AARHUS SCHOOL OF ENGINEERING

E4PRJ4 - iFollow

Systemdesign

Navn:	Studienummer
Jesper Toft Jakobsen	201708777
Hans Krag Halberg	201707343
Mikkel Jensen	201708684
Mathias Juhl Johansen	201708948
Nicklas Grunert	201707773
Lucas Paulsen	201707663

Dato: 29-05-2019

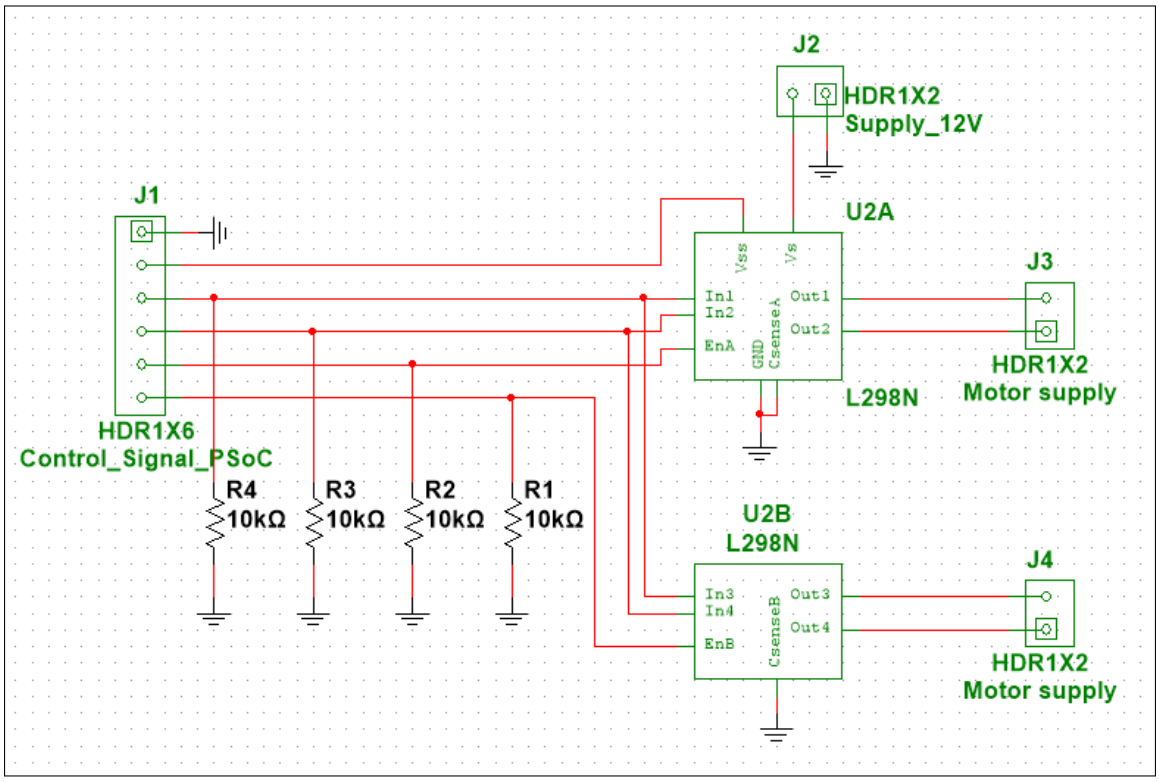
Indholdsfortegnelse

Ordforklaring

Forkortelse/Ord	**Forklaring**
iFollow	Det samlede system, prototypen, robotten
WebApp	Webapplikation - Systemets GUI
GUI	Graphical User Interface - Grafisk brugergrænseflade
μ-controller	Micro controller - f.eks. PSoC, RPi
PSoC	Programmable System on Chip (PSoC 5LP)
RPi	Raspberry Pi Zero W
GPS	Global Positioning System - enhed for systemets lokationsdata
SysML	Systems Modeling Language - brugt til HW-arkitektur
UML	Unified Modeling Language - brugt til SW-arkitektur
BDD	Block Definitions Diagram - beskrivelse af system-blokke
IBD	Internal Block Diagram - beskrivelse af forbindelser
HTML	HyperText Markup Language - Sprog til opsætning af WebApp-indhold
CSS	Cascading Style Sheet - Sprog til udseende af WebApp-indhold
JavaScript	Dynamisk programmeringssprog til WebApp'ens funktionalitet
API	Application Programming Interface, grænseflade mellem forskellig software
DC	Direct Current
IRT	Introduktion til Reguleringsteknik
jQuery	JavaScript-bibliotek til interaktion og animation af elementer
Node.js	Runtime system til udvikling af server-side webapplikationer.
Socket.io	JavaScript-bibliotek til kommunikation mellem server og client
Raspbian	Styresystem, som kører på Raspberry Pi
mySQL	Flertrådet SQL-databaseserver som understøtter flere brugere
MariaDB	Afgrening mySQL database håndteringssystem
MyPHPadmin	Redskab til håndtering af SQL database
DatabaseApp/ Databaseapplikation	Systemets positionsprogram til nedhentning og afkodning af GPS positioner

1 Hardware design

I denne sektion vil motorernes hardware design blive gennemgået og forklaret. På figur 1 herunder kan designet for motorprintet ses.

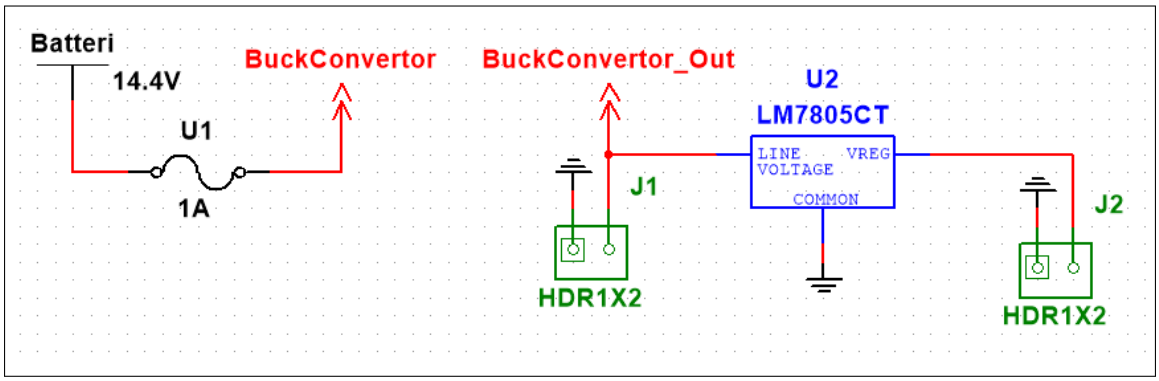


Figur 1: Design for motorprint

I designet er der blevet brug en H-bro IC, som gør det muligt skifte rotationsretning på motorerne og styre hastigheden på dem. Denne IC er valgt ud fra den meget simple opsætning, da det ville kræve en del transistorer og MOSFET's at udvikle dette selv. Udover dette kan denne IC holde til en konstant strøm på 4 ampere, hvilket er meget mere end der er brug for på projektet.
Udover IC'en er der også blevet tilføjet nogle pulldown modstande, som gør at vores output altid vil være 0 volt, hvis systemet ikke trækker den høje.

1.1 Powersupply

I denne sektion vil powersupply/ splitteren blive forklaret. Den vigtigste egenskab i denne blok er at konvertere spændingen fra 14.4 volt ned til hendholdsvis 12 volt og 5 volt. Designet kan ses på figur 2 herunder.



Figur 2: Design for powersupply/ splitter

På figuren kan man se at spændingen fra batteriet (14.4 volt) er supply voltage. Denne spænding går herefter ind i en DC-DC konvertor, som nedkonvertere spændingen fra 14.4 volt til 12 volt, da motorerne på iFollow ikke kan klare en højere spænding end 12 volt. Grunden til at der bliver brugt en DC-DC konvertor, fordi en DC-DC er en meget effektiv nedkonverterings metode, og når strømmen er så høj som den er vil tabet i en serieregulator være rimelig høj.
Herefter var det nødvendigt at nedkonvertere spændingen yderligere. Dette er grundet at microcontrollerne skal bruge en spænding på 5 volt. Her er der blevet brugt en serieregulator. Grunden til dette valg er fordi de er meget simplere end en DC-DC konvertor og de er meget billigere. Dog kan der også blive afsat en stor effekt i dem. Derfor blev effekttabet i serieregulatoren udregnet. Udregningen af denne effekt er en meget simple måde at udregne tabseffekten, men den vil give et godt anslag om det kan lade sig gøre. Udregningen kan ses på formel 1 herunder.

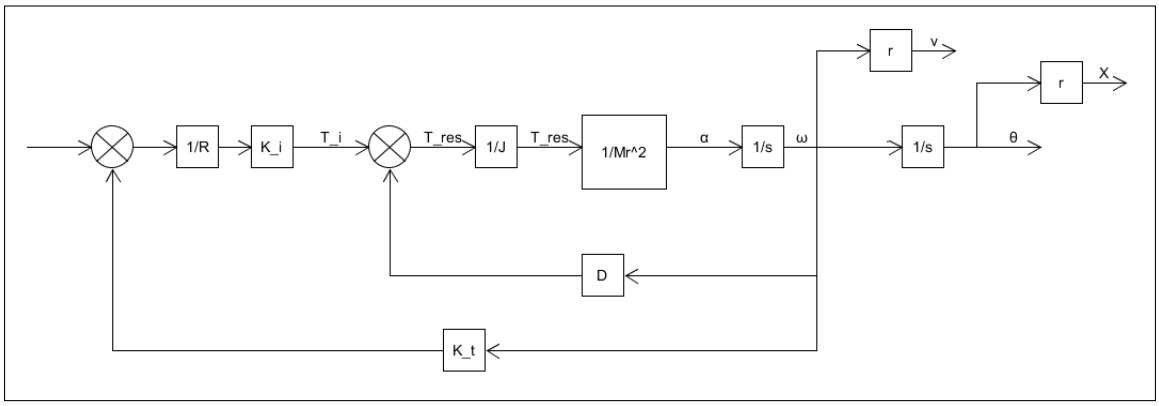
$$P_{Tab} = U_{DC} * I_Q + (U_{DC} - U_O) * I_O = 1.146Watt \quad (1)$$

I formlem er Udc = 12 volt, Iq = 8 mA (Aflæst i datablad), Uo = 5 volt og Io = 150 mA (målt).

Ud fra formlen kan det ses at den største effekt, som vil blive afsat i seriegualtoren være omkring 1.1 watt, hvilket sagtens kan køles ned med en lille køleplade.

1.2 Regulator design

Regulatoren for iFollow robotten er designet ud fra teorier gennemgået i faget IRT. For at få et overblik over systemet, blev der opsat et blok diagram som beskriver motorkarakteristikken i systemet. På figur 3 ses motor blokdiagrammet som danner fundamentet for motor overføringsfunktionen og dermed også regulatoren.



Figur 3: Blokdiagram over iFollow system

Diagrammet på figur 3 er opbygget med to feedback loops der beskriver D faktoren også kendt som væskefriktionen og K_t faktoren som beskriver motoren maksimale omregningshastighed. Diagrammet baserer sig på at der først er et summeringspunkt, hvor inputtet (spændingen som bliver påtrykt motoren) og feedbacket bliver vurderet i forhold til hinanden for at finde ud af hvad der skal løbe videre til motorerne. Den påtrykte spænding kan laves om til et moment, ved først at gange et $\frac{1}{R}$ led på spændingen for at få strømmen i Ampere. Derefter ganges motorens torque-konstant på for at finde momentet til den givende strøm. Denne konstant findes i motorens datablad til[14]:

$$K_i = 2.23kgf.cm/A$$

Denne torque-konstant er indgang til endnu et summeringspunkt, hvor også væskefriktionen summeres fra et feedback. Dette giver et resulterende moment, da iFollow robotten flyttes lineært ved en rotation af hjulene. Udtrykket for det resulterende moment bygger på kraft*arm princippet:

$$\tau_{res} = F * r \tag{2}$$

Da det ønskes at få et udtryk for motoren hvor en påtrykt masse indgår, substitueres F i formel 2 med et udtryk der indeholder masse.

$$F = M * a = M * r * \alpha \tag{3}$$

På denne måde fås et udtryk for det resulterende moment for motoren:

$$\tau_{res} = M * r^2 * \alpha \tag{4}$$

På diagrammet på figur 3 ses det at der fås en α udgang efter leddet $\frac{1}{M*r^2}$ som beskriver massefriktionen for motoren. Før dette led kommer et $\frac{1}{J}$ led, som beskriver motorens interne friktion.

