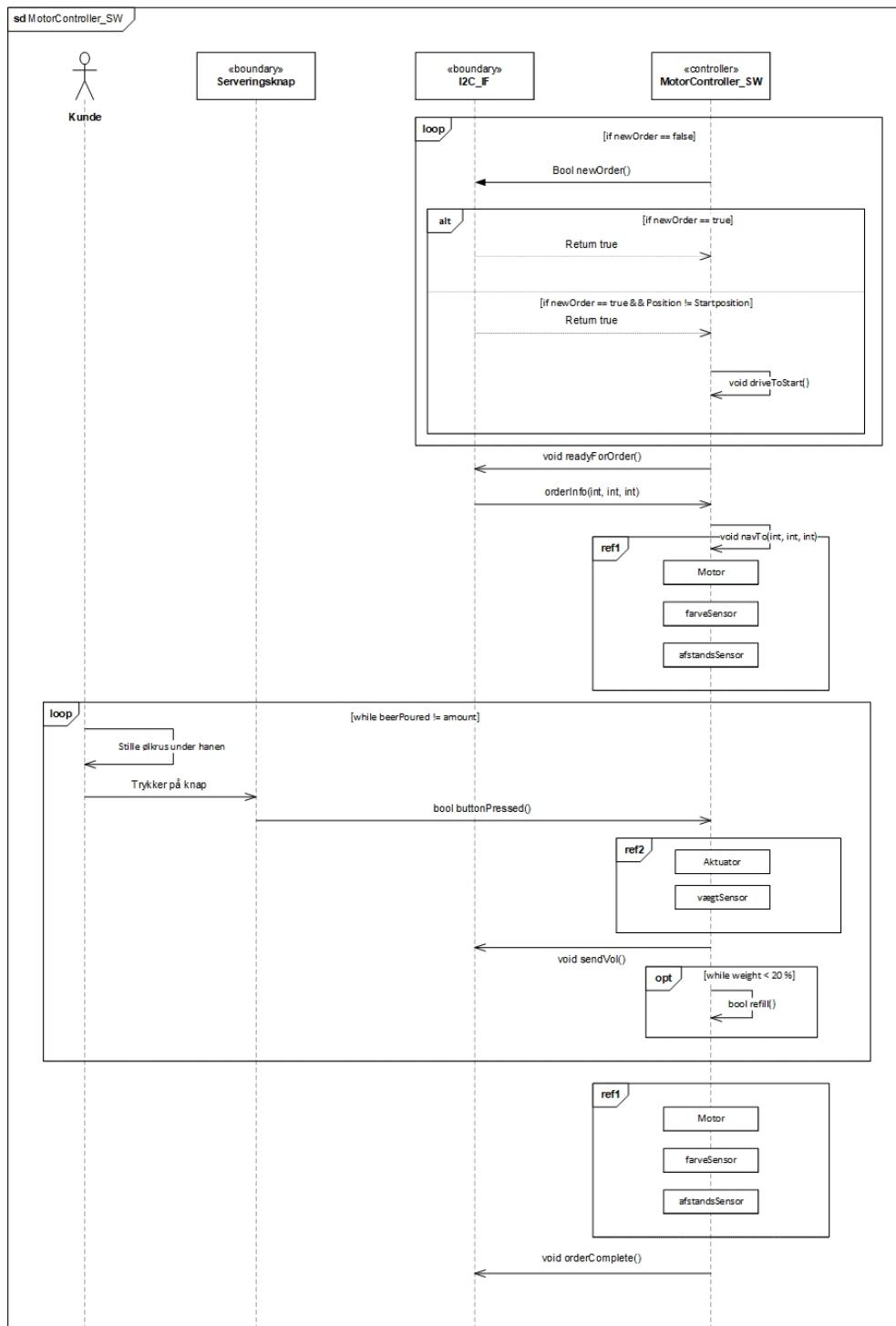
9.3.4 Applikationsmodel for MotorController_SW

I det følgende afsnit beskrives den overordnede software arkitektur for MotorController_SW ved brug af klassediagrammer og sekvensdiagrammer. Disse er blevet udarbejdet med udgangspunkt i Use Case 4 samt domænemodellen. Eftersom UC4 udelukkende kører på MotorControlleren, er denne valgt som controller.

Klassediagrammet kan ses på figur 11, og har en controllerklasse, der modtager bestillingsinformation fra InterfaceController via boundaryklassen I2C_IF, og sender information tilbage igen når ordren er afsluttet. Dernæst nавigerer MotorControlleren banen og serverer bestillingen til kunden ved hjælp af de andre boundaryklasser. Sekvensdiagrammet for MotorController_SW kan ses på figur 12. Ref1 og ref2 uddybes yderligere i projektdokumentationen under systemarkitektur.



Figur 11: Klassediagram for MotorController_SW med metodekald

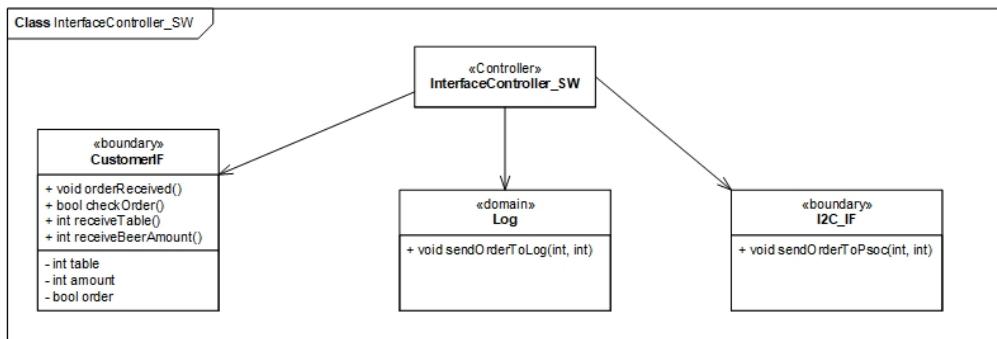


Figur 12: Sekvensdiagram for MotorController_SW

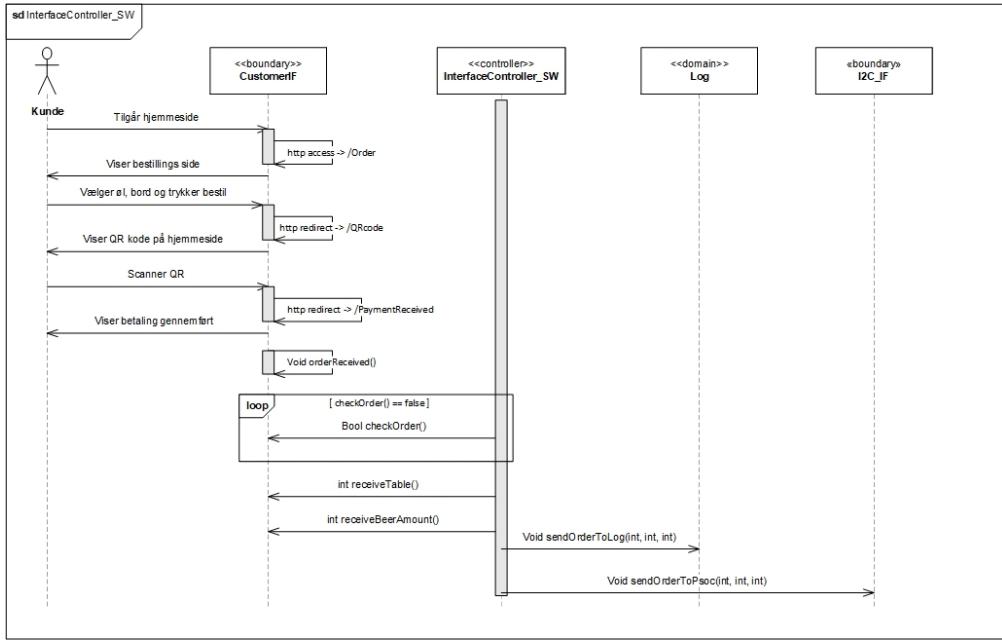
9.3.5 Applikationsmodel for InterfaceController_SW

I det følgende afsnit beskrives den overordnede software arkitektur for InterfaceController_SW ved brug af klassediagrammer og sekvensdiagrammer. Disse er blevet udarbejdet med udgangspunkt i Use Case 3 samt domænemodellen. Eftersom UC3 udelukkende kører på InterfaceController, er denne valgt som controller.

Klassediagrammet kan ses på figur 13 og har en controllerklasse, der modtager ordrer hos boundaryklassen CustomerIF og henter information om bestillingen. Den sørger for dette bliver registreret i domæneklassen Log og sender bestillingsinformationen videre til MotorController via boundaryklassen I2C_IF. Sekvensdiagrammet kan ses på 14.



Figur 13: Klassediagram for InterfaceController_SW med metodekald



Figur 14: Sekvensdiagram for InterfaceController_SW

For yderligere beskrivelser af applikationsmodellerne henvises til projektdokumentationen under systemarkitektur.

9.4 Beertress Protokol

Protokollen er sat op efter I2C kommunikation, hvor der først sendes en byte fra masteren med slaveadressen til den slave, der skal skrives til, og derefter en dataframe. Selve protokollen er følgende: Der sendes 3 bytes med data (udover den første byte som angiver slave adressen). Den første byte bestemmer hvilken type væske, der er ønsket. Det kunne være, at kunden har bestilt øl, cider, kaffe eller noget helt andet. I projektets nedskalerede tilfælde vil det altid være øl, men med denne protokol er det blevet skalérbart, så det vil være enkelt at udvide til andre væsketyper. Der vil kunne blive op til 255 forskellige væsketyper. Det andet byte angiver hvilket bord der har ønsket betjening. Der kan være op til 255 borde. Det tredje byte angiver antal af den ønskede væske, som bliver angivet i byte 1. Der kan igen være op til 255. En yderligere og mere dybdegående beskrivelse af protokollen kan findes i projektdokumentationen.

Tabel 9 viser 2 eksempler på protokollen. I det første eksempel bestilles der 5 øl til bord 2. I det

andet eksempel bestilles 3 cider til bord 1.

	Type	Bord	Antal
Dataframe, eks1:	0b00000001	0b00000010	0b00000101
Dataframe, eks2:	0b00000010	0b00000001	0b00000011

Tabel 9: Eksempler på dataoverførsel via protokol

Tabel 10 viser de 6 mulige dataoverførsler i dette projekt, da der er blevet nedskaleret til max 2 borde og max 3 øl.