Denne antages at være så tilstrækkelig lille, at den ikke ændrer på det resulterende moment. Ved at integrere for α fås en vinkelfrekvens for motoren, som også beskriver en hastighed.
Ved nu at have en idé om hvad motorsystemet indeholder, og hvad hvilke integrationer og ændringer betyder for outputtet af motoren, er det tid til at bestemme motorens overføringsfunktion. Dette gøres ud fra flere værdier fra motorenes datablad[14] samt den generelle motorligning der er blevet anvendt i IRT undervisningen [18]. Den generelle overføringsfunktion der bliver anvendt for motoren ses her i formel 5

$$G(s) = \frac{\frac{K_t}{R_a * J}}{(s + \frac{1}{J} * (D_m + \frac{K_t * K_b}{R_a}))} \tag{5}$$

Ikke alle værdier fra motorenes datablad er korrekte, og det er derfor nødvendigt selv at udregne nogle af værdierne. Derfor udregnes motornes K_b faktor, ved først at finde antallet af rotationer pr. minut ved en kendt spænding og strøm over motoren. Værdierne herfor er:

$$E_a = 1.972V$$

$$I_a = 0.1632A$$

$$\omega_{Rot} = \frac{46}{\frac{60}{2 * pi}} = 4.817$$

Ved at isolere K_b i formel 6 og indsætte de målte værdier samt R_a fra datablade, kan K_b findes:

$$E_a = R_a * I_a + K_b * \omega_{Rot} \tag{6}$$

$$K_b = 0.3634$$

En anden ukendt parameter for motoren og formel 5 er D_m som kan udregnes ved isolation af D_m i formel 7:

$$D_m * NoLoadSpeed = K_t * NoLoadCurrent \tag{7}$$

Her er det vigtigt at huske at det antages at $K_b = K_t$ som det er beskrevet i IRT undervisningen.

$$D_m = 0.00026353$$

Den sidste ukendte parameter i formel 5 er J-faktoren. Denne kan beregnes ved anvendelse af tidskonstanten som blev fundet i motor analysen. J_m udregnes ved at isolere den i formel 8:

$$\frac{1}{\tau} = \frac{1}{J_m} * (D_m * \frac{K_t * K_b}{R_a}) \tag{8}$$

Derved fås en værdi for J-faktoren på:

$$J_m = 0.0009894$$

Indsættes de ukendte parametre i formel 5 fås der et udtryk for motor overføringsfunktionen:

$$G = \frac{292}{s + 107}$$

Denne overføringsfunktion gælder kun for en generel motor, og tager derfor ikke højde for den masse der bliver påtrykt motoren når iFollow robottens yderligere systemer bygges på robotten og dermed ændrer massen. Det er derfor nødvendigt at samle blokdiagrammet sammen til sin egen overføringsfunktion ved at anvende teorien herfra beskrevet i IRT-undervisningen[17]. Gøres dette fås udtrykket for overføringsfunktionen til formel 9

$$G = \frac{K_t}{M * R_a * r^2 * s * (\frac{D_m}{M*r^2*S} + 1) * (\frac{K_b*K_t}{M*R_a*r^2*s*(\frac{D_m}{M*r^2*S}+1)})} \tag{9}$$

Indskrives den forventede masse af robotten samt radius på de monterede hjul, er det muligt at finde en specifik overføringsfunktion for iFollow robotten:

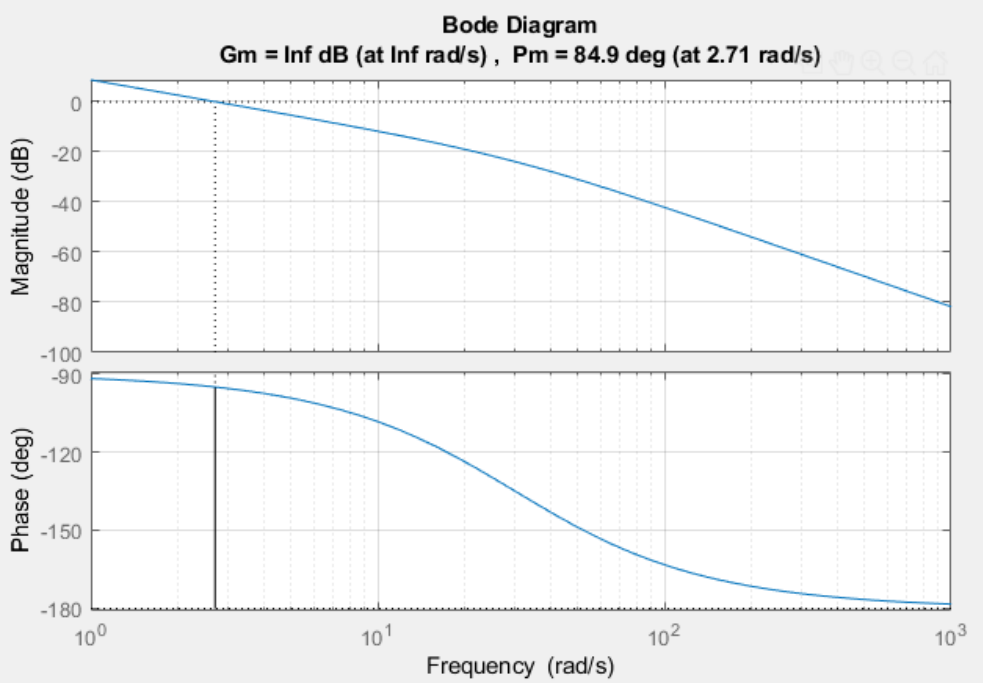
$$M = 5kg/2 = 2,5Kgpr.motor$$
$$r = \frac{0.075m}{2} = 0.0375m$$

På denne måde fås den endelige overføringsfunktion til formel 10

$$G = \frac{82.0499}{s + 30.1331} \tag{10}$$

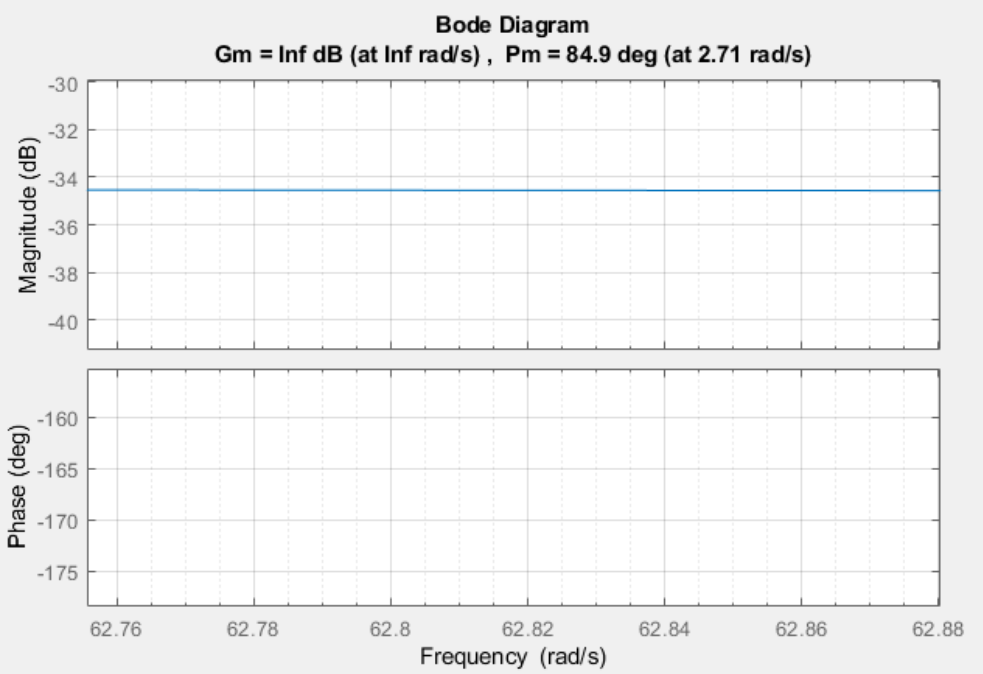
Denne overføringsfunktion danner grundlaget for regulatoren, som bygger et P og I led på, for at få den ønskede regulering. Disse led bliver implementeret i form af et Gain samt en lead-regulator. Regulatoren skal ende ud med en differensligning, for at det er muligt at implementere den digitalt i robotten.

Den fundne overføringsfunktion overføres til Matlab, hvor der først ses på bodeplottet for overføringsfunktionen.



Figur 4: Bodeplot for overføringsfunktion uden regulering

På figur 4 ses det hvordan fasemarginsfrekvensen ligger meget lavt på $2.71 rad/s$. For at forstærke denne ses der først på hvor høj det er nødvendigt at lave den. Da iFollow robotten skal følge en person, er det dennes acceleration der designes i forhold til. Det forventes at en person kan accelerere fra 0-ca.5 Km/t på 1 sekund. Dette svarer til en frekvens på 1Hz. For at systemet er hurtigt nok, designes dets fasemarginsfrekvens til 10 gange personens frekvens, dvs. $10Hz$. Til at gøre dette omregnes de 10 Hz til rad/s hvilket giver 62.83rad/s. For at finde ud af hvor højt et gain der skal tilføjes for at få denne fasemarginsfrekvens, kigges der på magnituden ved denne frekvens.

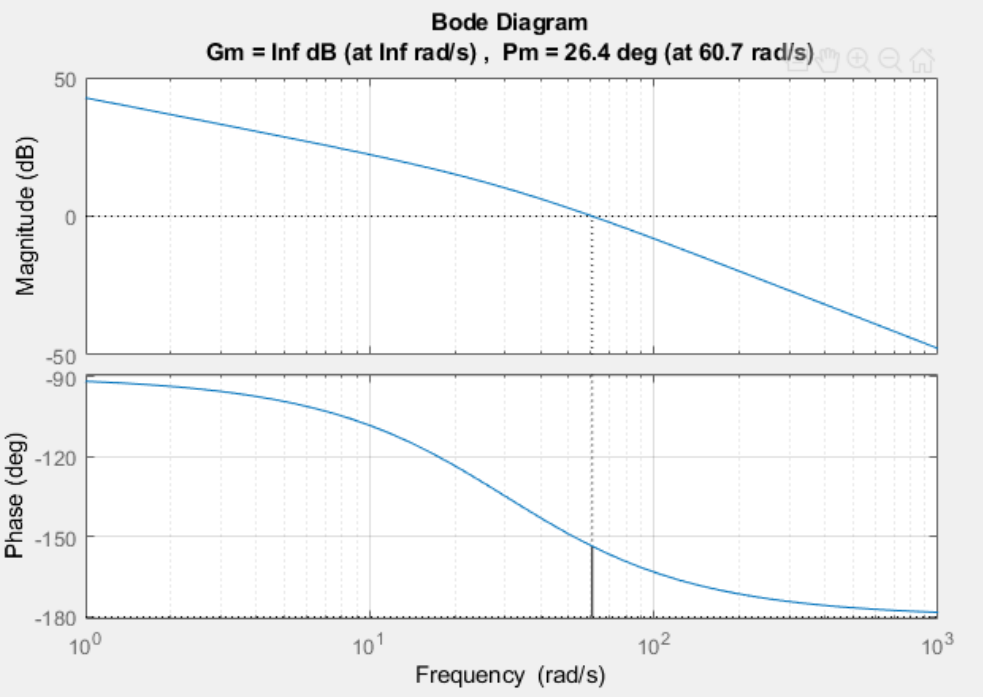


Figur 5: bodeplot for fasemaginfrekvens 62.83rad/s

På figur 5 ses det at der skal laves en forstærkning på 34 dB, hvilket kan omregnes til en gain værdi ved anvendelse af online værktøjet [19].

$$34dB = 50.1187Gain$$

Dette gain ganges på overføringsfunktionen, hvilket giver et bodeplot med den ønskede fasemargin-frekvens:



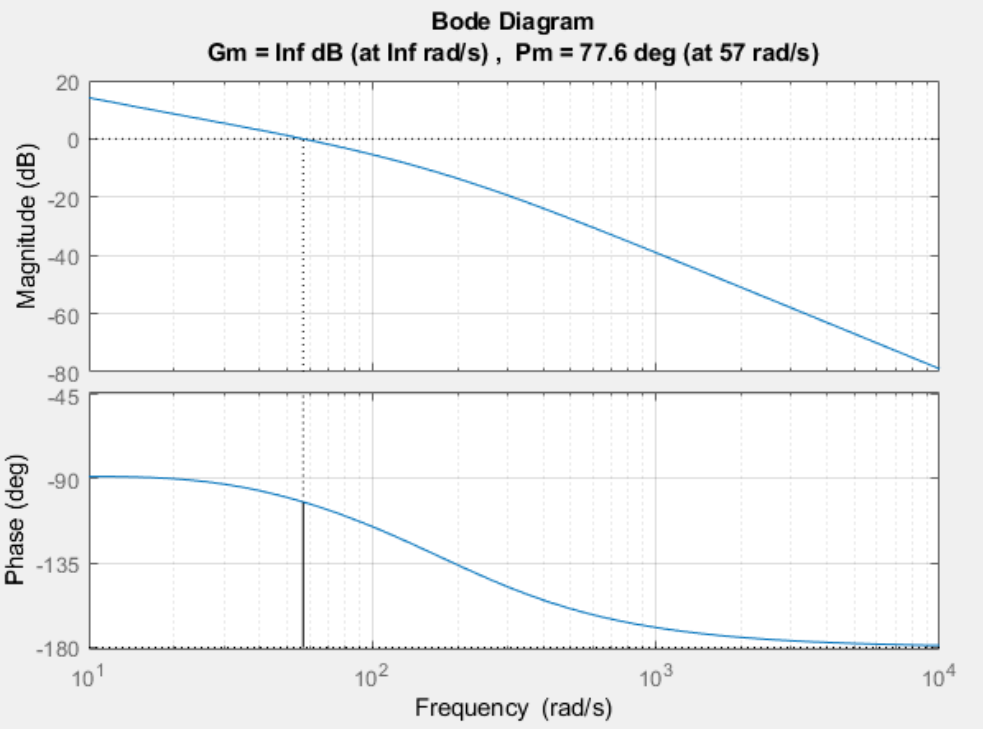
Figur 6: Bodeplot for overføringsfunktion med P led.

Det ses på figur 6 at den ønskede fasemarginfrekvens er opnået, men at fasemarginen er alt for lav. For systemet vil det være tilstrækkeligt med en fasemargin over 70 grader, hvilket kan opnås med en lead-regulator der hæver 50 grader. I listing 1 herunder ses hvordan leadregulatoren designes.

Listing 1: Lead regulator design Matlab

```
%Design lead - pm skal haeves 50 deg;
%50 deg = 0,872664626 rad!!

rad = 0.872664626;
w0 = 66;
Beta = (1-sin(rad))/(1+sin(rad));
T=1/(w0*sqrt(Beta));
kc = sqrt(Beta);
GLead = ((1/Beta)*(s+(1/T)))/(s+(1/(Beta*T)))*kc;
```

Implementeres denne lead-regulator fås bodeplottet som kan ses på figur 7 herunder:



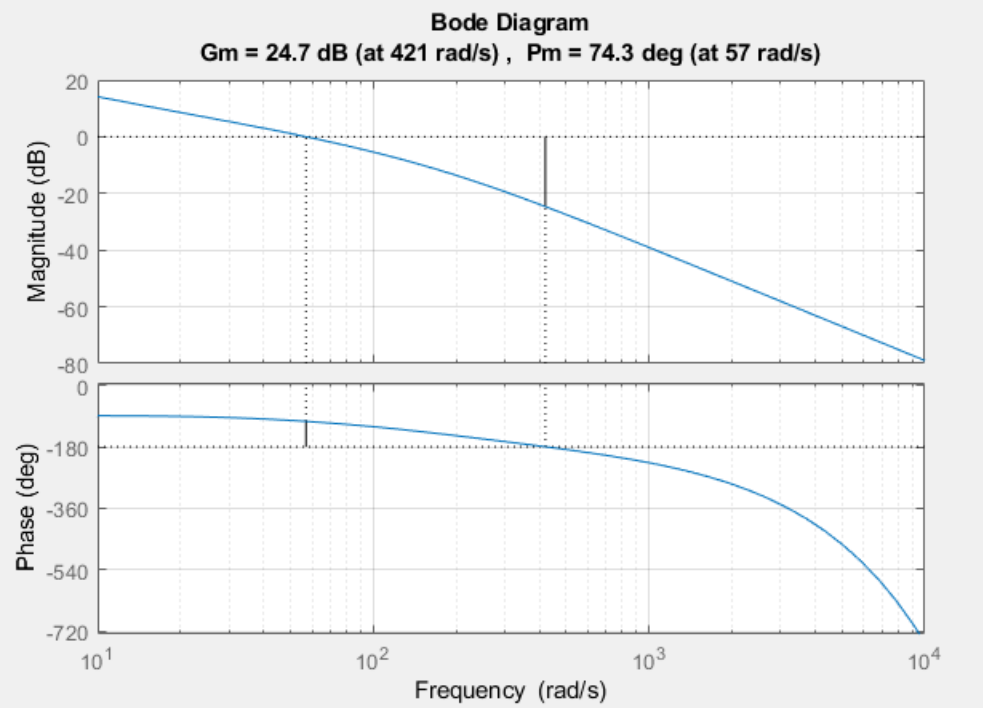
Figur 7: Bodeplot for reguleret overføringsfunktion

det ses her at fasemarginen kommer op på mere end 77 grader, hvilket er meget passende for systemet. Dog ses det også at fasemarginfrekvensen falder en smule, hvilket dog ikke har den største betydning og dermed er ok. Da regulatoren skal imeplementeres digitalt på en PSoC, er dette ikke det endelige bodeplot for regulatoren, da der vil være en indflydelse fra PSoCens Sample-Hold kredsløb. Denne indflydelse findes derfor for at finde den endelige overføringsfunktion.

Listing 2: Sample Hold inflydelse

```
% fasemargins tab grundet Sample hold.
Ts = 0.002;
Gsh = exp(-s*Ts/2);

margin(G1_Kp_GLead*Gsh)
grid on;

% Ud fra denne kan vi se at vi mister 4 grader fasemargin, hvilket vi stadig er okay
    med, da vi ligger over 70 grader. Dette er fint for vores system.
```

Listing 2 beskriver indflydelsen af Sample Hold kredsløbet, og på figur 8 ses det endelige bodeplot for overføringsfunktionen:



Figur 8: Bodeplot for regulator

Da regulatoren implementeres digitalt er det nødvendigt at finde en differensligning for regulatoren, da det er denne der skal anvendes til implementeringen. For at komme frem til denne, laves der først en transformering af regulatorens overføringsfunktion, for at få den omdannet til det diskræte domæne. Dette gøres ved at anvende matlabs $c2d$ funktion, og definere at den skal transformere ved bilineær transformation, som er transformationen der er blevet anvendt i IRT-undervisningen.

Listing 3: Bilineær transformering

```
% z-transformering

Gc_z = c2d((Kp*GLead),Ts,'tustin')

% Hermed faar vi vores funktion til digital regulering.
```

Ved at anvende listing 3 til transformering, fås overføringsfunktionen i frekvensdomænet, som anvendes til bestemmelse af differensligningen til:

$$G(z) = \frac{119.1z - 113.5}{z - 0.693} \tag{11}$$

Formel 11 ganges på hver led med z^-1 og sættes om på anden form, hvor Y(z) og X(z) indgår.

$$Y(z) = \frac{119.1 - 113.5z^-1}{1 - 0.693z^-1} * X(z) \tag{12}$$

I formel 12 ganges brøken ud, så alt kommer til at stå på samme brøkstreg

$$Y(z) * (1 - 0.693z^-1) = (119.1 - 113.5z^-1) \tag{13}$$

Herefter ganges der ind i parenteserne for at ophæve dem, hvorefter z^-1 udskiftes med tidsskift.

$$y(z) - y(z)0.693z^-1 = 119.1x(z) - 113.5x(z) \tag{14}$$

$$y(k) - y(k-1)0.693 = 119.1x(k) - 113.5x(k-1) \tag{15}$$

Ved nu at isolere for $y(k)$ i formel 15 findes den differensligning som skal implementeres for regulatoren i dennes software.

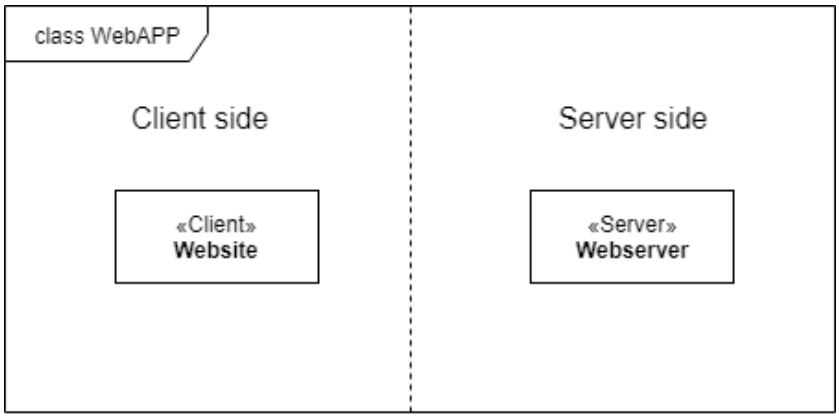
$$y(k) = 119.1x(k) - 113.5x(k-1) + 0.693y(k-1) \tag{16}$$

2 Software design

I dette afsnit designes software for de forskellige applikationer, som der blev opstillet applikationsmodeller for i software arkitekturen. Her forklares for de forskellige applikationer modulbeskrivelser, generelle implementeringer, funktions- og attributbeskrivelser mm. Flere steder vil disse dele vil også være illustreret med diagrammer og billeder. For indblik i den specifikke kode til disse applikationer henvises til *Software*-bilag[13].

2.1 WebAPP

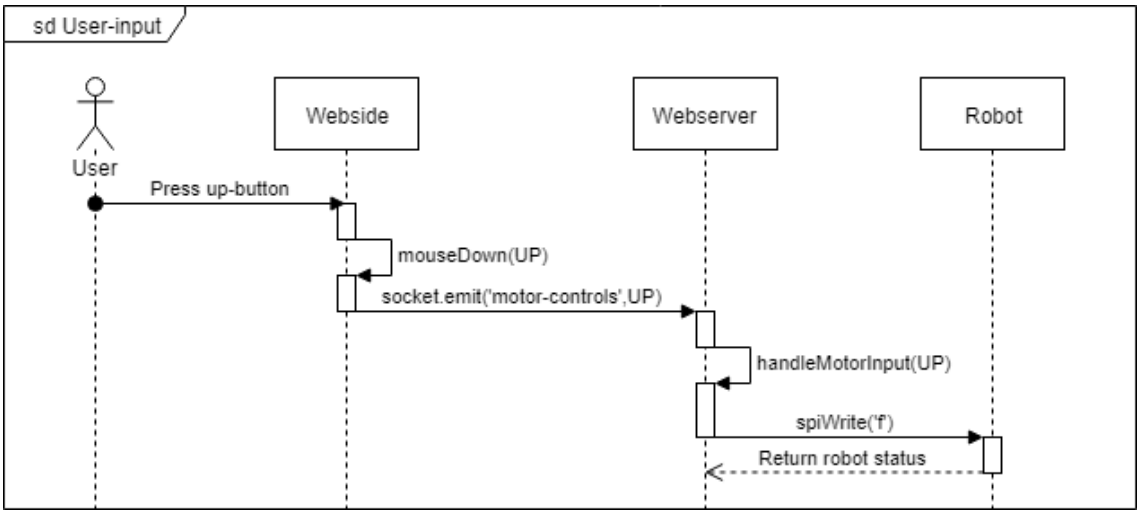
I denne del skal webapplikationens funktionalitet designes. Selve udseendet til webapplikationen dækkes i næste afsnit (GUI). I software arkitekturen blev her opstillet et samlet klassediagram til applikationsmodellen for WebAPP. Her skal primært opsættes design for de funktioner, der i arkitekturen blev indført i WebController-klassen. Designmæssigt kan disse funktioner opdeles i en client side og en server side fil. Den fil, som står for client side funktionerne kaldes "index.html". Filen som varetager server side funktionaliteten kaldes "webserver.js".



Figur 9: Klassediagram for opdeling af server og client side funktioner

I klassediagrammet på Figur 9 ovenfor ses WebController-klassen opdelt i disse to filer. Disse to filer vil hver især blive beskrevet i et afsnit for sig selv, som kan ses i de følgende underafsnit. De to sider kommunikerer med hinanden gennem et websocket, med dette websocket er det muligt at emitte events fra klient til server og vice versa.

Nedenfor ses et sekvensdiagram over en situation hvor klient-siden emitter et event til server-siden, for at sende motor-kommandoer gennem RPi'ens SPI-port til robotten.



Figur 10: Sekvensdiagram til illustration af kommunikation fra webside til webserver

Funktionerne der kan ses kaldet på ovenstående sekvensdiagram, kan findes beskrevet i afsnittene nedenfor for henholdsvis *Webserver* og *Webside*.

2.1.1 Webserver - Server-side

Serversiden repræsenterer webserveren, som bruges til håndtering af klient-forespørgsler og websidens dynamiske egenskaber.

2.1.1.1 Modulbeskrivelse

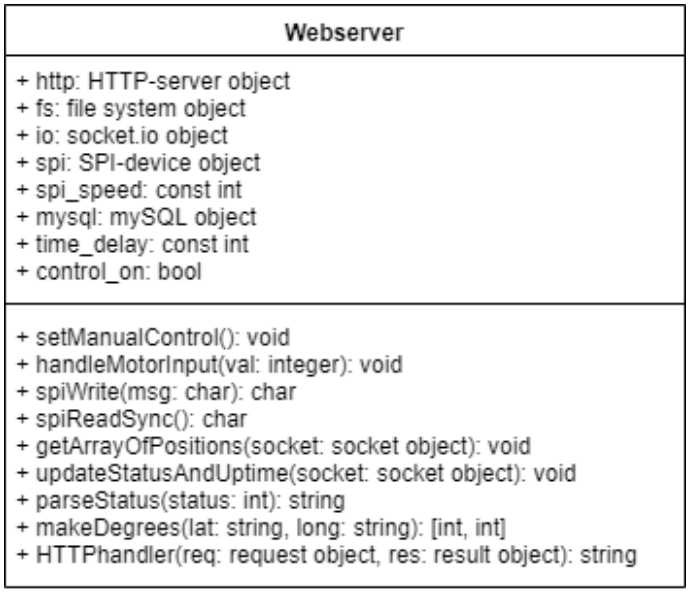
Modulet har ansvar for håndtering af forespørgsler fra klienter, herunder klient inputs, som tryk på knapper m.m. Derudover sørger webserveren også for at svare på HTTP-requests, ved at sende index-html siden til klienten.

2.1.1.2 Generel implementering

Webserveren implementeres på en Raspberry Pi Zero-W, med styresystemet: Rasbian. Selve programmet skrives i *Node.js* som er en JavasSript runtimeengine[15]. Denne runtime er oplagt til implementering af en webserver, da den er asynkron i sin natur, og derfor vil være god til håndtering af mange tilsluttede brugere samtidig. Derudover har node.js også mange relevante pakker til håndtering af bl.a. SQL-forspørgsler[9], SPI-kommunikation[12], filhåndtering[8], websockets[11] og HTTP-protokollen[10]. For at skabe et bindeled mellem klient og server bruges node.js-pakken *socket.io*, som bruges til at emitte events fra og til klienten. Derudover understøtter node.js også

cross-platforming, således at webapplikationen kan tilgås på flere forskellige enheder.

Nedenfor ses klassediagram af de anvendte funktioner i webserveren.



Figur 11: Klassediagram over webserverens funktioner

I næste afsnit beskrives funktionerne, som ses på Figur 11.

2.1.1.3 Funktionsbeskrivelser

Dette afsnit vil beskrive JavaScript funktionerne i serversiden af WebAPP-modulet med beskrivelser af funktionernes parametre, returværdier og formål. Disse vil være opdelt i primære funktioner og hjælpefunktioner.

Primære funktioner

- **HTTPHandler()**
  Parametre: Request-objekt der indeholder information om klientens request og response-objekt der indeholder serverens respons.
  Returværdi: Formodentligt repsonse-objektet.
  Beskrivelse: Bruges til håndtering af HTTP-forespørgsler fra klienter.

- **setManualControl()**
  Parametre: Ingen (*void*).

Returværdi: Ingen (*void*).
Beskrivelse: Bruges til efterspørgsel om adgang til manuel kontrollering af robotten.

- **HandleMotorInput(val)**
  Parametre: Værdi tilsvarende til en retning, hvor 1 er fremad, 2 er bagud, 3 er venstre og 4 er højre.
  Returværdi: Ingen (*void*).
  Beskrivelse: Bruges til at oversætte bruger-inputs til retnings-efterspørgselser til robotten.

- **getArrayOfPositions(socket)**
  Parametre: Socket-objekt til emitting af events til klienten.
  Returværdi: Ingen (*void*).
  Beskrivelse: Bruges til at hente og emitte GPS-data fra SQL-server.

- **updateStatusAndUptime(socket)**
  Parametre: Socket objekt til emitting af events til klienten.
  Returværdi: Ingen (*void*).
  Beskrivelse: Bruges til at hente og emitte status fra robot og uptime for lokal-enhed.

- **emit(event-name, parameters)**
  Parametre: Brugerdefineret event-navn og eventuelle parametre.
  Returværdi: Ingen (*void*).
  Beskrivelse: Bruges til emitting af brugerdefineret events fra server til klient, hvis klienten har hooket dette event kaldes callback-funktionen hos klienten.

Hjælpe-funktioner

- **makeDegrees(lat,long)**
  Parametre: Latitude og longitude med minutter.
  Returværdi: Oversat latitude og longitude i grader.
  Beskrivelse: Bruges til at oversætte latitude og longitude til grader, således at det er kompatibelt med Bing Maps API.