	Type	Bord	Antal
Bord 1, 1 øl:	0b00000001	0b00000001	0b00000001
Bord 1, 2 øl:	0b00000001	0b00000001	0b00000010
Bord 1, 3 øl:	0b00000001	0b00000001	0b00000011
Bord 2, 1 øl:	0b00000001	0b00000010	0b00000001
Bord 2, 2 øl:	0b00000001	0b00000010	0b00000010
Bord 2, 3 øl:	0b00000001	0b00000010	0b00000011

Tabel 10: Mulige dataoverførsler i nedskaleret udgave

10 Design, Implementering og Modultest

I de næste afsnit vil design, implementering og modultest af systemet blive beskrevet.

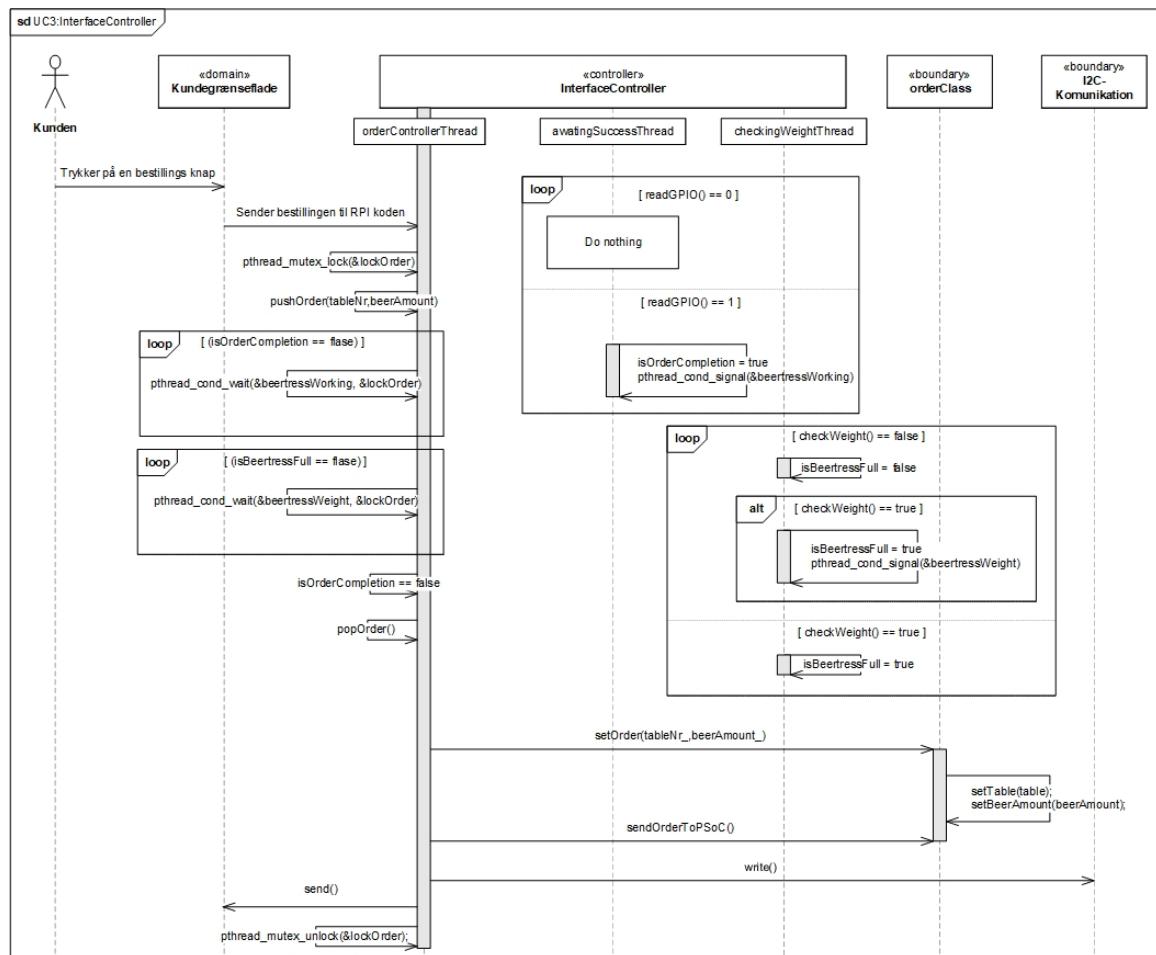
I forhold til arkitekturen, blev det klart for gruppen, at yderligere begrænsninger har været nødvendige, i forhold til realiseringen af en fungerende prototype. Det har ikke været realistisk, at implementere aktuatoren, og grundet begrænsede muligheder for køretøj, har det ikke været muligt at implementere fustage. Dette er i stedet nedskaleret til en dåseøl, som bliver påsat Beertress.

Udover de næste afsnit, er der også implementeret en spændingsregulator. Denne er ikke beskrevet i hovedrapporten, men er blevet forklaret dybdegående i projektdokumentationen under systemdesign og implementering.

10.1 InterfaceController_SW

I nedenstående afsnit vil der kort blive beskrevet, hvordan InterfaceControllerSW er blevet designet og implementeret. I dette afsnit vil InterfaceControllerSW refereret til som RPi. RPi'ens formål er at håndtere de ordrer, der bliver sendt fra kundegrænsefladen og sende dem videre til PSoC'en, når den er klar til at modtage nye information. For en mere detaljeret forklaring og uddybelse af design og implementering henvises der til projektdokumentationen under systemdesign og implementering.

10.1.1 Software design



Figur 15: Overbiks sekvensdiagram for InterfaceController_SW

Ovenstående ses et overbliks sekvensdiagram for hvordan RPi koden kommer til at virke. Her er tanken, at eftersom programmet skal håndtere flere ting på samme tid, er det nødvendigt at oprette det som et flertrådet program. Programmet skal kunne modtage data fra hjemmesiden og gemme de vigtige værdier i en ordre og sende den i en kø. Der vil blive oprettet en tråd til dette, som kan aflæses på Figur 15 under ”orderControllerThread”, den vil blive sat op til at benytte pthread_cond_wait. Der skal på samme tid hele tiden aflæses en fysik pin på RPi’en, som skal bestemme om PSoC’en er klar til at modtage en ny ordre. Til dette formål vil der blive oprettet en tråd: ”awatingSuccessThread”. Til sidst skal det også være muligt, at aflæse hvad PSoC’en sender til RPi’en. Hvis der bliver sendt 1 til RPi’en fra PSoC’en betyder det, at der ikke længere er en øl på vægten. Til dette formål vil der blive oprettet en tråd: ”checkingWeightThread”. Disse to threads vil benytte pthread_cond_signal til at kunne styre hvornår ”orderControllerThread” må arbejde.

10.1.2 Software implementering

Selve InterfaceController programmet er skrevet i C++ i linux. Dette program benytter en char driver, der er blevet udviklet i forbindelse med projeket, som er skrevet i C. InterfaceController programmet er bestående af 3 pthread tråde som kan ses i tabel 11:

orderControllerThread	Denne tråd sørger for at håndtere største delen af programmet: Aftyde beskeder fra hjemmeside, opbevaring af bestillinger, sende bestillinger til PSoC og sende en godkendelse besked til hjemmesiden.
awatingSuccessThread	Denne tråd sørger for at aflæse på GPIO pin om der sendes et 1 eller 0. Denne værdi bliver så brugt til at bestemme om der må blive sendt en ny ordre eller om der skal ventes.
checkingWeightThread	Denne tråd sørger for at aflæse på I2C om der modtages et 1 eller 0. Denne værdi bliver så brugt til at bestemme der må blive sendt en ny ordre eller om der skal ventes.

Tabel 11: Tråd beskrivelse

”awatingSuccessThread” og ”checkingWeightThread” har begge til ansvar, at bestemme hvornår ”orderControllerThread” må arbejde og hvornår den skal vente indtil at PSoC’en er klar til at modtage en ny ordre. De styrer ”orderControllerThread” igennem to pthread conditions: ”beertressWorking” og ”beertressWeight”.

”orderControllerThread” benytter en mutex til at låse den data, der bliver arbejdet på når den starter funktionen og låser op igen når den er færdig. For at sikre, at ”orderControllerThread” ikke sender data til PSoC’en når den ikke er klar, bliver der benyttet to pthread_cond_wait. Her bliver tråden sat til at vente indtil der bliver sendt et pthread_cond_signal fra ”awatingSuccessThread” og ”checkingWeightThread” med mindre ”orderControllerThread” krav, for at forsætte, allerede er opnået.

”awatingSuccessThread” benytter en funktion, der bruger en char driver på RPi’en til at åbne, aflæse og lukke en fysik GPIO pin. Denne driver er udarbejdet i forbindelse med undervisningen og modificeret til at passe til projektet. Eftersom alt på en raspberry pi arbejder med filer, skal man åbne en bestemt fil for at kunne benytte char driveren. Dette bliver gjort igennem ”open(/dev/orderCompleteGPIO”) -kaldet. Denne char driver sørger for, at oprette filerne til brug af GPIO 25, når driveren bliver kaldt, samt at ændre den valgte GPIO pin til input. Efter driveren er åbnet får man mulighed for, at benytte dens read-funktion, der vil returnere værdien af den valgte GPIO pin.

”checkingWeightThread” aflæser konstant på I2C forbindelsen og hvis der modtages et 1 vil det betyde, at der ikke længere står en øl på vægten og derfor skal der ventes med at sende en ny ordre indtil, der er fyldt op igen.

For at få programmet til at køre når RPi’en bliver tilsluttet strøm, er der blevet oprettet et script, som står for at opsætte og indsætte GPIO driveren, og derefter køre programmet. Dette script er blevet sat til at køre når RPi’en tænder.

10.1.3 Kundegrænseflade Design

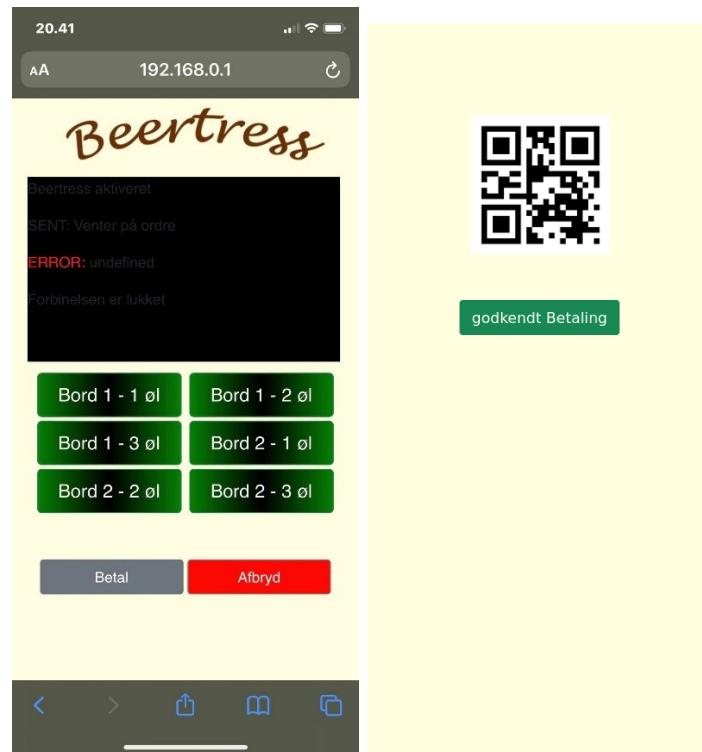
Når der skal designes en brugergrænseflade, er det vigtigt at starte med en skitse, så man kan danne sig et billede af, hvordan det færdige resultat vil komme til at se ud. På figur 16 ses en skitse, som blev konstrueret, inden selve implementationen gik i gang.