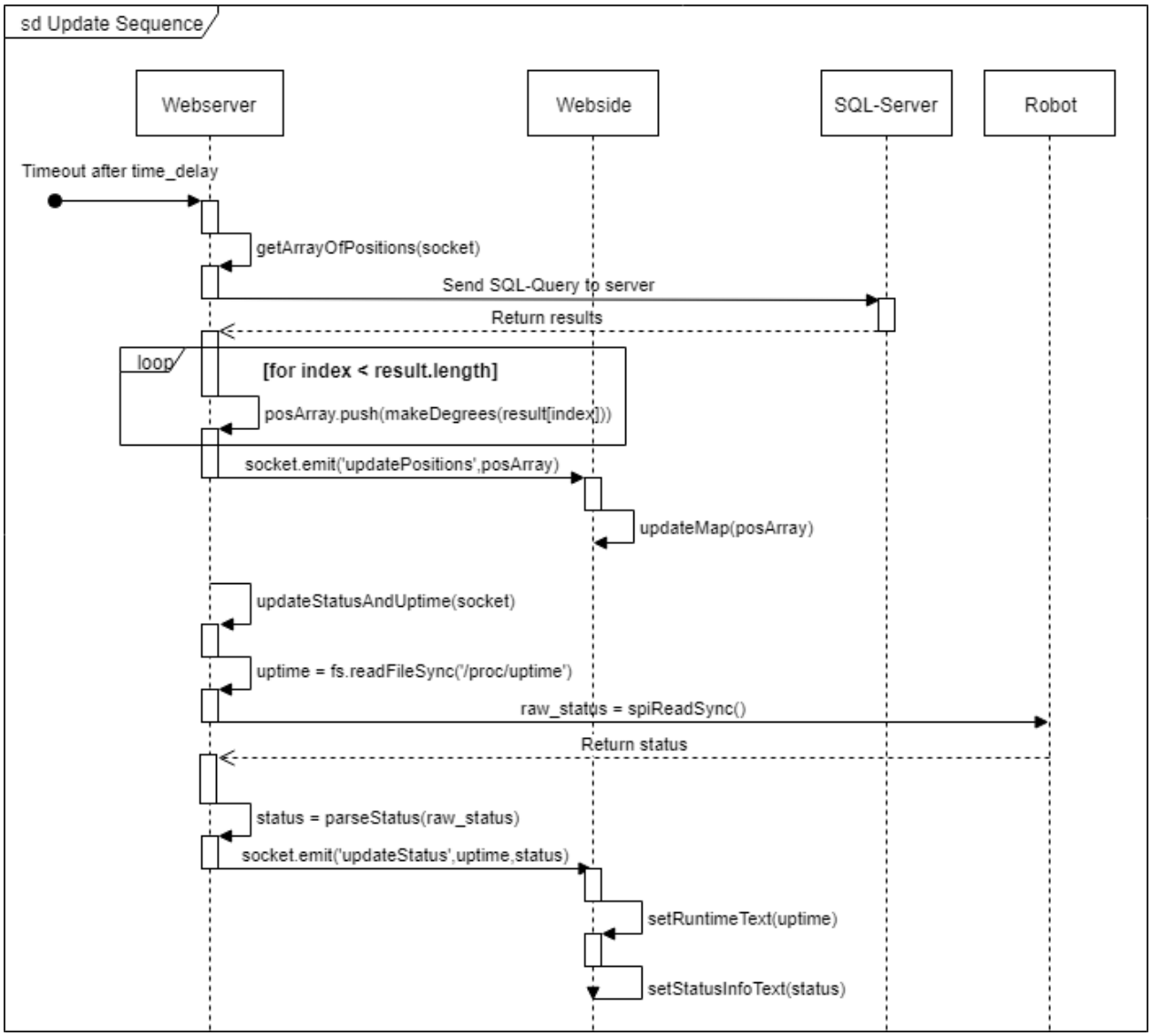
- **parseStatus(status)**
  Parametre: Robottens status i en integer-værdi.
  Returværdi: Oversat status-streng.
  Beskrivelse: Bruges til at oversætte status fra robot til læselig streng.

- **spiReadSync()**
  Parametre: Ingen (*void*).
  Returværdi: Læst karakter fra SPI0-bussen.
  Beskrivelse: Bruges til synkron læsning af en enkel karakter på SPI0-bussen.

- **spiWrite(msg)**
  Parametre: Ønsket karakter til afsendelse på SPI0-bus.
  Returværdi: Læst karakter fra SPI0-bussen.
  Beskrivelse: Bruges til asynkron skrivning af en enkel karakter på SPI0-bussen.

Udover funktionerne beskrevet i de to lister ovenfor, er der også implementeret funktioner til styring af LED'er på RPi'en og til kommunikation på UART-bussen. Disse funktioner blev dog ikke benyttet i det endelige design, og kan derfor ikke findes i de ovenstående beskrivelser.

For at illustrere hvordan positionerne på websidens map og status på websidens status-panel opdateres, ses nedenfor et sekvensdiagram med både webside, webserver, robot og sql-server.



Figur 12: Sekvensdiagram over opdateringssekvens

På det ovenstående diagram ses hvordan opdateringssekvensen forløber mellem klient, webserver, SQL-server og robotten. Denne sekvens vil blive kørt efter en fast tidsforsinkelse på *time_delay*, denne tidsforsinkelse kan dog ikke helt garanteres, da kaldet er asynkront og derfor må vente på højere prioriteret funkionskald på call-stacken.

2.1.1.4 Attributbeskrivelser

I dette afsnit vil de forskellige attributter for Webserveren blive beskrevet. Alle variablerne/objekterne er implementeret som globale variabler.

- **http**
  Formål: Handler til serveren, bruges bl.a. til hooking af klient-tilslutninger.
- **fs**
  Formål: Filsystem-handler, bruges til generel filhåndtering og til læsning af *proc/uptime/*.
- **io**
  Formål: Socket.io handler til håndtering af websocket mellem klient og server, bruges bl.a. til emitting af events
- **spi**
  Formål: spi-device handler til håndtering af SPI-kommunikation
- **spi_speed**
  Formål: En konstant clock-hastighed til SPI0-bussen
- **mysql**
  Formål: mysql handler til håndtering af kommunikationen med SQL-serveren.
- **time_delay**
  Formål: En konstant tidsforsinkelse til hvor ofte positioner, status og uptime skal opdateres på websiden
- **control_on**
  Formål: Boolean til at holde styr på om robotten er i manuel-kontrol tilstand.

Udover de ovenstående attributter kan der igen findes ekstra variabler i source-koden, som ikke benyttes i det endelige design.

2.1.2 Website - Klient-side

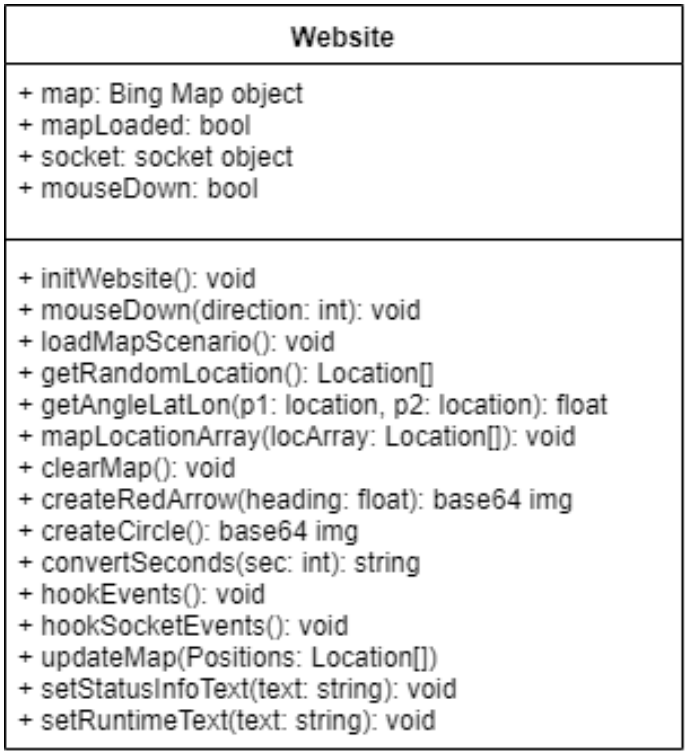
Klientsiden repræsenterer websiden som brugeren har adgang til. Det er her brugeren får præsenteret et kort over robottens positioner, robottens status og robottens levetid. Derudover er det også på websiden at robotten skal kunne styres manuelt.

2.1.2.1 Modulbeskrivelse

Modulet lever kun i klientens side, da klienten får tilsendt denne webside over TCP/UDP med HTTP-protokollen. Modulet har altså ansvaret for at få præsenteret brugergrænsefladen på en ordentlig og brugervenlig manér.

2.1.2.2 Generel implementering

Klientsiden vil indeholde en *index.html* fil, bestående af kode i sprogene: HyperText Markup Language (*HTML*)[4], Cascading Style Sheets (*CSS*)[3] og JavaScript[5]. Af disse sprog vil HTML blive brugt til at definere indhold på websiden, CSS til ordentlig præsentation af indhold og JavaScript til implementering af funktionaliteten. For at gøre websiden interaktiv, oprettes et socket til webserveren gennem socket.io (*node.js bibliotek*), dette socket bruges til udsending af events frem og tilbage mellem klient og server. Derudover bruges Javascript-pakken JQuery til animering af websiden. Til kortlægning af robottens ruter, bruges Bing Maps API. [16]



Figur 13: Klassediagram over websidens funktioner

Grundet JavaScripts asynkrone natur implementeres det meste af funktionaliteten med eventstyret-programmering. Derfor vil designet også afvige fra arktitekturen i *Systemarkitektur*-dokumentet. Figur 13 viser de anvendte funktioner i website-klassen. Disse funktioner bruges f.eks. som callback-funktioner til diverse events. Nedenfor ses beskrivelser for de ovenstående funktioner.

2.1.2.3 Funktionsbeskrivelser

Dette afsnit vil beskrive JavaScript funktionerne i klientsiden af WebAPP-modulet med beskrivelser af funktionernes parametre, returværdier og formål.

Primære funktioner

- **initWebsite()**
  Parametre: Ingen (*void*).
  Returværdi: Ingen (*void*).
  Beskrivelse: Bruges til at initialisere websiden med hooking af events.

- **mouseDown(direction)**
  Parametre: Den ønskede retning motoren skal køre i en integer værdi, hvor 1 svarer til fremad, 2 svarer til bagud, 3 svarer til venstre og 4 svarer til højre.
  Returværdi: Ingen (*void*).
  Beskrivelse: Bruges til at kalde de rigtige kommandoer til motoren efter ønsket retning.

- **loadMapScenario()**
  Parametre: Ingen (*void*).
  Returværdi: Ingen (*void*).
  Beskrivelse: Callbackfunktion som kaldes, når kortet indlæses på hjemmesiden..

- **updateMap(positions)**
  Parametre: Et array af positioner/lokationer.
  Returværdi: Ingen (*void*).
  Beskrivelse: Sørger for at kortet bliver opdateret med nye positioner/lokationer.

- **emit(event-name, parameters)**
  Parametre: Brugerdefineret event-navn og eventuelle parametre.
  Returværdi: Ingen (*void*).
  Beskrivelse: Bruges til emitting af brugerdefineret events fra klient til server, hvis serveren har hooket dette event kaldes callback-funktionen hos serveren.

Hjælpe-funktioner

- **getAngleLatLon(p1,p2)**
  Parametre: To lokations/positions-objekter.
  Returværdi: Vinkel i grader
  Beskrivelse: Beregner vinklen mellem to lokationer - bruges til at rotere en "pushpin" korrekt

- **mapPositionArray(locArray)**
  Parametre: Et array af location-objekter.
  Returværdi: Ingen (*void*).
  Beskrivelse: Opdaterer kortet med lokationer fra locArray, indtegner "polylines" mellem disse og sørger for, at den sidste lokation/position vises som en rød pil og resten som blå cirkler.

- **clearMap()**
  Parametre: Ingen (*void*).
  Returværdi: Ingen (*void*).
  Beskrivelse: Sletter alle pushpins og polylines fra kortet.

- **createRedArrow(heading)**[2]
  Parametre: En retning i grader.
  Returværdi: Et canvas element som base64 image URL.
  Beskrivelse: Laver en rød pil som pushpin - bruges til visning af den sidst kendte lokation.

- **createCircle()**
  Parametre: Ingen (*void*).
  Returværdi: Et canvas element som base64 image URL.
  Beskrivelse: Laver en blå cirkel som pushpin - bruges til visning af de de forrige lokationer.

- **convertSeconds(sec)**[1]
  Parametre: Tid i sekunder.
  Returværdi: En streng med tiden i formatet HH-MM-SS.
  Beskrivelse: Tager antal sekunder for uptime fra RPi'en og formatterer dette. Bruges til visning af driftstid.

- **hookEvents()**
  Parametre: Ingen (*void*).
  Returværdi: Ingen (*void*).
  Beskrivelse: Sørger for at hooke input-events.

- **hookSocketEvents()**
  Parametre: Ingen (*void*).
  Returværdi: Ingen (*void*).
  Beskrivelse: Sørger for at hooke socket-events.

- **setStatusInfoText(text)**
  Parametre: Tekst til status-info i status-panelet.
  Returværdi: Ingen (*void*).
  Beskrivelse: Bruges til at sætte teksten til status-info i status-panelet.

- **setRuntimeText(text)**
  Parametre: Tekst til runtime-info i status-panelet.
  Returværdi: Ingen (*void*).
  Beskrivelse: Bruges til at sætte teksten til runtime-info i status-panelet.

Udover funktionerne ovenfor blev der til udviklingen af denne kode også lavet test-funktioner. Her blev eksempelvis lavet tilfældige lokationer, som de kort-relaterede funktioner blev testet på.

2.1.2.4 Attributbeskrivelser

I dette afsnit vil de forskellige attributter for Websiden blive beskrevet. Alle variablerne/objekterne er implementeret som globale variabler.

- **map**
  Formål: Bing map-objekt hvis funktioner tilgås med forskellige API-funktioner.

- **map_loaded**
  Formål: Boolean til at holde styr på, om kortet er indlæst.

- **socket**
  Formål: Socket.io objekt til socket-events, motor styring f.eks.

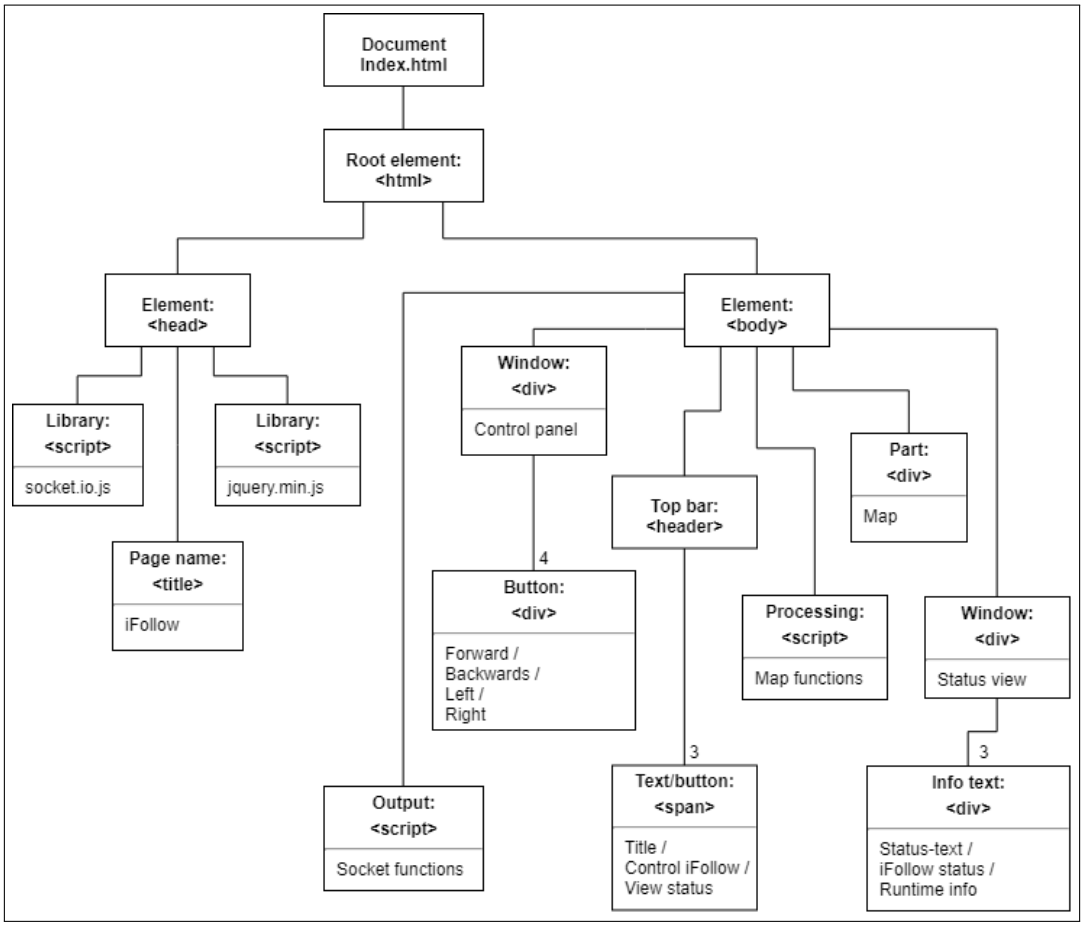
- **mousedown**
  Formål: Boolean til event-listeners, der skal holde styr på, om en knap bliver holdt inde.

2.2 GUI frontend

Webapplikationens GUI frontend designes med HTML og CSS. Disse bruges til henholdsvis at sætte rammer for indholdet og herefter modificere dets udseende samt tilføje basale animationer ved brugerinteraktion.

2.2.1 Struktur

Da strukturen i denne slags programmering er forskellig fra den slags kode, der normalt skrives arkitektur og design for, er ideen her at skabe et overblik over de grafiske elementer af webapplikationens GUI. Her er forsøgt at overskueliggøre, hvordan hjemmesidens indhold skal deles op, når det samlede HTML-dokument skal skrives.



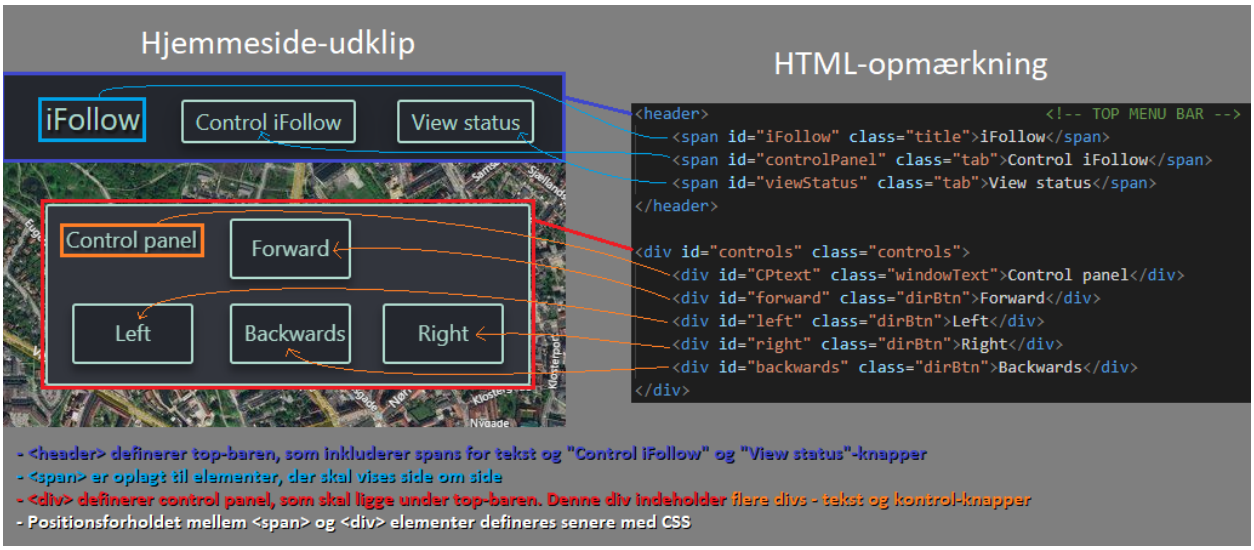
Figur 14: Oversigt over webapplikationens html dokument

På Figur 14 ovenfor ses et forsøg på at danne et overblik over indholdet af webapplikationens HTML-dokument. Dette er ikke et bestemt diagram, som hører til noget modelleringssprog, men er inspireret af de velkendte blokdiagrammer fra SysML. Opdelingen af de forskellige blokke skal hjælpe med at identificere, hvilke dele der hører til dokumentets "body" og "head", og hvordan nogle "div"-elementer indeholder flere "div"-elementer. Eksempelvis kan man ud fra blokken midt i diagrammet se, at denne er et "div"-element, som skal opføre sig som kontrolpanelets vindue. Dette kontrolpanel indeholder så 4 "div"-elementer, én for hver retningsknap.

Sammen med illustrationerne for webapplikationen, som blev opstillet i de funktionelle krav, kan denne oversigt også bruges til at bestemme, hvordan CSS-koden skal skrives.

2.2.2 HTML-Opmærkning

Ud fra diagrammet på Figur 14 skrives HTML-kode til opmærkning af hjemmesidens indhold. Nedenfor ses et udklip af hjemmesiden med den HTML-kode der definerer det pågældende indhold. Her er forsøgt at illustrere og forklare, hvilke bestemte dele af hjemmesiden, der afhænger af HTML-opmærkningen.



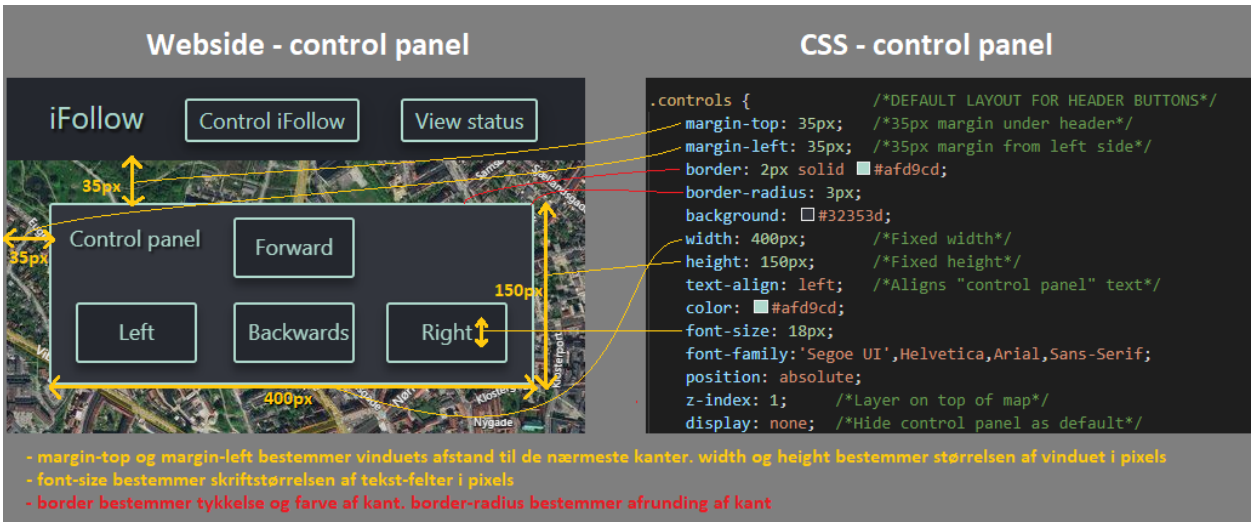
Figur 15: Eksempel på HTML-opmærkning af hjemmeside-indhold

På Figur 15 ovenfor ses en forklarende illustration af hjemmesidens indhold med den tilhørende HTML-kode. Her er udklippet af hjemmesiden taget fra den endeligt implementerede GUI frontend-kode. Udseendet af hjemmesiden er altså på dette tidspunkt et resultat af både HTML-opmærkning og CSS-styling. Dog holder den forklarende del af illustrationen kun fokus på HTML kontekksten. Her ses hvordan et delelement kan være en eller en , og hvordan der kan være flere -elementer inde i et enkelt -element. Desuden ses, hvordan de forskellige HTML-elementer tildeles et ID og en klasse. Brugen af disse forklares i næste afsnit.

2.2.3 CSS-Styling

På samme måde, som der i ovenstående afsnit blev forklaret, hvordan HTML er brugt til opmærkning, bliver her vist, hvordan CSS bruges til styling af hjemmesidens indhold. Nedenfor ses en

illustration af hjemmesidens kontrolpanel og den CSS-kode der er skrevet til dette, med sammenhængende forklaring af, hvordan de to dele hænger sammen.



Figur 16: Eksempel på CSS-styling af hjemmeside-indhold

Som det ses på Figur 16 ovenfor, fastlægger CSS alt fra de grundlæggende grafiske dimensioner i pixels til de helt specifikke egenskaber som afrunding af kanter, skrifttype osv. Det ses af kode-eksemplet, at dette beskriver stilen af klassen "controls". Dette er muligt, da der i HTML-opmærkningen blev lavet netop denne klasse til det -element, som skulle stå for kontrol-panelet. Med CSS kan man altså vælge delelementer af HTML-indholdet og modificere, hvordan visningen af dette skal være. Det smarte ved dette er eksempelvis, at hver kontrol-knap er af den samme klasse "dirBtn", så man undgår at skulle skrive CSS-kode til hver enkelt knap, ved blot at skrive CSS-kode for "dirBtn"-klassen. Hvis man dog ønsker, at tilgå et specifikt element, uden at modificere andre elementer af samme klasse, kan man tilgå det pågældende element med dets ID. Derudover tilbyder CSS også mulighed for basale animationer. Til hjemmesidens GUI frontend-kode er der eksempelvis gjort brug af CSS-selectoren "hover". Denne gør det muligt at bestemme, hvordan et HTML-element skal se ud, når musemarkøren befinder sig på det pågældende element.

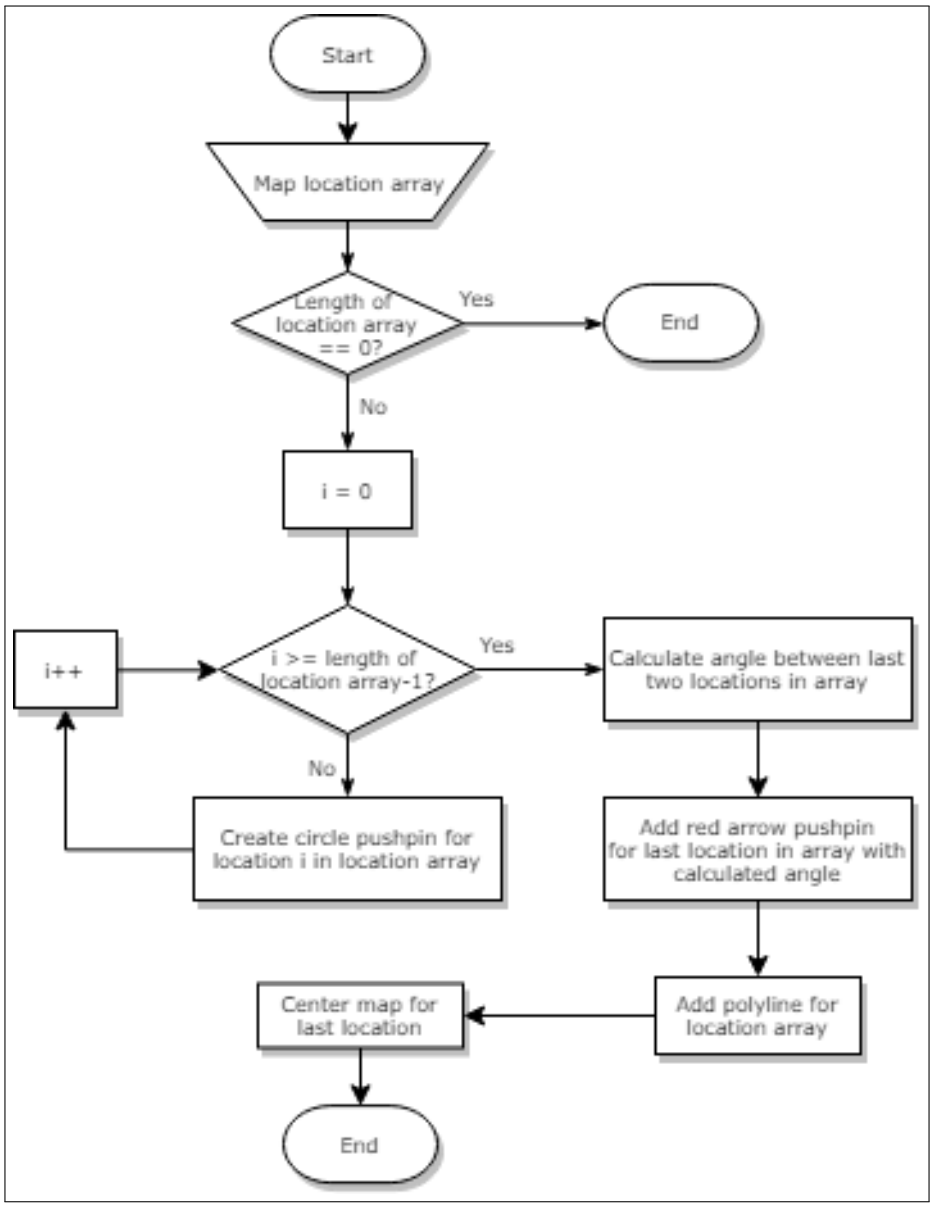
```
.dirBtn{                          /*DEFAULT LAYOUT DIRECTION BUTTONS (same as tab buttons)*/
    width: 100px;
    height: 50px;
    border: 2px solid #afd9cd;
    border-radius: 3px;
    text-align: center;
    background: #24272E;
    padding: 10px;
    transition: all 0.3s ease;
    box-shadow: 0 6px 6px 0 rgba(0,0,0,0.2), 0 6px 10px 0 rgba(0,0,0,0.19);

}

.dirBtn:hover {                   /*ON MOUSE HOVER LAYOUT DIRECTION BUTTONS (same as tab buttons)*/
    background: #3b3f4a;
    cursor: pointer;
    box-shadow: 0 6px 6px 0 rgba(0,0,0,0.69), 0 10px 25px 0 rgba(0,0,0,0.19);
}
```