Som kunde skulle det være muligt at bestille service igennem en app eller en brugergrænseflade, monteret ved det bord man sad ved. Her skulle det være muligt, også at fortælle Beertress, hvor mange genstande, man ønskede. Til at starte med blev der valgt, at lave en dropdown menu, hvor man kunne vælge antal øl.



Figur 16: Skiste af brugergrænse flade

Dette udgangspunkt blev dog ændret til følgende design, som ses på figur 17.



Figur 17: Hjemmeside med QR kode

10.1.4 Kundegrænseflade Implementering

For at gøre kunde-brugergrænsefladen lettest og hurtigst tilgængelig for kunderne, er der lavet en simpel html hjemmeside, der fungerer som en App. Som kunde skal det nemlig være enkelt og hurtigt at kunne få adgang til systemet og få den ønskede service, uden unødig at interagere med personalet. Når først man har adgang til det netværk, hvor Appen befinner sig, er det nemt at gå ind og bestille. Alternativet ville være, at man lavede en rigtig app som kunderne kunne hente ned, men denne løsning er smartere og hurtigere, fordi de blot skal gå ind på en hjemmeside/IP-adresse, og få sekunder efter kan der fortages en bestilling. Selve stylingen af Appen er gjort ved hjælp af css. For at gøre brugerfladen mobil-venlig, er der brugt Bootstrap 4.0, som automatisk skalerer elementerne, når størrelsen ændres.

For at nedskallere programmet lidt blev det valgt at der skulle implementeres 6 knapper, der styrer om der bliver bestilt imellem 1-3 øl til enten bord 1 eller bord 2, afhængig af hvad kunden vælger. Den ideelle løsning havde været at lave en drop down menu eller et tekst felt, hvor kunden selv kan vælge det bord, de sidder ved, eller hvor mange øl de ønsker. For at opnå de ønskede krav til kundegrænsefladen er der benyttet websocket, HTML og CSS. Her er det sat op så, når der bliver trykket på en knap vil der blive kørt websocet funktionen doSend("Bord: " + tableNr + "Antal øl: " + beerAmount). Denne function sørger for at køre funktionen, der skriver den sendte bestilling på skærmen i loggen samt sender char beskeden videre til InterfaceController_SW-koden.

10.1.5 Modultest af RPI

For at kunne teste om RPi programmet virker, som forventet, er det blevet forbundet med hjemmesiden. Ved at have RPi'en tilsluttet en computer har der været mulighed for, at følge med i en terminal og printe de forskellige data'er ud, så man kan følge med i at programmet arbejder som forventet. Her er alle 6 knapper blevet testet og det kan aflæses i terminalen at RPi-koden håndterer beskederne korrekt og printer de forskellige bestillinger ud på terminalen afhængig af hvad der bliver valgt på hjemmesiden.

Efterfølgene er der blevet inkluderet en PSoC med i test-delen så man kunne afprøve om RPi-koden kunne håndtere og sende de korrekte bestillinger igennem I2C. Her er der brugt det I2C, som senere i rapporten er blevet beskrevet og testet for sig selv. Dog er der blevet tiltøjjet noget kode til PSoC'en så den vil blinke med dens indbyggede LED afhængig af hvilken bestilling, der er blevet sendt.

10.2 WaiterApp_SW

10.2.1 Software Design



Figur 18: Personalegrænseflade design

Personale-brugerfladen er lavet som en wpf .net framework Desctop App. Designet er lavet så det er simpelt og enkelt for de ansatte at bruge. Den består af en grøn knap, som starter beertress og en rød, som slukker. På højre side ses en sort skærm, hvor der bliver udskrevet, hvor i processen beertress befinner sig. Det hele er styled med Xaml, som er WPF's alternativ for css.

10.2.2 Software Implementering

Personalegrænsefladen er lavet i en asp.net 3.1 Windows-Applikation. En af fordelene ved at benytte Microsoft Asp.net framework til personalets brugergrænseflade på figur 18, er at man får stillet nogle værktøjer til rådighed, som gør implementeringen nemmere. Når man fx arbejder med websockets, kan der hentes ekstra pakker ind i projektet fra Microsoft NuGet, med indbyggede funktioner, der blot skal implementeres. I dette tilfælde kunne Nuget-pakker som System.Net.WebSockets(4.3.0) og Websocket.Client(4.3.21). hentes ind. Med de to pakker gjorde det

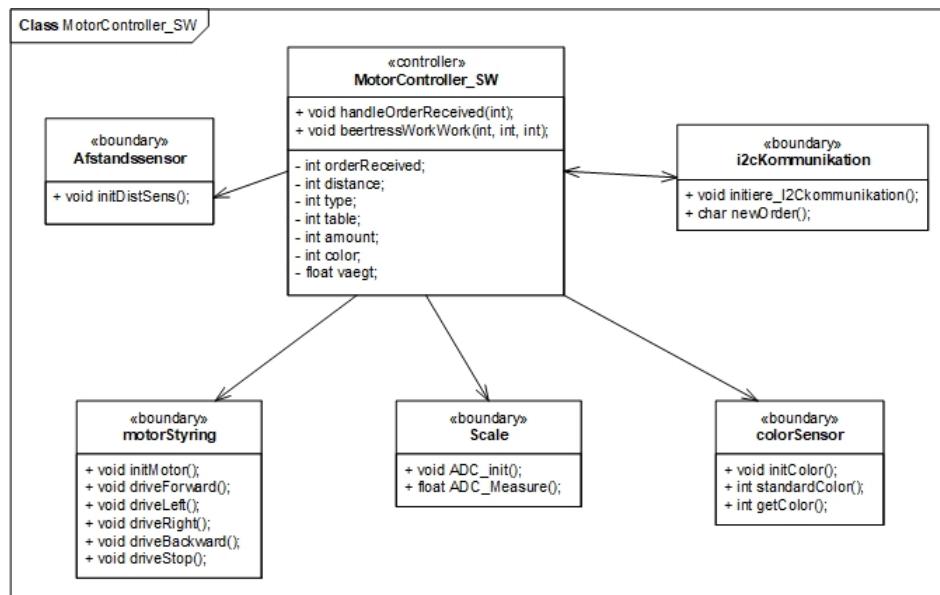
arbejdet med at lave forbindelsen nemmere. Softwaren i bruger-grænsefladerne består af c#,C++, css, bootstrap, og html.

Valget af at benytte .net desktop App er udelukkende fortaget, fordi man på det tidspunkt ikke havde kendskab til, de muligheder, som findes. At bruge en Windows App, som kører på en pc er en fin løsning, men det ville være smartere at have brugerflade, som ligger online på en server i stedet.

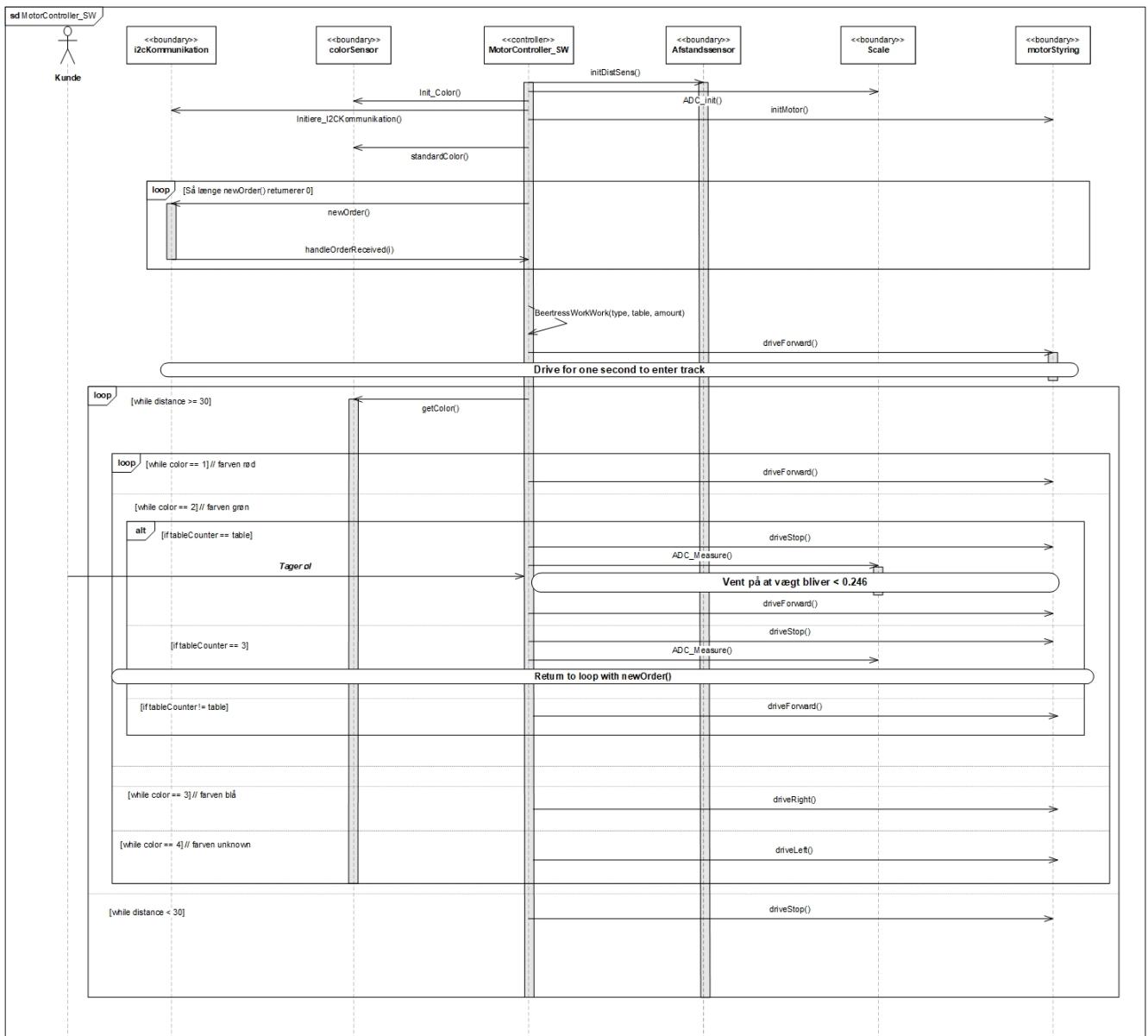
10.3 MotorController_SW

10.3.1 Software design

Figur 19 viser klassediagrammet for MotorController_SW. Her ses forbindelserne mellem MotorController_SW og de forskellige boundaries. Eftersom MotorController_SW er CPU'en er denne valgt som controller.