Figur 17: Eksempel på hover-selector i CSS

På Figur 17 ovenfor ses et kode-eksempel, der gør brug af hover-selectoren med klassen "dirBtn". For denne klasse skrives først CSS-kode, der angiver, hvordan kontrol-knapperne skal se ud som udgangspunkt. Herefter kodes med hover-selectoren, hvordan kontrol-knapperne skal ændre udseende, når musemarkøren svæver over knappen; baggrunden for kontrolknapperne ændres til en mørkere farve, musemarkøren skal ændre type til "pointer", og fremtoningen af knappens skygge skal gøres tydeligere. Ved at fjerne musemarkøren fra denne knap igen, vil knappens udseende herefter skifte tilbage. Parameteren "transistion" i "dirBtn"-klassen sørger for, at skiftet mellem knappens udseende foregår glidende over 0,3 sekunder. Alt dette gør brugeroplevelsen ved interaktion med knapperne mere naturlig, da brugeren får et grafisk respons ved blot at holde markøren over en knap.

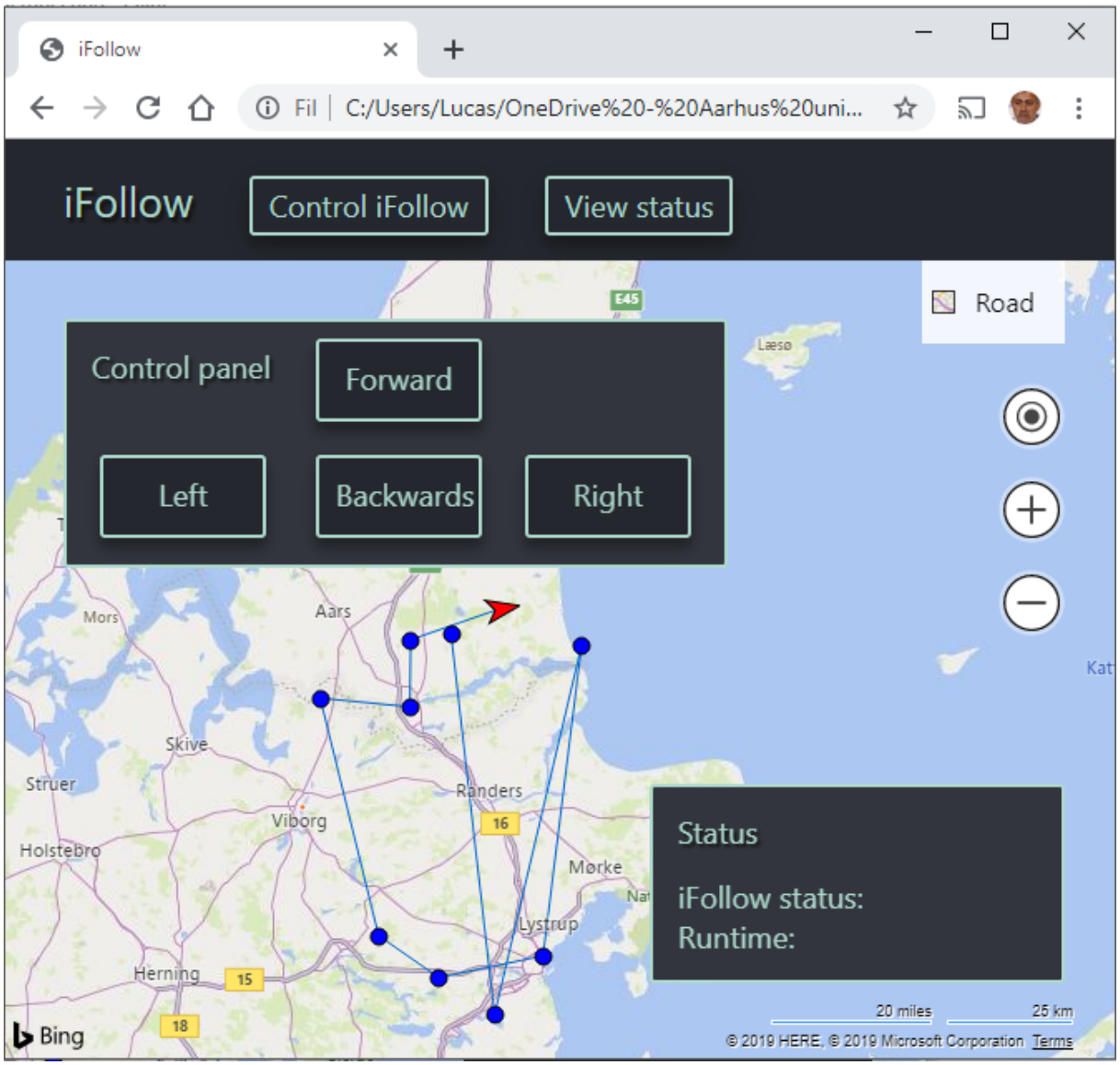
2.2.4 Modultest

Under udviklingen af GUI frontend-koden foretages løbende små tests for at sikre, at hjemmesiden opfører sig som ønsket. Her eftertjekkes om knapper, vinduer, kortvisning mm. ser ud som forventet. Desuden afprøves hjælpefunktionerne til hjemmesiden. Den væsentligste hjælpefunktion, som ønskes testet er mapLocationArray, hvori funktionerne getAngleLatLon, createCircle og createRedArrow indgår. Tilsammen skal funktionerne sørge for, at et array af location-objekter bliver kortlagt på en brugervenlig måde. Nedenfor ses et rutediagram, som illustrerer, hvordan denne kortlægning skal finde sted.



Figur 18: Rutediagram for mapLocationArray

Denne funktion skal nu testes. Her er anvendt en funktion, som genererer 10 tilfældigt placerede lokationer. Funktionen er i dette tilfælde sat til at blive kaldt, når der trykkes på "Forward"-knappen. Når der trykkes på "Backwards"-knappen, bliver kortet ryddet. Et screenshot af den testede hjemmeside ses på billedet nedenfor.



Figur 19: Modultest af webside GUI

Her på Figur 19 ovenfor ses et udklip af hjemmesidens GUI under test. Af den afbillede modultest ses, at knapper, tekst, vinduer og Bing Maps-visningen ser ud som ønsket. Desuden ses, at visningen af de forskellige lokationer ser rigtig ud; de første 9 lokationer vises med blå cirkler, den sidste lokation vises med en rød pil roteret i den korrekte retning, og der er indtegnet forbindende linjer mellem lokationerne. Dette gør bane for en god implementering med GPS-modulet senere hen. I det afbillede tilfælde er browser-vinduet desuden inskrænket, så billedvisningen er overskuelig. Det betyder altså, at der ved normal visning på en computer- eller telefonskærm ikke vil være så store vinduer for kontrolpanelet og status-vinduet, men disse vil her fylde mindre i forhold til resten

af vinduet.

2.3 Robot

Efter at have lavet applikationermodeller for PSoC'ens usecases var det blevet tid til at designe de klasser, som skulle bruges på PSoC'en. I sektionerne herunder vil hver klasse blive gennemgået.

2.4 I2C

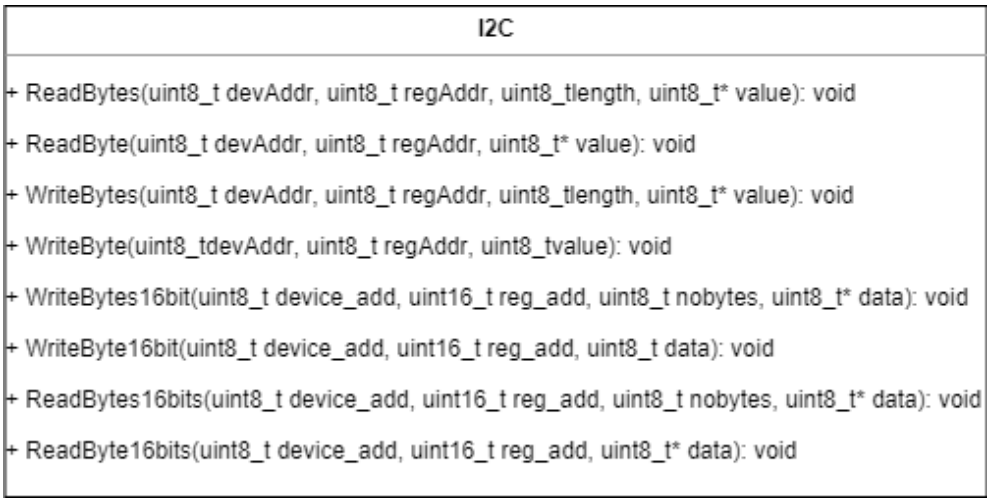
Flere hardware enheder under test kræver I2C-kommunikation. Der skal derfor designes et I2C-modul for at hæve abstraktionsniveauet for kommunikationen mellem disse enheder og PSoC, og for dermed at gøre håndtering af kommunikationen lettere. Dette I2C-modul vil hovedsageligt blive brugt som den kommunikative grænseflade mellem PSoC *(CY8KIT)* og den anvendte Gyrosensor.

2.4.1 Modulbeskrivelse

Som nævnt i afsnittet før, vil I2C-modulet hovedsageligt blive brugt til kommunikation med Gyrosensor fra PSoC. Derudover vil I2C-modulet også blive anvendt som en nem måde, at implementere eventuelle andre I2C-sensorer, dette vil været nyttigt, når man skal afprøve forskellige sensorer.

2.4.2 Generel implementering

Modulet implementeres som en C++ klasse i PSoC-Creator, da det er PSoC'ens kommunikation med I2C-enheder, modulet skal understøtte. Klassen vil indeholde funktioner til afsending og modtagelse af bytes gennem I2C protokollen. Derudover benytter funktionerne sig også af PSoC's indbyggede I2C-funktionaliteter. Funktionerne kan ses i det nedenstående klassediagram.



Figur 20: Klassediagram over I2C-Klassen

Funktioner på det ovenstående klassediagram beskrives i næste afsnit.

2.4.3 Funktionbeskrivelser

Dette afsnit vil beskrive funktionerne i I2C-klassen, med beskrivelser af funktionernes parametre, returværdier og formål.

- **ReadBytes(*uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t *value*)**
  Parametre: Addressen på I2C-enheden, den specifikke registeraddresse, mængden af bytes der skal læses og en pointer til den valgte data-container.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at læse en eller flere bytes fra en I2C-enheds register.

- **ReadByte(*uint8_t devAddr, uint8_t regAddr, uint8_t *value*)**
  Parametre: Addressen på I2C-enheden, den specifikke registeraddresse og en pointer til den valgte data-container.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at læse én byte fra en I2C-enheds register.

- **WriteBytes(*uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t *value*)**
  Parametre: Addressen på I2C-enheden, den specifikke registeraddresse, mængden af bytes der skal skrives og en pointer til de bytes der skal skrives.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at skrive en eller flere bytes til en I2C-enheds register.

- **WriteByte(*uint8_t devAddr, uint8_t regAddr, uint8_t value*)**
  Parametre: Addressen på I2C-enheden, den specifikke registeraddresse og en pointer til den byte der skal skrives.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at skrive én byte til en I2C-enheds register.

- **WriteBytes16bit(*uint8_t device_add, uint16_t reg_add, uint8_t nobytes, uint8_t *data*)**
  Parametre: Addressen på I2C-enheden, den specifikke 16-bit registeraddresse, mængden af bytes der skal skrives og en pointer til de bytes der skal skrives..
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at skrive en eller flere bytes til 16-bit register addresser.

- **WriteByte16bit(*uint8_t device_add, uint16_t reg_add, uint8_t data*)**
  Parametre: Addressen på I2C-enheden, den specifikke 16-bit registeraddresse og en pointer til den byte der skal skrives.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at skrive én byte til 16-bit registeraddresser.

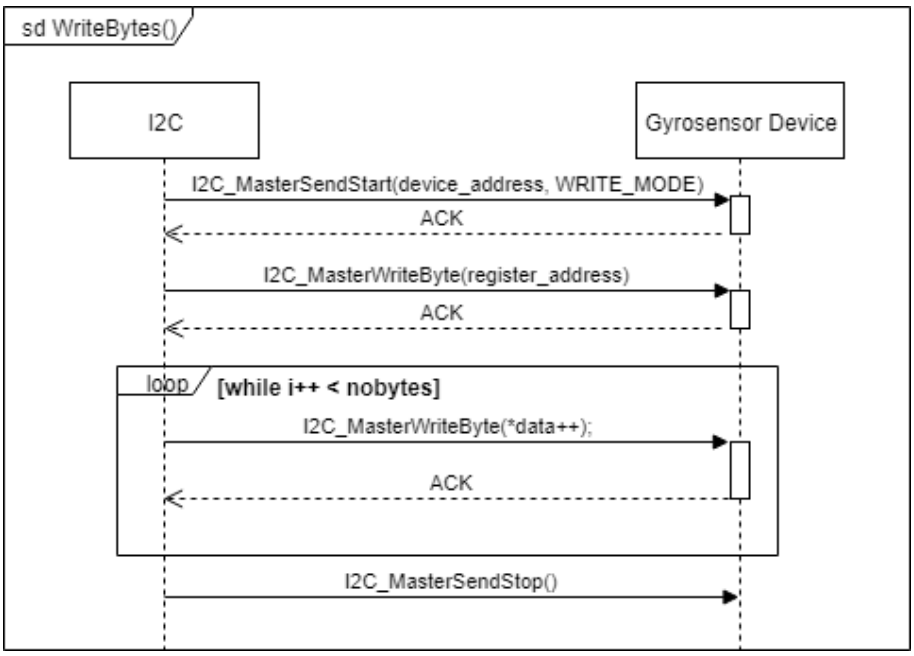
- **ReadBytes16bits(*uint8_t device_add, uint16_t reg_add, uint8_t nobytes, uint8_t *data*)**
  Parametre: Addressen på I2C-enheden, den specifikke 16-bit registeraddresse, mængden af bytes der skal læses og en pointer til den valgte data-container.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at læse en eller flere bytes fra 16-bit registeraddresser.