Figur 19: Klassediagram med funktioner



Figur 20: Sekvensdiagram af MotorController_SW

Som det kan ses på figur 20, starter det hele fra controlleren, MotorController_SW, som initierer alle sensorerne og motoren. Nogle af disse sensorer, f.eks. afstandssensoren, består af funktioner som ikke fremstår af sekvensdiagrammet. Disse bliver beskrevet i implementeringsafsnittet under den pågældende sensor.

MotorController_SW tjekker efter en ordre fra RPi inde i en for-løkke, og bliver i denne løkke så længe ingen ordrer er modtaget. Når en ordre modtages, bliver denne ordre først håndteret inde i Beertress-WorkWork(), som angiver hvor Beertress skal hen. Ved hjælp af farvesensoren navigerer Beertress derefter afsted mod det ønskede bord, og hhv. drejer eller køre ligeud alt efter hvilken farve underlaget har.

10.3.2 MotorController_SW Implementering

MotorController_SW består af følgende funktioner, som kan ses på tabel 12:

Funktion	Beskrivelse
int main()	Initierer motor og sensorer, kalder standardColor og tjekker om der er kommet en ny ordre via newOrder fra i2cKommunikation
void beertressWorkWork(int type, int table, int amount)	Sørger for at Beertress Roveren navigerer ned til det korrekte bord ved hjælp af de forskellige sensorer og motor. Når øl er taget af kunde, sørger funktionen for at Beertress Roveren kører tilbage til startposition
void handleOrderReceived(orderReceived)	Handle funktion som håndterer returværdien fra I2C og kalder beertressWorkWork() med korrekt type, bord og antal.

Tabel 12: Funktioner i MotorController_SW

Main i MotorController_SW kalder først init-funktionerne og standardColor funktionen, som er beskrevet under colorSensor-afsnittet. Derefter håndteres den int som er modtaget via I2C i for-loopet i en handler kaldet handleOrderReceived. Ved hjælp af tabel 14, vides det, hvad der skal sendes videre til funktionen beertressWorkWork(). Denne funktion indeholder tre parametre; type, table og amount.

Type bliver håndteret i en switch case, og eftersom der kun bruges øl, er der kun implementeret

en case med '1'. I denne case bliver en variabel, tableCounter sat til 0, og bliver efterfølgende talt en op, hver gang en grøn farve rammes. Når tableCounter bliver lig med table parametren, stopper Beertress indtil vægten noterer, at øllen er taget. Beertress kører derefter videre, og stopper når tableCounteren bliver 3, da startposition så er nået. Derefter sættes en PIN, som kaldes "Done" højt, for at indikere til RPi at startpositionen er nået. På samme tidspunkt sættes status på vægten på en read-buffer som er gjort klar via den indbyggede funktion I2C_SlaveInitReadBuf(), så RPi kan vide, om der er behov for ny øl. På den måde er MotorControlleren klar til en ny ordre. For at håndtere banen implementeres while-loops, som håndterer informationerne fra farvesensoren. Her er afstandssensoren også implementeret, så hvis denne registrerer noget under 30 centimer foran den, vil Beertress stoppe med at køre, indtil der er fri bane igen.

Koden kan findes i Kodebilag under main_psoc_done.

10.4 i2cKommunikation

10.4.1 Software implementering

Ved i2cKommunikation er der taget udgangspunkt i Beertress protokollen, og ved hjælp af switch cases, er protokollen blevet implementeret.

i2cKommunikation består af følgende funktioner, som kan ses på tabel 13 :

Funktion	Beskrivelse
void initiere_I2Ckommunikation()	Initierer I2C slaven, samt klargøre en buffer til at modtage ordre via I2C
char newOrder()	Håndterer den ordre der er modtaget via I2C

Tabel 13: Funktioner i i2cKommunikation

I2C-Slave komponenten i PSoC Creator har nogle indbyggede funktioner som kan findes i slavens datablad[3]. Den, som gør overførslen mulig, er funktionen I2C_SlaveInitWriteBuf(). Den fortæller hvilken buffer Masteren(RPi) skal sende til, samt størrelsen på antal bytes som kan modtages. Derudover er I2C-Slave komponenten sat til at have slave-adressen 0x08.

Funktionen newOrder() har til formål, at håndtere det som bliver sendt fra masteren til slaven, i forhold til Beertress protokollen. Ved inspiration fra databladet for slaven[3], er det blevet implementeret sådan, at når en succesfuld write dataoverførsel er overstået fra Masteren, altså når Masteren

har skrevet til slaven med data uden fejl(dette tjekkes via funktionen I2C_SlaveStatus()), så starter et switch case statement, som håndterer de modtaget bytes. Alt efter hvilke bytes der modtages i forhold til Beertress-protokollen, returneres en int på 1 til 6, eftersom der er 6 mulige udfald i dette tilfælde. Dette vil kunne skaleres til 255 forskellige cases for hver byte, så der er rigelig plads til skalering. Den int, der returneres, bliver så brugt i mainPSoC.

For at undgå fejl er der lavet en kort udvidelse til protokollen mht. retur værdien alt efter hvilken ordre der modtages. Eftersom dette kun er aktuelt i PSoC-delen er det ikke blevet indført i den overordnede protokol, da det ikke har noget med RPi'en at gøre. Denne udvidelse kan ses i tabel 14:

	Type	Bord	Antal	Retur værdi:
Bord 1, 1 øl:	0b00000001	0b00000001	0b00000001	0x1
Bord 1, 2 øl:	0b00000001	0b00000001	0b00000010	0x2
Bord 1, 3 øl:	0b00000001	0b00000001	0b00000011	0x3
Bord 2, 1 øl:	0b00000001	0b00000010	0b00000001	0x4
Bord 2, 2 øl:	0b00000001	0b00000010	0b00000010	0x5
Bord 2, 3 øl:	0b00000001	0b00000010	0b00000011	0x6

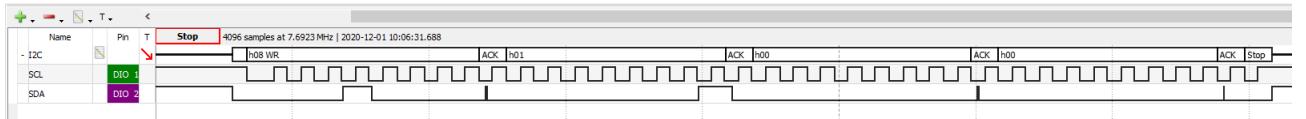
Tabel 14: Protokol med retur-værdi

10.4.2 Modultest af I2C

For at teste I2C-kommunikationen, blev der sat et simpelt kode dokument op i PSoC Creator og Linux. PSoC'en(Slaven) skulle kunne modtage data fra Masteren(RPi), og derefter flytte disse data over i en read-buffer, så Masteren kunne læse dem igen. RPi'en skulle derfor sende 3 bytes, vente 1 sekund og derefter læse 3 bytes, På den måde ville man kunne se, om dataoverførslen var succesfuld. En LED ville lyse, hvis den første byte var korrekt.

Der opstod dog det problem, at kun den første byte blev sendt ordenligt, mens de næste to bytes altid blev sendt som 0 og 0. Dette kan ses på figur 21. Efter mange test, bl.a. med brug af Analog Discovery, blev fejlen fundet og rettet - de to read- og write-buffere i linux-dokumentet var int arrays, og eftersom int består af 32 bit, fungerede overførslen ikke, da der skulle sendes 8 bit ad gangen. Da disse blev rettet til char arrays, fungerede I2C-Kommunikationen som forventet og ønsket.

Yderligere beskrivelse kan findes i Projektdokumentationen.



Figur 21: Test af I2C via Analog Discovery

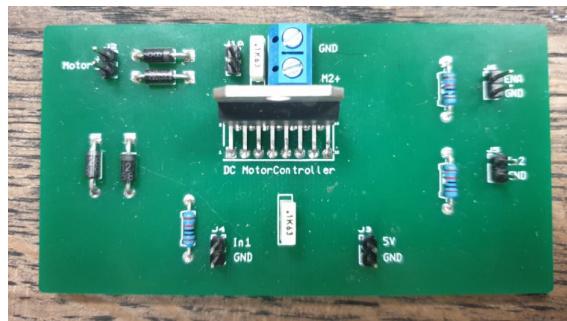
10.5 Motor og Motorstyring

10.5.1 Hardware design

Eftersom at køretøj blev lånt ved Embedded Stock[6] vil denne ikke blive beskrevet dybdegående. Den består af et aluminiumskarosseri og fire 12V DC-motor, der er sat sammen parvis. Disse styrer henholdsvis venstre og højre side af larvefødderne. Et billede af køretøjet kan ses i projektdokumentationen under Systemdesign og Implementering.

10.5.2 Hardware implementering

Efter at have lavet en simpel test på det udleverede køretøj fandt man frem til, at larvefødderne kørte invers af hinanden. Derfor blev der lavet en implementation med to H-broer[8] så man kunne vende polariteten af strømmen på DC-motorerne. Derved ville det være muligt, at få DC-motor til at køre i samme retning. Det udleverede H-Bro print fra anden undervisning fra semestret blev brugt, hvilket kan ses på figur 22, da det ikke ville være relevant eller tidsmæssigt forsvarligt at lave det selv i netop dette projekt. Man kan se i kredsløbsdiagrammet, at kredsløbet[7] skal forsynes med 5V. Kredsløbet er bygget op omkring L298, som er en Dual Full Bridge Driver. Hvis man kigger i databladet[4] på figur 23, kan man se at L298 kan forsynes med op til 50V.