- **ReadByte16bits(*uint8_t device_add, uint16_t reg_add, uint8_t *data*)**
  Parametre: Addressen på I2C-enheden, den specifikke 16-bit registeraddresse og en pointer til den valgte data-container.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at læse én byte fra 16-bit registeraddresser

Fra de ovenstående funktioner, ses det at der også er blevet implementeret funktioner til håndtering af 16-bit register addresser. Dette skyldes at flere af de enheder der var under test brugte 16-bit addressering til nogle af deres registre.

Nedenfor ses sekvensdiagrammer for hvordan nogle af de vigtigste funktioner virker med eksempler på kommunikationsprotokollen med Gyrosensoren.

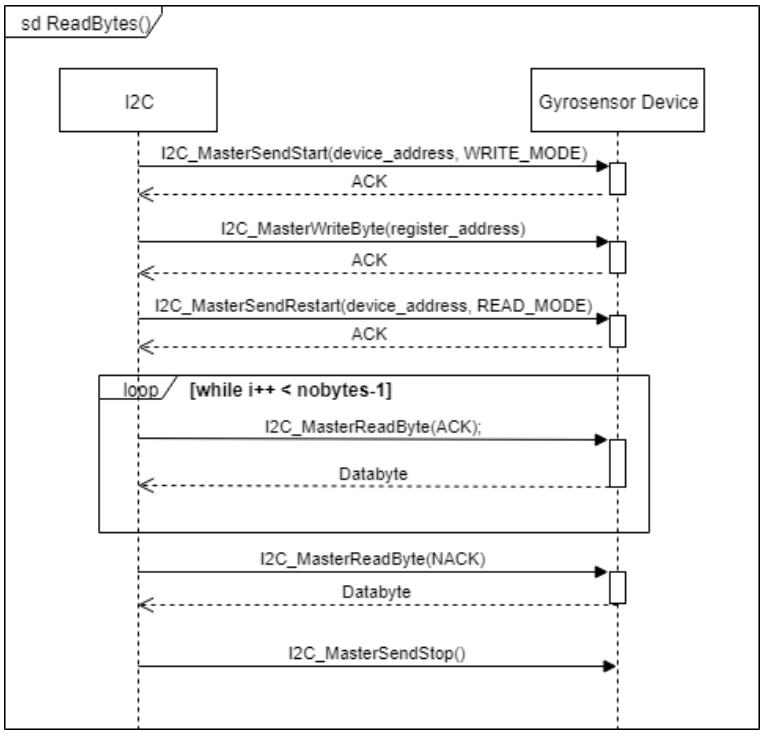
Sekvensdiagram - WriteBytes()



Figur 21: Sekvensdiagram af WriteBytes()

På ovenstående Figur 21 ses et sekvensdiagram, der illustrerer hvordan *WriteByte()*-funktionen, internt kalder PSoC's indbyggede funktioner.
Det er altså for at undgå at skulle kalde alle disse funktioner i den korrekte rækkefølge hver gang der skal sendes data, at I2C-klassen har implementeret en WriteByte()-funktion.

Sekvensdiagram - ReadBytes()



Figur 22: Sekvensdiagram af ReadByte()

På ovenstående Figur 22, ses hvordan bytes læses fra en I2C-enheds register, internt i funktionen *ReadBytes()*.

En simplificeret beskrivelse af PSoC's funktioner ses nedenfor.

- **I2C_MasterSendStart(device_address,READ/WRITE_MODE)**
  Parametre: Addressen på I2C-enheden og Read/Write tilladelse.
  Returværdi: Ingen (*void*).
  Beskrivelse: Sender start-condition og device addresse med Read/Write bit til slave.

- **I2C_MasterSendRestart(device_address,READ/WRITE_MODE)**
  Parametre: Addressen på I2C-enheden og Read/Write tilladelse.
  Returværdi: Ingen (*void*).
  Beskrivelse: Sender (re)start-condition og device addresse med Read/Write bit til slave.

- **I2C_MasterWriteByte(data)**
  Parametre: Byte af data til afsendelse gennem I2C.
  Returværdi: Ingen (*void*).
  Beskrivelse: Sender byte af data til slave.

- **I2C_MasterReadByte(ACK/NACK)**
  Parametre: ACK- eller NACK-bit, ACK hvis der ønskes at læse flere bytes og NACK hvis aflæsning ønskes afbrudt.
  Returværdi: Den læste byte. (*void*)
  Beskrivelse: Læser byte af data fra slave.

- **I2C_MasterSendStop()**
  Parametre: Ingen (*void*).
  Returværdi: Ingen (*void*).
  Beskrivelse: Sender stop-condition til slave.

2.5 Distancesensor

Det er i analysen blevet fastslået, at afstandsensoren skal bestå af tre VL53L0X ToF sensorer, samt en Arduino. Arduinoen er nødvendig, da producenten af sensoren ikke vil offentligøre en registeroversigt. Derfor er det ikke muligt hverken at initiere eller læse fra sensoren uden brug af producentens API.

2.5.1 Modulbeskrivelse

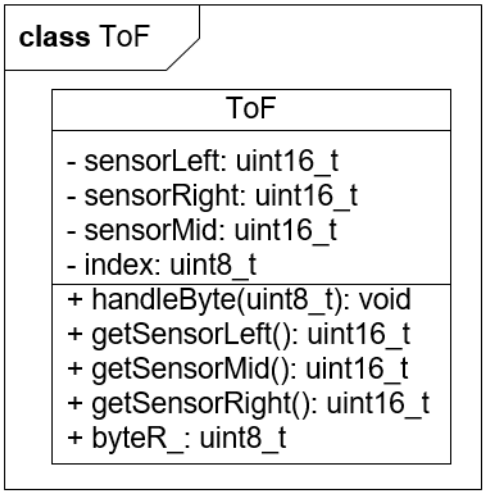
Design af dette modul kan deles op i to. Den første er Arduino delen. Denne har til formål at initiere, aflæse og videresende data fra sensorene. Den anden er PSoC'en der skal modtage og fortolke data fra Arduinoen.

2.5.2 Generel implementering

Arduino delen tager udgangspunkt i et standardprojekt producenten har lavet. I dette standard projekt initieres to sensorer hvorefter de aflæses og udskrives på UART'en. Grundet programmet størrelse anvendes en Arduino Mega. Dette projekt skal skrives om, så det er tre sensorer der initieres og aflæses. Derudover skal dataen sendes vha. SPI til PSoC'en og ikke udskrives på UART'en. PSoC'en skal modtage og håndtere dataen. Denne håndtering ligger i, at sensorene sender 16-bit værdier til Arduinoen. PSoC'ens SPI komponent, kan kun modtage 8-bit ad gangen. Derfor skal dataen deles op i to 8-bit værdier, for derefter at blive samlet korrekt på den anden side af SPI-bussen.

Dette opfyldes med en handler funktion på PSoC siden. Denne er lavet til en klasse sammen med de nødvendige attributter. Klassen kan ses på Figur 23. Funktionen kaldes fra PSoC'ens interrupt

handler.



Figur 23: Klassediagram over ToF-klassen

2.5.3 Funktionsbeskrivelser

Her vil PSoC funktionen fra klassen herover, samt de løse Arduino funktioner blive beskrevet.

PSoC

- **handleByte()**
  Parametre: uint8_t
  Returværdi: Ingen (*void*).
  Beskrivelse: Samler dataen der modtages i tre 16-bit variable, der hver udgør en sensor.

- **getSensorLeft()**
  Parametre: Ingen (*void*)
  Returværdi: uint16_t.
  Beskrivelse: Returnere data fra venstre sensor.

- **getSensorMid()**
  Parametre: Ingen (*void*)
  Returværdi: uint16_t.
  Beskrivelse: Returnere data fra midterste sensor.

- **getSensorRight()**
  Parametre: Ingen (*void*)

Returværdi: uint16_t.
Beskrivelse: Returnere data fra højre sensor.

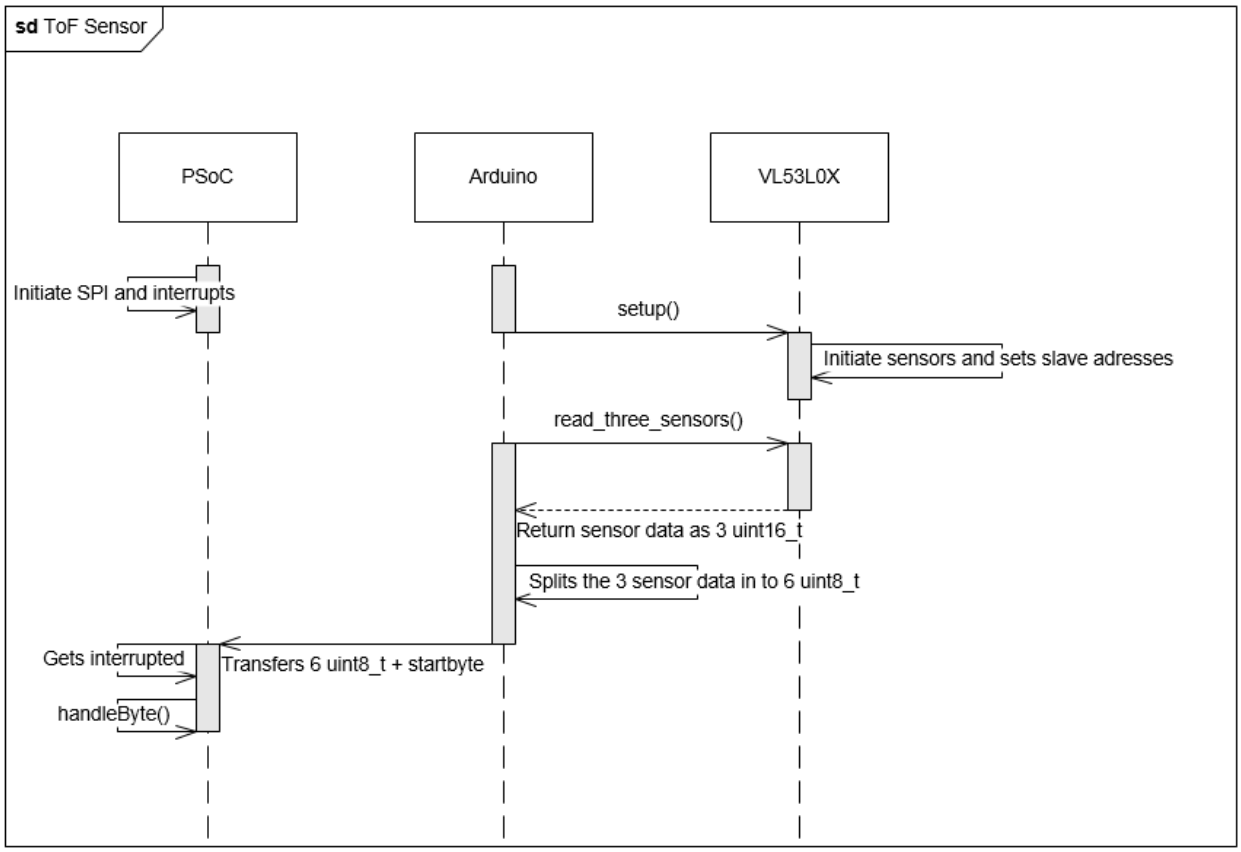
Arduino

- **setID()**
  Parametre: Ingen (*void*)
  Returværdi: Ingen (*void*).
  Beskrivelse: Sætter slave ID'et på hver af de tre sensorer.

- **read_three_ranges()**
  Parametre: Ingen (*void*)
  Returværdi: Ingen (*void*).
  Beskrivelse: Aflæser alle tre sensorer.

- **setup()**
  Parametre: Ingen (*void*)
  Returværdi: Ingen (*void*).
  Beskrivelse: Initiere SPI og I2C samt kalder setID().

- **loop()**
  Parametre: Ingen (*void*)
  Returværdi: Ingen (*void*).
  Beskrivelse: Aflæser de tre sensorer og videresender dataen. Dette fortsætter, indtil at strømmen tages fra Arduinoen.



Figur 24: Sekvens diagram over data indsamling fra ToF sensorer til PSoC

På Figur 24 ses et sekvens diagram over data indsamlingen fra sensorene til PSoC'en. Det ønskes, at dataen kommer hurtigst muligt. Dette skyldes, at det er sensor dataen der sætter grænsen for hvor hurtig reguleringssløjfen kan blive. Derfor spørger PSoC'en ikke efter data, den får det så snart Arduinoen har det klar. Der indføres dog et markant delay, i denne særlige måde at dataen overføres til PSoC'en. Det skyldes at hver gang sensor dataen skal opdateres, skal der modtages og håndteres 7 bytes.

Data opdelingen sker som vist på Listingen herunder. Det ses, at der i denne opdeling også ligges 1 til alle most significant bytes. Dette skyldes at start bytet der får PSoC'ens handler funktion til, at samle tre nye sensor værdier er 0. På denne måde er der aldrig nogle data der kan være 0. Denne +1 trækkes selvfølgelig fra i PSoC handler funktionen.

Listing 4: Opdeling af sensor data

```
1  sensorData[1] = (range_1>>8)+1;              \\MSB Sensor 1
2  sensorData[2] = range_1;                     \\LSB Sensor 1
```

```
3   sensorData[3] = (range_2>>8)+1;              \\MSB Sensor 2
4   sensorData[4] = range_2;                     \\LSB Sensor 2
5   sensorData[5] = (range_3>>8)+1;              \\MSB Sensor 3
6   sensorData[6] = range_3;                     \\LSB Sensor 3
```

Herunder ses et eksempel på hvordan handler funktionen samler dataen. Handleren er en switch case der der afhænger af index. index sættes til 0 når den modtaget byte er 0 derefter ligges der en til for hver byte modtaget. Sådan håndteres alle bytes modtaget bytes indtil næste 0 modtages og index igen sættes til 0.