Figur 22: H-Bro print

L298

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I, V_{EN}	Input and Enable Voltage	-0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive ($t = 100\mu s$) – Repetitive (80% on – 20% off; $t_{on} = 10ms$) – DC Operation	3 2.5 2	A
V_{SENS}	Sensing Voltage	-1 to 2.3	V
P_{TOT}	Total Power Dissipation ($T_{CASE} = 75^\circ C$)	25	W
T_{OP}	Junction Operating Temperature	-25 to 130	$^\circ C$
T_{STG}, T_J	Storage and Junction Temperature	-40 to 150	$^\circ C$

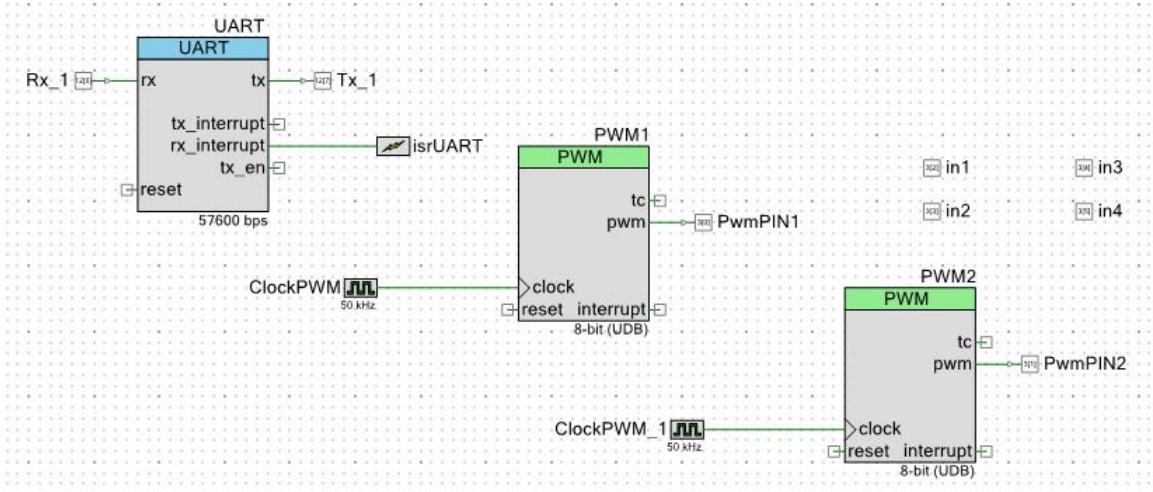
Figur 23: L298 Dual Full Bridge[4]

10.5.3 Software implementering

Som man kan se i sekvensdiagrammet for MotorController_SW på figur 20, skal motoren kunne køre fremad, til venstre og til højre, men eftersom man implementerer køretøjet med H-broer har der også været mulighed for at køre baglæns. Derfor bliver dette også implementeret i motorstyringen, så man fremadrettet kan skalere systemet til at navigere i alle retninger.

Koden til Motorstyring blev skrevet i PSoC Creator i C, da det ikke kan understøtte C++.

Topdesignet indeholder to PWMs, fire direction pins og en UART (som vil kun vil blive brugt i testøjemed). Hver H-bro vil blive styret af hver sin PWM og to direction pins til at styre retningen af strømmen. Topdesignet kan ses på nedestående billede:



Figur 24: Topdesign af Motorstyring fra PSoC Creator

Motorstyringen indeholder i alt seks funktioner, der blev defineret i en .h fil og implementeret i en .c fil og disse kan ses i nedestående tabel samt en beskrivelse af hver:

Funktion	Beskrivelse
void initMotor()	Initiere begge PWMs
void driveForward()	Sætter direction på pins til fremad og starter PWM med 100 procent på begge PWMs.
void driveStop()	Stopper begge PWMs
void driveLeft()	Starter PWM i venstre side med 100 procent og stopper PWM i højre side
void driveRight()	Starter PWM i højre side med 100 procent og stopper PWM i venstre side
void driveBackward()	Sætter direction på pins til bagud og starter PWM med 100 procent på begge PWMs.

Tabel 15: Funktioner i Motorstyring Software

Nedestående kodeudsnit viser koden til driveForward-funktionen. Samme fremgangsmåde er blevet brugt til de andre funktioner. Her ses det at PWMs initieres igen. Dette gøres i tilfælde af, at driveStop-funktionen er blevet kaldt og dermed har stoppet PWM. Begge PWMs bliver sat til 100

procent ved hjælp af WriteCompare. H-Bro, der styrer venstre side, får in1 og in2, der bliver sat hhv. høj og lav. Dette gør at retningen bliver fremadrettet. H-Broen, der styrer højre side får in3 og in4 og her sker det inverse ift. den venstre H-Bro, så også dens retning bliver fremadrettet.

```

1 void driveForward()
2 {
3     // Saetter direction paa DC motor ved hjælp af H-Broer
4     in1_Write(1);
5     in2_Write(0);
6     in3_Write(0);
7     in4_Write(1);
8     // Starter PWMs
9     PWM1_Start();
10    PWM2_Start();
11    // Saetter PWMs til 100%
12    PWM2_WriteCompare(100);
13    PWM1_WriteCompare(100);
14 }
```

Listing 1: Kodeudsnit for driveForward

Implementering af de andre funktioner kan ses i Motor under Kodebilag.

10.5.4 Modultest af Motor og Motorstyring

For at teste Motor og Motorstyring blev der implementeret en UART i en main, så man via en terminal (f.eks. RealTerm) kunne indtaste input, alt efter hvilken funktion man ønskede at teste. Dette blev gjort ved hjælp af switch cases, så hvis man f.eks. trykkede '1' i terminalen ville man teste driveForward osv. Dette blev programmeret til PSoC. Der blev valgt en PWM på 100 procent under hele testen, da hastigheden af køretøjet allerede ved maksimum PWM kørte forholdsvis sløvt tempo, så der var ingen grund til at teste med mindre. Der blev også i main initieret PWM ved hjælp af initMotor-funktionen.

Dernæst blev køretøjet med H-broer sat sammen med PSoC og forsynede de nødvendige dele med spænding på hhv. 9.6V, 5V og ground og åbnede terminalvinduet. Den printede de tilgængelige kommandoer ud i outputvinduet. Alle funktionerne virkede efter hensigten og Beertress navigerede som ønsket. En mere dybdegående observering af hver funktion kan ses i projektdokumentationen

under Test.

10.6 Farvesensor

Farvesensoren har til formål at navigere Beertress hen til bordene og tilbage til startposition. Sensoren er derfor en nødvendighed, når Beertrees skal kunne navigere rundt på banen alt afhængig af hvilken farve, som sensoren har registreret.

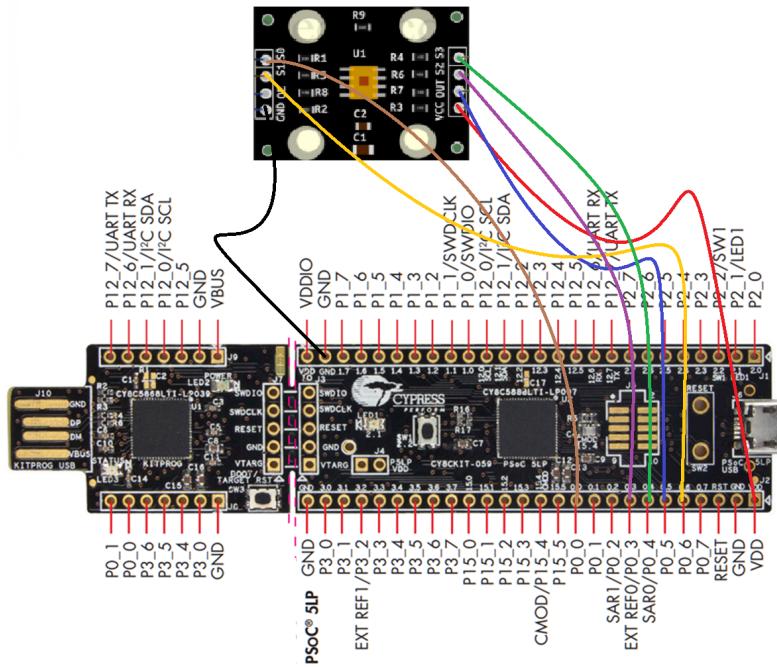
10.6.1 Hardware design

Det er tidligere beskrevet, at valget af farvesensor til dette projekt, er en TCS230[5]. Denne er lånt ved Embedded stock, hvorfor der ikke vil blive lavet en dybdegående beskrivelse af designet.

Selve sensoren, som ligger i midten, består af et 8x8-array fotodioder med 4 farvefiltre i toppen. Disse fire typer af filtre er rød, blå, grøn og clear. Derudover består den af otte pins. Disse pins består af en ground, VCC, output frekvens, OE og signalerne S0, S1, S2 og S3. S0 og S1 er output frekvens skalering, som er en intern clock chip, der tæller input signalet. Jo højere værdien er, desto højere er følsomheden. S2 og S3 er fotodiodetyperne, der afgør hvilken farve sensoren skal aflæse.

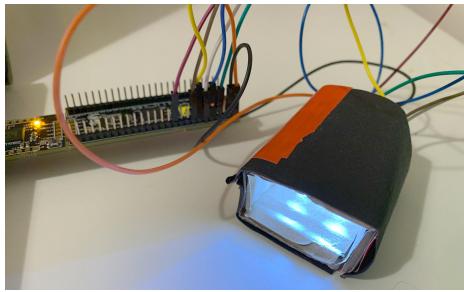
10.6.2 Hardware implementering

Farvesensorens pins forbides til PSoC'en, som kan ses på figur 25. Pin OE benyttes ikke i dette tilfælde, da den bruges til deling af flere enheder af en mikrocontroller-input. Topdesignet, bestående af signalerne, UART, timer og counter ses i projektdokumentationen.



Figur 25: Farvesensor TCS230 forbindelse til PSoC

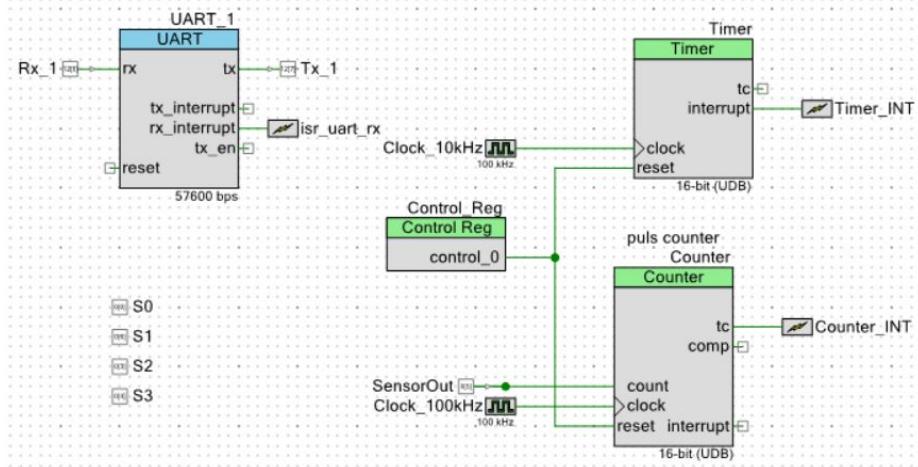
Derudover er der blevet monteret et stykke pap rundt om farvesensoren, for at skærme for generende lysindfald på fotodioderne. Farvesensoren er blevet placeret ca. 4 cm fra bunden af pap-monteringen da dette viser sig at give de mest stabile afmålinger fra denne afstand. På nedenstående figur 26 vises hvordan pap-stykket er monteret på sensoren



Figur 26: Color sensor inden i papkasse

10.6.3 Software implementering

Farvesensoren består blandt andet af UART, timer, og en counter, som kan ses på nedenstående Topdesign for colorSensor.



Figur 27: Topdesign af farvesensor fra PSoC Creator

Timeren og counteren har til formål, at omdanne output fra sensoren til en frekvens. UART er brugt til at lave løbende test, og for at kunne lave en endelig test i modultest. Disse, samt de resterende komponenter, kan ses uddybet i projektdokumentationen.

Derudover består implementering af softwaren af en række funktioner, som hver spiller en vigtig rolle, dog vil det kun være de mest væsentlige, som bliver beskrevet i dette afsnit. I tabel 16 kan man se de mest interessante funktioner fra colorSensor modulet.

Funktion	Beskrivelse
void initColor()	Initierer Counter og Timer
void read_color(void)	Omdanner input til frekvens gennem Counter og Timer
int get_freq(char color)	Her defineres signalerne S0-S3 og farvefiltrene tilføjes
int standardColor()	Aflæser startfrekvensen af hver farve via getFreq()
int getColor()	Aflæser den farve, som sensoren detekterer

Tabel 16: Funktioner i Color Sensor Software

En af de fundationale funktioner er get_freq(), som skal sætte scaling, samt udskrive værdier til de

enkelte farvefiltre, så frekvenserne kan registreres. To andre vigtige funktioner er standardColor(), og getColor(), som bruges til at registrere en standardfrekvens, og til at registrere farverne. Logikken der ligger i getColor funktionen, bruges til at vurdere hvilken farve der registreres på baggrund af frekvenserne. For at en frekvens bliver vurderet til at være en bestemt farve, skal den blandt andet være ændret med +10%. Dette gøres på baggrund af det arbejde og tests, der har været løbende. På figur 28 kan man se hvordan farvesensoren har registreret en standardfrekvens på hver farve (R0, G0, B0) ved neutralt underlag. Den nye frekvens opdateres nu hver gang getColor() bliver kaldt. Derudover vil der blive lavet sammenligning med de andre farvefrekvenser for at vurdere hvilken farve, der med størst sandsynlighed, er blevet registreret. Yderligere informationer vedrørende funktionerne kan ses, i projektdokumentationen.

10.6.4 Modultest af Farvesensor

Farvesensoren testes ved at pakke den ind i noget pap (Forklaring kan ses i Hardware implementering). Sensor med indpakning løftes 3cm fra bordet og holdes stabilt. Sensoren ligger 4 cm oppe i indpakningen, hvilket betyder at der er en samlet afstand på 7 cm fra sensoren til overfladen. Denne værdi er blevet valgt, da der blev testet med hvor langt oppe sensoren kunne sættes fra overfladen før den ikke længere detekterede farven. Sensorindpakningen kunne sagtens løftes op til 5 cm, men dog var aflæsninger ikke stabile. På trods af at sensoren stadig detekterer farverne, når den er løftet 5 cm, sættes den ned til 3 cm, for at der ingen fejlmålinger opstår.

Der anvendes farvet karton til at teste de forskellige farver, og disse sættes ind under sensoren en efter en. Ved at benytte den allerede implementeret UART, kunne der ligeledes laves en samlet modultest.

Der oprettes 4 switch cases i main, og alt efter hvilket tal der returneres fra getColor(), udskrives den detekterede farve på konsolvinduet i RealTerm. Når sensoren ikke detekterer farverne rød, blå og grøn, udskriver den, som forventet, 'unknown'. For yderligere information vedrørende modultest af farvesensoren henvises til projektdokumentationen. På figur 28 vises et screendump af konsolvinduet med farvefrekvenserne og den detekterede farve. Her kan det ses, at rød bliver detekteret, da frekvensen for rød stiger fra 72 (standardfrekvens) til 84, hvilket er en stigning på over 10%. I koden implementeres en counter, som sørger for at frekvenserne blive aflæst to gange, inden den udskriver en farve. På denne måde sikres der, at den korrekte farve bliver detekteret, og at der ikke opstår en fejl. Dette ses eksempelvis, når grøn bliver detekteret. Inden der udskrives '2GREEN', aflæses frekvenserne to gange, hvor hvor G0 stiger med over 10%. Ligeledes vil blå og unknown blive

dtekteret.

Figur 28: Output af farvefrekvenser og den detekterede farve på konsolvindue

10.7 Afstandssensor

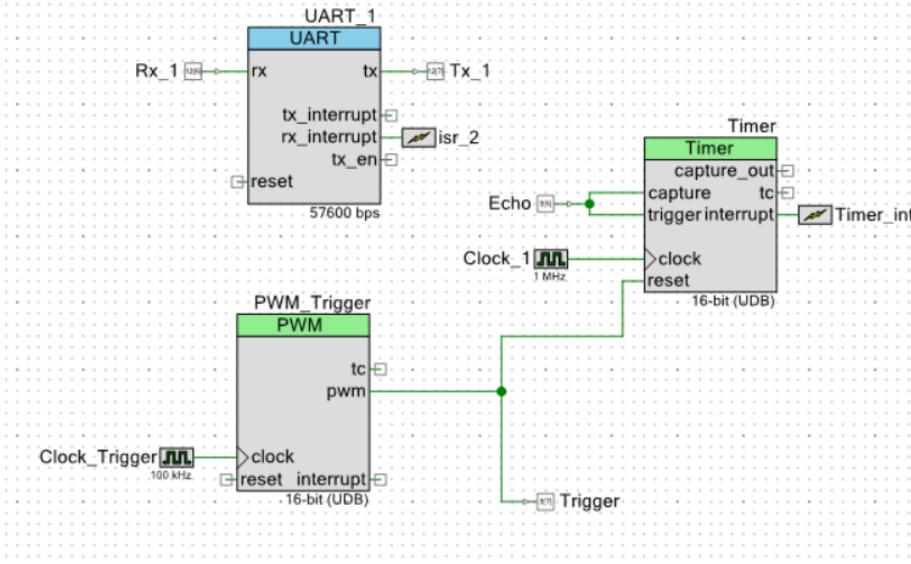
10.7.1 Hardware design

Da der hardwaremæssigt ikke er foretaget ændringer på afstandssensoren, henvises der til databladet [2], og afstandssensoren beskrives derfor kun kort og overfladisk.

Sensoren består af 4 pins, VCC, Ground, Trigger og Echo. For at aktivere sensoren sendes et højt signal til trigger i 10us. Det får den til at udsende en lydbølge, som består af 8 impulser på 40 kHz. Straks efter impulsene er sendt, går signalet på Echo højt, og når lydbølgjen kommer tilbage, går signalet lavt. Tidsperioden hvor signalet på Echo har været højt, kan derefter konverteres til en afstand ved at kende lydens hastighed.

10.7.2 Software implementering

Topdesignet, som ses på figur 29, indeholder en PWM, som er tilsluttet 'Trigger'-pin. Den sørger for, at sende en impuls på 10us hver periode.

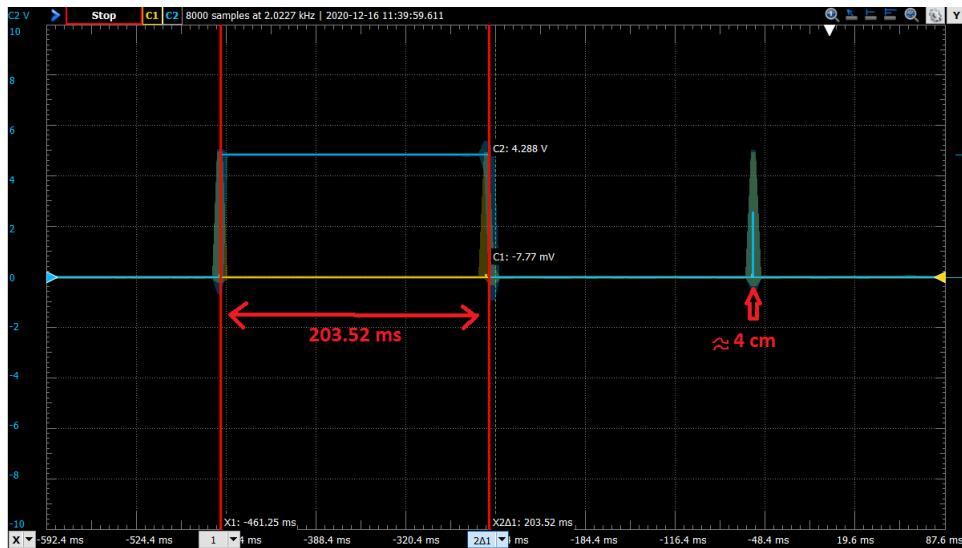


Figur 29: Topdesign af afstandssensor fra PSoC Creator

Inde i den digitale PWM-komponent er én periode sat til 200ms. Samtidig nulstilles timeren og når 'Echo' går lav igen, gemmes værdien af timeren i 'capture'-registeret. Derefter bliver et interrupt aktiveret, som gemmer værdien i 'capture'-registeret i en variabel. 'Clock' er sat til 1 MHz, således at 1 count svarer til 1 us.

I datasheetet til HC-SR04 anbefales det, at vælge en målecyklus på mere end 60 ms, for at undgå at et trigger-signal sendes, mens sensoren stadig er ved at afvente en tidligere sendt lydbølge.

Ved målingen på figur 30 ses det dog, at echo-signalet kan vente op til 203ms, for at lydbølgen kommer tilbage. PWM-komponenten er derfor valgt til 300ms, for at sikre at et triggersignal ikke bliver sendt, mens sensoren venter.



Figur 30: Måling som viser, når der ikke detekteres et objekt

Funktion	Beskrivelse
void initDistSens()	Initierer PWM og Timer
void calcDist()	Aflæser capture-register og omregner til distance
int getDistance()	Returnerer distancen
void outputDistanceToUART(int distance)	Outputter distancen ud til terminal

Tabel 17: Funktioner i Afstandssensor

Der implementeres 4 funktioner, initDistSens(), calcDist(), getDistance() og outputDistanceToUART(). Den vigtigste er calcDist(), som ses i listing 2, som tager den gemte capture-værdi og omregner denne til en afstand i cm, hvor det antages at stuetemperaturen er på $20C^o$:

$$afstand(cm) = captureVærdi \times 0.034(cm/us)/2$$

```

1 void calcDist()
2 {
3     int capture = Timer_ReadCapture(); //Capturing timer count
4     distance = (65535 - capture) * 0.017; //Calculating distance in cm
5     return distance;
6 }
```

Listing 2: Kodeudsnit for calcDist()

Et problem som afstandssensoren har, er, at når den ikke registrerer et objekt inden for 400cm, vil den returnere en afstand på 3-4cm på trods af at oscilloskopet viser en længde på 203.52ms, som svarer til en afstand på ca. 3400cm.

Det ses på figur 30, at det kan skyldes, at når sensoren ikke modtager et signal tilbage, vil echo-signalen sende en meget kort impuls, som restarter timeren, og gemmer en ny værdi svarende til 4cm.

10.7.3 Modultest

Afstandssensoren blev testet ved at tilslutte et oscilloskop og benytte et målebånd til at måle afstand. Først blev sensoren testet i stationær tilstand. Et objekt blev, ved at måle med et målebånd, placeret 180cm væk fra afstandssensoren. Oscilloskopet viste tidsperioden hvor echo var høj. Denne tidsperiode blev omregnet til en afstand med fornævnte formel i afsnit 10.7.2.

Den blev derefter sammenlignet med afstanden målt med målebåndet, som burde være ens. Terminalen blev også observeret samtidig med, som det ses på figur 31 .



The terminal window displays a series of text lines, each consisting of the word "Distance:" followed by "50 cm" and a carriage return character. This pattern repeats approximately 20 times, indicating the sensor is detecting an object at a fixed 50cm distance.

```

Distance: 50 cm
```

Figur 31: Terminal

Som det ses på figur 31, er sensoren forholdsvis præcis, hvor den svinger med +/-2%.

Herefter blev sensoren testet i mobil tilstand. Samme opstilling blev benyttet som før, men i denne test benyttes målebånd ikke. Her var forventningen, at afstanden ændrede sig, som objektet bevægede sig tættere på. Objektet bevæges tættere på, mens der tages 20 målinger, som er efterfulgt af hinanden. Størrelsen på dem alle observeres i terminalen, og måles på oscilloskopet for bekræftelse, og de bør blive mindre og mindre.

```
Distance: 48 cm CRLF
Distance: 43 cm CRLF
Distance: 42 cm CRLF
Distance: 40 cm CRLF
Distance: 36 cm CRLF
Distance: 33 cm CRLF
Distance: 29 cm CRLF
Distance: 26 cm CRLF
Distance: 24 cm CRLF
Distance: 22 cm CRLF
Distance: 19 cm CRLF
Distance: 17 cm CRLF
Distance: 15 cm CRLF
Distance: 14 cm CRLF
Distance: 11 cm CRLF
Distance: 10 cm CRLF
Distance: 8 cm CRLF
Distance: 7 cm CRLF
Distance: 6 cm CRLF
Distance: 5 cm CRLF
```

Figur 32: Terminalvindue hvor afstanden vises, mens objekt er i bevægelse

Det ses i figur 32, at sensoren kan detektere og hurtigt nå at opdatere afstanden på objektet.

Der beskrives i datasheetet for HC-SR04, at sensoren er dårlig til at detektere bløde objekter. Der henvises til projektdokumentation SystemTest, Afstandssensor for denne test.

10.8 Vægtsensor

10.8.1 Hardware design

Vægtsensoren som har formålet til at veje med vægten af ”produktet”, hvilket i dette tilfælde er øl som der skal serveres. Når vægtsensoren detektere at vægten ændrer sig skal dette sendes videre til PSoC der skal behandle signalet. Vægtsensoren modulet består af en load cell og en amplifier med formålet at øge signalet output der kommer fra load cellen. Komponenter der anvendes kan ses i projektdokumentationen under Systemdesign og Implementering. Da der er begrænset mulighed på

hvor meget vægt der kan komme på Beertress robotten vedrørende vægt, var der valgt at anvende en 1 kg load cell. Det var ikke nødvendigt at anvende mere end dette da der kun bliver målt 1 øl dåse.

10.8.2 Hardware implementering

Vægtsensor blev forsynet med 5V via spændingsregulator med en tilhørende ground forbindelse til load cell amplifieren. Load cell amplifieren giver så spændingen videre til load cell. Load cell har så S+ og S- hvilket er spændingsforskellen som forstærkes. Efter signalet blev forstærket sendes signalet videre til PSoC'en ved hjælp af Aout pin. Aref pin bliver anvendt som ADC reference der forbundet imellem Load cell amplifier og PSoC. Til at beskytte input pin på PSoC bliver der anvendt en 5V pin som er forbundet imellem amplifier og PSoC, hvor der også er tilsluttet en tilhørende fælles ground imellem load cell amplifier og PSoC. Vægtsensoren blev kalibreret i en tidligere undervisnings øvelse, hvor der blev indstillet load cell amplifieren og der blev fundet den generelle funktion som konverterer ADC signalet om til vægten i kg. Kalibreringen og yderligere forklaring og figurer kan ses i projektdokumentation under Systemdesign og Implementering.

10.8.3 Software implementering

Softwaren for vægtsensoren har det formål, at kunne initialisere ADC_SAR og have en funktion, der kan måle vægt og returnere dette tilbage. Koden blev skrevet i PSoC Creator i C. Topdesign har en ADC_SAR, som har et ADC-input. Dette ADC-input kommer fra Aout. Vref kommer fra Aref på load cell amplifieren. Ydligere forklaring af funktioner kan findes i projektdokumentation under Systemdesign og implementering.

Funktion	Beskrivelse
void ADC_init()	Initierer ADC_SAR og starter konverteringen så der bliver målt spændingen der kommer ind fra Aout på PIN_ADC_in pin
float ADC_Measure()	Håndtere udregningen af ADC signalet der kommer ind og konverttere det til kg og returnere dette

Tabel 18: Funktioner i Scale

10.8.4 Modultest af Vægt

For at teste Scale programmet at det virker som forventet blev der testet ved hjælp af en UART, og en konsolvindue som kan udskrive værdien af ADC og vægten der konverteres om til. Ved hjælp af kalibreringen kan der omregnes direkte om til kg og der skal ses om vægten er korrekt lavet om til kg. En gennemgang af testen kan ses i projektdokumentation under System Test.

Faktisk vægt	Returværdi	Observering
Vægt på 1 kg	0,9982 kg	1 kg blev placeret og den målte ADC værdi konverteres til kg.
Vægt på 0,502 kg	0,4977 kg	0,502 kg blev placeret og målte ADC værdi konverteres til kg.
Vægt på 0,301 kg	0,3017 kg	0,301 kg blev placeret og målte ADC værdi konverteres til kg.
Vægt på 0,156 kg	0,1507 kg	0,156 kg blev placeret og målte ADC værdi konverteres til kg.

Tabel 19: Scale Test

Efter at have testet vægten kom der frem til at vægten vejede som forventet. Blev der testet ved hjælp af en LED på PSoC, at ændringen af vægten kan tænde og slukke en LED. Ved at have en grænse på hvis vægten er under 300 g vil LED ikke lyse, hvor hvis den er over vil LED lyse.

11 Test

Nedenstående afsnit omhandler test af systemet. Grundet tidspres, har det ikke været muligt at implementere forbindelse mellem Personalegrænsefladen og systemet, så der blev i stedet fokuseret på at oprette forbindelse mellem Kundegrænsefladen og systemet.

11.1 Fremgangsmåde

I software delen, er der først blevet lavet modultest på alle de forskellige moduler, så sensorerne, motorne, I2C og kunde- og personalegrænsefladerne er blevet testet individuelt. Derefter blev der påbegyndt en integrationstest med motoren i centrum, da de fleste test var afhængig af denne, før

at man kunne få en idé om, om systemet fungerede. Hvert modul som havde forbindelse til PSoC'en blev testet med motoren en af gangen for at teste funktionaliteten, og til sidst blev alle modulerne sat sammen i en systemtest. Der er altså primært benyttet en "bottom-up" strategi til integrationstesten.

11.2 Integrationstest

Først blev alt kode samlet i en main, hvorefter hvert moduls funktionalitet blev udkommenteret en efter en, så det blev muligt at teste hvert modul sammen med motoren. Til sidst blev alle modulerne udkommenteret, og der blev observeret, om Beertress kunne fungere i systemtesten. Integrationstesten kan ses i tabel 20.

Grunden til, at farvesensor og afstandssensor ikke blev sat ovenpå I2C var, at det tog lang tid at få forbindelse til hjemmesiden. Eftersom at I2C'en fungerede blev der vurderet, at der ikke var behov for at teste den igen før systemtesten.

Moduler under test	Beskrivelse	Observering
Motor, I2C og RPi	Sender 6 forskellige kommandoer via kundegrænsefladen, og observerer om Beertress gør 6 forskellige ting.	Beertress laver 6 forskellige ting, alt afhængig af modtaget ordre.
Motor, farvesensor og afstandssensor	Der blev testet om Beertress kunne navigere rundt på banen ved hjælp af farvesensoren, og testede om afstandssensor virkede ved at sætte hånden ind foran	Beertress navigerer rundt på banen som ønsket. Når man satte hånden ind foran afstandssensoren stoppede Beertress, og kørte igen når man fjernede hånden.
Vægtsensor	Vægten blev testet ved at stille en øldåse på vægten og når den blev fjernet skulle LED på PSoC lyse op	Øldåsen blev fjernet og LED lyste som forventet.

Tabel 20: Integrationstest af de forskellige moduler

11.3 Systemtest

I systemtesten bliver det testet, om alle modulerne kan snakke sammen og fungere som en helhed. Eftersom denne prototype kun kan indeholde 1 øl, bliver der kun testes med en øl. Alle moduler blev udkommenteret, og der blev først via hjemmesiden sendt en ordre om 1 øl til bord 1 og derefter en øl til bord 2. Systemtesten kan ses i tabel 21.

Beskrivelse	Observering
Bestilling af 1 øl til bord 1	Bilen kører hen til bord 1 og stopper. Når øllen tages, kører Beertress videre til start og stopper. Når ny ordre sendes, kører Beertress igen.
Bestilling af 1 øl til bord 2	Bilen kører forbi bord 1 og kører videre til bord 2 og stopper. Når øllen tages, kører Beertress videre til start og stopper. Når ny ordre sendes, kører Beertress igen.

Tabel 21: Systemtest af Beertress

12 Resultater

12.1 Accepttest af Use Case 1: Initialiser Beertress

Der har i accepttesten for denne use case været fokus på at initiere Beertress vha. personalegrænsefladen. Da denne ikke er blevet integreret med resten af systemet, er det ikke lykkedes at gennemføre 2 ud af 3 accepttests i UC1.

3. accepttest, som omhandler, at frakobling af batteri vil medføre, at Beertress viser en log og fustagevægt, er ikke implementeret, og fejler derfor også.

12.2 Accepttest af Use Case 2: Udkiftning af fustage

I accepttesten for denne use case har der også været fokus på personalegrænsefladen, som ikke er integreret med systemet. Derudover er accepttestene sat op til at teste med en fustage. Grundet en afgrænsing af projektet, benyttes en dåseøl i stedet. Begge dele medfører, at alle accepttests fejler.

12.3 Accepttest af Use Case 3: Bestilling

I accepttesten for use case 3 er 2 ud af 7 accepttests OK. 4 ud af 5 accepttests fejler, da de ikke kan testes, da der ikke er implementeret en betalingsdel. Den sidste af de fejlende accepttests, ikke går igennem, da der ikke er implementeret en 'TILBAGE'-knap efter QR-kode er vist.

De to accepttests som går igennem, omhandler åbning og lukning af hjemmesiden på eget device.

12.4 Accepttest af Use Case 4: Betjening

Alle 5 accepttests for use case 4 fejler. Her fejler 2 accepttests, da personalegrænsefladen ikke er implementeret. Derudover fejler 2 andre, da der hverken er fustage, hane eller en 'Server øl'-knap. Sidste accepttest fejler, da Beertress ikke kan placeres et vilkårligt sted på banen og derefter køre tilbage til startposition. Denne funktionalitet er ikke implementeret.

12.5 Accepttest af Use Case 5: Slukning af Beertress

I use casens accepttest har der været fokus på at personalegrænsefladen viser beskeder tilbage til personalet, når Beertress slukkes. Da personalegrænsefladen ikke er integreret med systemet og batteriet skal tages ud og oplades, fejler alle 3 accepttests.

12.6 Accepttest af funktionelle krav

Der har i accepttesten for de funktionelle krav været fokus på personalegrænsefladen og opladning. Personalegrænseflade er ikke integreret med systemet og opladning kan kun ske ved at tage batteri ud af Beertress. Dette medfører at alle 4 accepttests fejler. Én accepttest er dog tæt på at være OK, som omhandler at Beertress skal kunne styres trådløst. Men da det er beskrevet, at der forventes at ordren kan ses på personalegrænsefladen, går denne accepttest ikke igennem.

12.7 Accepttest af ikke-funktionelle krav

De ikke-funktionelle krav indeholder 23 accepttests, hvoraf 10 er OK. En stor del af de accepttests som er OK, er krav til hvad Beertress skal indeholde af software og fysiske komponenter. Derudover omhandler 2 af dem, tiden det tager, at udskifte og reparere dele af Beertress.

To krav stillet til batteriet, samt funktionaliteten at Beertress kan dreje 360 grader er OK.

Nogle af de tests som har fejlet, har været tests, som har krævet testpersoner, der ikke har haft kendskab til Beertress. Derudover fejler flere også, da der har været behov for en fustage. Der er enkelte accepttests, som ikke er blevet testet. Det er bl.a. kommunikationsrækkevidden og vægten på Beertress. Én accepttest som blev udført, men ikke godkendt, var hastigheden på Beertress, som skulle opnå en hastighed på mindst 1 km/t.

13 Diskussion

Det ses ud fra resultaterne, at mange af de ønskede mål ikke er opnået. Størstedelen af testene under 'Accepttest' er fejlet. Det skyldes, at mange af testene har indeholdt en personalegrænseflade, som ikke nåede at blive integreret med systemet. Derudover blev der heller ikke implementeret en betalingsdel.

Gruppens udgangspunkt til Use Casene og derfor også accepttestene, har været for fokuseret på funktionaliteten i en restauration med henblik på restaurationspersonalet, i stedet for at fokusere på, at få det basale ved en robot til at fungere efter hensigten. Så kunne man derfra have udvidet til mere funktionalitet med et personalemæssigt synspunkt, hvis tiden havde været til det. Dette er der en del læring i.

Det er derimod lykkedes, at implementere og integrere de fleste moduler, så der derved er en robot, som kan styres trådløst. Dette afspejler sig ved at flere af accepttestene for ikke-funktionelle krav kan vurderes som OK. Generelt kan det siges, at det udviklede system viser ideen, som er beskrevet i projektbeskrivelsen. Hjemmesiden fungerer som ønsket, og når antal øl og bordnummer bliver valgt på mobilen, startes motoren og navigeres igennem kørebanen, indtil det valgte bord nås. Herved undgås den fysiske kontakt mellem den, som giver øllen og den, som modtager øllen.

Som noget nyt i dette semesterprojekt er der, som tidligere nævnt, gjort brug af Scrum. Scrum har hjulpet med at fordele arbejdsopgaver ud, og først da Scrum blev introduceret, begyndte udviklingen af de enkelte moduler at tage fart, da alle nu havde ét arbejdsmønster at fokusere på. Dette har klart bidraget til at få dele af Beertress til at fungere, og dermed succesfuldt fuldføre dele af accepttesten.

14 Konklusion

I 2020 har COVID-19 præget verden og den måde vi omgås på. Derfor er der i dette projekt prøvet at komme en af de udfordringer, en verdensomspændende epidemi kan forårsage, til livs. Der er blevet udviklet en robot ved navn Beertress, som kan navigere rundt på en bane ved hjælp af en farvesensor, undgå forhindringer ved hjælp af en afstandssensor, notere om der er behov for påfyldning af øl ved hjælp af en vægtsensor, og modtage ordre fra en kunde via en hjemmeside. For at muliggøre dette er der blevet gjort brug af en RPi og en PSoC, som har kunnet håndtere alle disse data.

Hvis man ser på accepttesten, vil det noteres, at størstedelen af testene er fejlet. Den hovedsaglige årsag kan skyldes, at der ikke blev lavet forbindelse til personale grænsefladen grundet tidsbegrænsninger, på trods af at den var en vigtig del af systemet. Personalegrænsefladen er essentiel i forhold til betydningen for selve projektet med en robot, som skal navigere rundt i en restaurant. Derudover har der også været en klar begrænsning i embedded stock, hvor udvalget har været begrænset. Eksempelvis har det ikke været muligt at få en robot, som kunne klare en vægtbelastning på fx 15 kg. Hvis dette var taget med i overvejelserne inden, var en del af kravene og forventningerne blevet nedskaleret, i forhold til hvad de var fra start.

Med tanke på at det nuværende semester er notorisk kendt for at være svært belastet, skulle man måske også have haft det in mente fra start, og have haft et lidt lavere ambitionsniveau. Tiden har

været knap, selvom det har været muligt at mødes hver fredag. Det skal heraf siges, at eftersom dette semesterprojekt er det første projekt, hvor det har været muligt selv at bestemme emnet, er det svært at vide, hvor meget man egentlig kan nå. Set på denne måde er projektet ikke blevet skudt helt forbi målet, selvom accepttesten kan indikere anderledes.

Når alt dette er sagt, er det til gengæld også værd at tage et blik på projektbeskrivelsen og projektafgrænsningen ved MoSCoW. Her kan det faktisk ses, at der nås i mål med mange af de visioner gruppen havde for selve projektet. Det er f.eks. blevet muligt at navigere rundt på en bane på baggrund af en kommando fra en hjemmeside(bestilt fra et bord), hvor der samtidig tages hensyn til udefrakommende forhindringer. Disse visioner og beskrivelser kan man diskutere om skulle have haft større indflydelse i Use Cases og accept test. Når der ses på dette må det alligevel konkluderes, at man kommer i mål med mange af de visioner og mål, som er blevet sat op for projektet.

Gruppen har tilegnet sig nye færdigheder og evner inden for nye områder af IT, da dette har været nødvendigt for at implementere en robot, som skal have de færdigheder, som Beertress skulle have. F.eks. har man tilegnet viden omkring WebSocket, HTML, diverse sensorer osv. Derudover er der blevet stiftet bekendtskab med Scrum som udviklingsproces, hvilket gruppen vil vurdere til at være en succes. Derudover er emner fra undervisningen blevet implementeret i projektet, hvilket har givet en større forståelse af emnerne. Disse er bl.a. Thread-Communication og I2C, samt enhederne RPi og PSoC'en. Derudover er Linux blevet brugt til implementering af koden på RPi, og eftersom Linux er et så udbredt operativ system i samfundet, er dette også vigtig læring.

15 Fremtidigt arbejde

Gruppen har ikke fået implementeret forbindelsen mellem personalegrænsefladen og resten af systemet grundet tidspres, men dette ville helt klart være det næste punkt at gå til. Eftersom selve personalegrænsefladen er implementeret vil dette også være overskueligt inden for en forholdsvis kort tidsramme. Derudover har gruppen ikke fået implementeret en aktuator, som skulle fungere som betjeningsknap. Dette blev nedskaleret i projektet, da det på et tidligt stadie blev klart, at det lå udenfor det realistiske område i forhold til hvad man ville kunne nå. Det samme gælder vægten og selve robotten, som i en endelig version skulle kunne håndtere mere vægt og køre med større hastighed.

Gruppen har fået implementeret en QR-side på kundegrænsefladen, men grundet besværliggørelsen ved oprettelse af mobilepay eller andre betalingsformer, blev dette ikke prioriteret eller realiseret i projektet. Det er dog en vigtig og essentiel del af Beertress systemet, og i tilfælde af en færdig udgave, ville dette selvfølgelig skulle implementeres. Alternativt skulle Beertress kunne notere hvor mange øl et bord har købt, og derefter sende dette til personalegrænsefladen, så en tjener vil kunne varetage betaling.

Generelt har det været et fokus, at Beertress skal være skalerbart, så derfor vil fremtidigt arbejde også kunne være, at Beertress skal kunne servere andre typer af drikkevarer i stedet for øl. For den sags skyld kunne Beertress også servere snacks, selvom navnet i så fald skulle genovervejes.

Litteratur

- [1] *ASE-Modellen, side 3, figur 1.* https://studerende.au.dk/fileadmin/studerende.au.dk/ST/diplomingenior/Filer/Projektvejledning_-_Katrinebjerg/Vejledning_til_udviklingsprocessen_for_semesterprojekt_3_V1_10.pdf.
- [2] *Datablad for HC-SR04.* <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>.
- [3] *Datablad for I2C-slave komponent i PSoC.* <https://www.cypress.com/file/130946/download>.
- [4] *Datablad for L298 Dual Full Bridge.* <https://www.st.com/resource/en/datasheet/1298.pdf>.
- [5] *Datablad for TCS230.* <http://www.w-r-e.de/robotik/data/opt/tcs230.pdf>.
- [6] *Embedded Stock til lån af elektroniske dele.* <https://stockmanager.ase.au.dk/>.
- [7] *Kredsløbdiagram over H-Bro udleveret af GFV.* <https://bit.ly/3nosHq3>.
- [8] *Teori om H-Broer.* <https://blog.digilentinc.com/what-is-an-h-bridge/>.