Listing 5: Opdeling af sensor data

```
1          case 1:
2         byteR_ --;
3         sensorRight = byteR_<<8;
4         index++;
5         break;
6
7         case 2:
8         sensorRight = sensorRight|byteR_;
9         index++;
10        break;
```

Modultest

ToF sensorene testes ved, at tælle antallet af interrupts på PSoC'en i 10 sekunder. På de 10 sekunder interruptes 512 gange, dette skal dog deles med 7, da der interruptes 7 gange for at opdatere alle sensor værdierne. Derefter kan det deles med ti for at opnå samplingfrekvensen. Samplingfrekvensen fås til 7.3Hz, dette er meget langsommere end hvad var ønsket. Det skyldes dog i stor grad det tidligere nævnte delay.

2.6 Gyrosensor

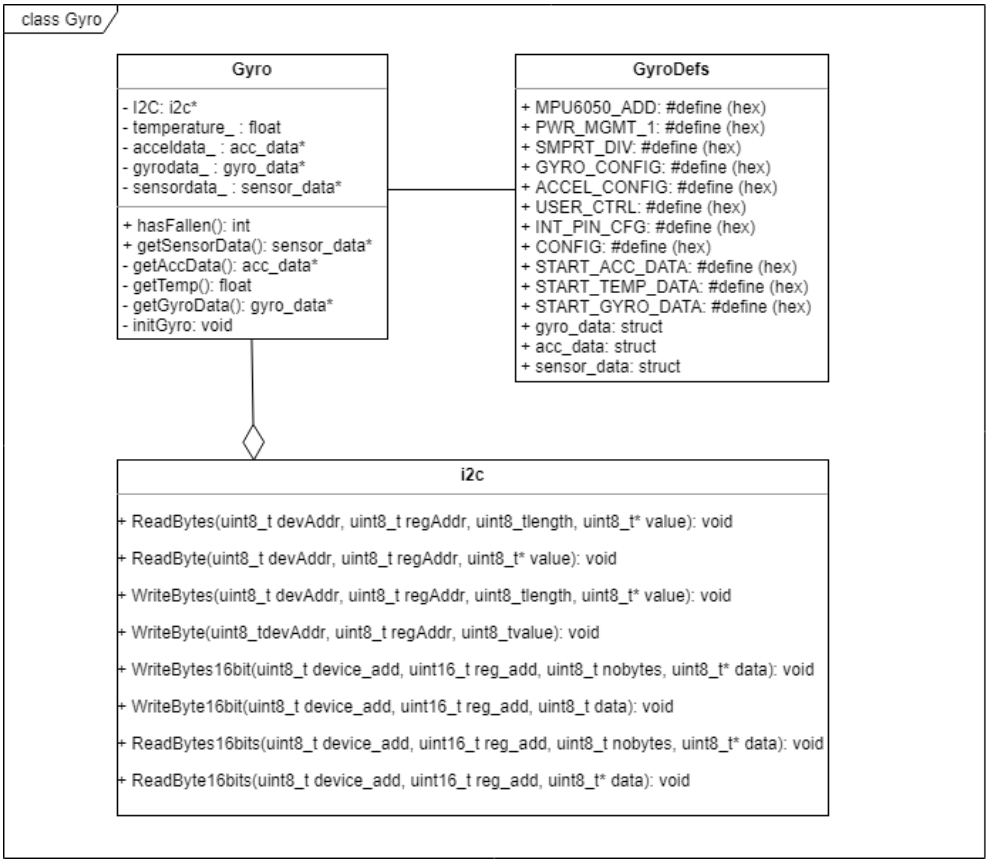
Selvom navnet på denne klasse er *Gyrosensor*, er dette faktisk en driver til understøttelse af *10DOF-Gyro + G-sensor + Kompas + Barometer*, alle på samme print. Derudover er det kun data fra G-sensoren der vil blive brugt fra denne klasse. Denne driver skrives dog til også at understøtte enhedens Gyro, i tilfælde af ønsket om implementering af selvbalancering på to hjul.

2.6.1 Modulbeskrivelse

Dette modul vil blive brugt til at hente og fortolke data fra 10DOF's G-sensor, for at kunne bestemme om robotten er væltet eller oprejst. Derudover vil klassens gyrosensor-data, kunne blive brugt til implementering af selvbalancering på to hjul.

2.6.2 Generel implementering

Modulet implementeres som en C++ klasse i PSoC-Creator, da modulet skal bruges i en PSoC-enhed. Klassen vil indeholde funktioner til aflæsning af sensor-data struktureret i objekter (*structs*) og hurtige funktioner til at hente status om orientation. Klassen vil benytte et I2C-objekt til kommunikationen med den fysiske hardware enhed. Funktionerne kan ses nedenfor i et klassediagram.



Figur 25: Klassediagram over Gyro-klassen og dens forhold

På den ovenstående Figur 25, ses det at Gyro-klassen har et i2c-modul, men også et *GyroDefs*-modul. *GyroDefs*-modulet indeholder diverse adresser på registre og devices, samt structs til gyro

og accelerometer-data, med en samlet struct til alt sensor data (*sensor_data*). Adresserne i *GyroDefs* implementeres som makroer (*Preprocessor directives*), for at spare plads på PSoC'en.

2.6.3 Funktionsbeskrivelser

Dette afsnit vil beskrive funktionerne i Gyro-klassen, med beskrivelser af funktionernes parametre, returværdier og formål.

Public funktioner

- **getSensorData()**
  Parametre: Ingen (*void*)
  Returværdi: Pointer til struct med sensor-data fra accelerometer og gyroskop (*sensor_data**).
  Beskrivelse: Bruges til at hente opdateret data fra sensorerne.

- **hasFallen()**
  Parametre: Ingen (*void*)
  Returværdi: Værdi der indikerer om robotten væltet: *0 for stående og 1 for væltet* (*int*).
  Beskrivelse: Bruges til at læse om robotten er væltet.

Private/hjælpe funktioner

- **getAccData()**
  Parametre: Ingen (*void*)
  Returværdi: Pointer til struct med accelerometer-data. (*acc_data**)
  Beskrivelse: Bruges til at hente opdateret data fra accelerometeret.

- **getTemp()**
  Parametre: Ingen (*void*)
  Returværdi: Sensorens aktuelle temperatur, bruges ikke i iFollow. (*float*)
  Beskrivelse: Bruges til at læse om sensorens temperatur.

- **getGyroData()**
  Parametre: Ingen (*void*)
  Returværdi: Pointer til struct med gyroskop-data. (*gyro_data**)
  Beskrivelse: Bruges til at hente opdateret data fra gyroskopet.

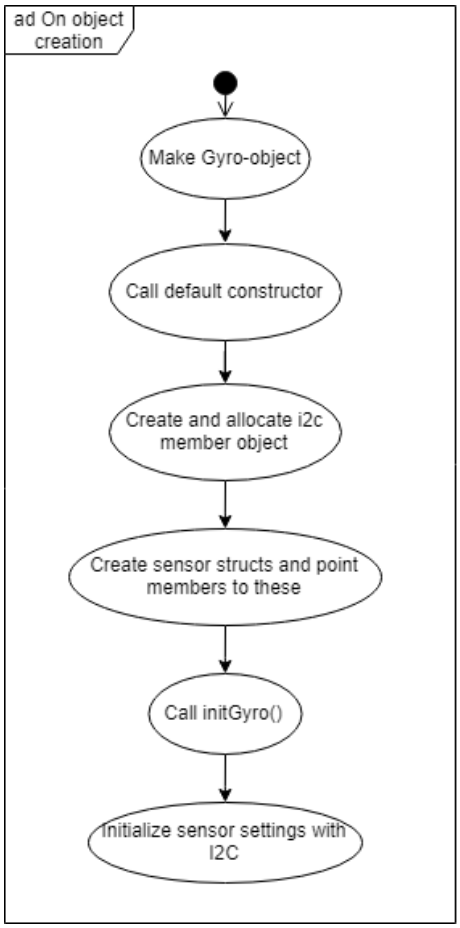
- **initGyro()**
  Parametre: Ingen (*void*)

Returværdi: Ingen (*void*).

Beskrivelse: Bruges til at initialisere sensoren.

De fleste funktioner beskrevet i de ovenstående funktionsbeskrivelser, er meget selvforklarende. Nedenfor ses et aktivitetsdiagram for oprettelse af et Gyro-objekt.

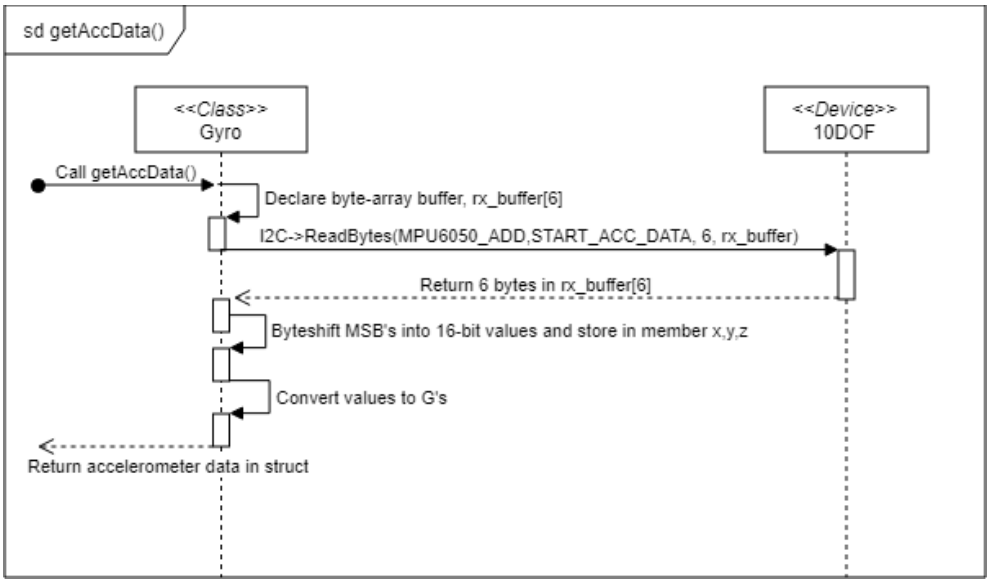
Aktivitetsdiagram for objekt-oprettelse



Figur 26: Aktivitetsdiagram for Gyro-objektoprettelse

Ovenfor på Figur 26, ses hvordan Gyro-objektet initialiserer sine member-variabler under oprettelse. Nedenfor vises et sekvensdiagram over *getAccData()*, for at vise hvordan data fra accelerometer hentes og fortolkes.

Sekvensdiagram for getAccData()



Figur 27: Sekvensdiagram for getAccData()

På ovenstående Figur 27, ses hvordan data hentes fra accelerometer-sensoren og hvordan dataen bagefter behandles. Denne struktur er den samme i alle get-metoderne, hvilke er hvorfor de ikke illustreres med sekvensdiagrammer. *getSensorData()*-funktionen, kalder blot alle get-funktionerne.

2.6.4 Attributbeskrivelser

I dette afsnit beskrives Gyro-klassens attributter, og deres formål. Derudover vises også hvordan de forskellige sensor-data structs er sat op.

- **i2c* I2C**
  Formål: Bruges til I2C-kommunikationen med sensorne.
- **float temperature_**
  Formål: Bruges til opbevaring af sensorens temperatur i Celsius.
- **acc_data* acceldata_**
  Formål: Bruges til opbevaring af sensorens accelerometer data i G's.
- **gyro_data* gyrodata_**
  Formål: Bruges til opbevaring af sensorens gyroskop data i $\frac{rad}{s}$.
- **sensor_data* sensordata_**
  Formål: Bruges til opbevaring af alle sensorernes data.

Ovenfor ses en overordnet beskrivelse af Gyro-klassens attributter, nedenfor vises hvordan de forskellige sensor-structs er opbygget.

Listing 6: Struct gyro_data

```
/*Structure for gyrosensor data*/
typedef struct gyro_data {
    int16_t x = 0;
    int16_t y = 0;
    int16_t z = 0;
    float x_rad = 0.0f;
    float y_rad = 0.0f;
    float z_rad = 0.0f;
} gyro_data;
```

Listing 7: Struct acc_data

```
/*Structure for accelerometer data*/
typedef struct acc_data {
    int16_t x = 0;
    int16_t y = 0;
    int16_t z = 0;
    float x_G = 0.0f;
    float y_G = 0.0f;
    float z_G = 0.0f;
} acc_data;
```

Listing 8: Struct sensor_data

```
/*Structure for all sensor data*/
typedef struct sensor_data {
    gyro_data *gyro_ = NULL;
    acc_data *acc_ = NULL;
    float *temperature_ = NULL;
    sensor_data(gyro_data *gyro,acc_data *acc,float *temperature);
    sensor_data(){}
} sensor_data;
```

De ovenstående structs indeholder både rå-data fra sensoren, samt data omskrevet til standardenheder. Enhederne kan aflæses i attributbeskrivelserne længere oppe.

Nedenfor vises hvordan de forskellige makroer er sat op for addresser, som kan findes i *GyroDefs.h*.

Listing 9: Makroer i GyroDefs.h

```
/*ADDRESSES FOR I2C-COMMUNICATION WITH MPU6050*/
#define MPU6050_ADD      (0x68)      // Device / Slave Address
#define PWR_MGMT_1       (0x6B)      // Reset with 0x00 to wakeup
#define SMPRT_DIV        (0x19)      // Register for holding sample rate 109
#define GYRO_CONFIG      (0x1b)      // Register for self-test and scale 0x18 for
    fullscale
#define ACCEL_CONFIG     (0x1c)      // --||-- 0x08 fullscale +- 4g
#define USER_CTRL        (0x6a)      // 0x00 to disable master mode | 0x20 to enable
#define INT_PIN_CFG      (0x37)      // 0x02 I2C bypass enable
#define CONFIG           (0x1A)      // For setting DLPF - fast or slow
#define START_GYRO_DATA (0x43)       // First address in gyro registers      | 6bytes
    long
#define START_ACC_DATA   (0x3b)      // First address in accelerometer       | 6bytes
    long
#define START_TEMP_DATA (0x41)       // First address in temperature sensor  | 2bytes
    long
```

Makroerne som ses ovenfor bliver bl.a. brugt til opsætning af sensorer og læsning af sensor-registre.

2.7 Motor

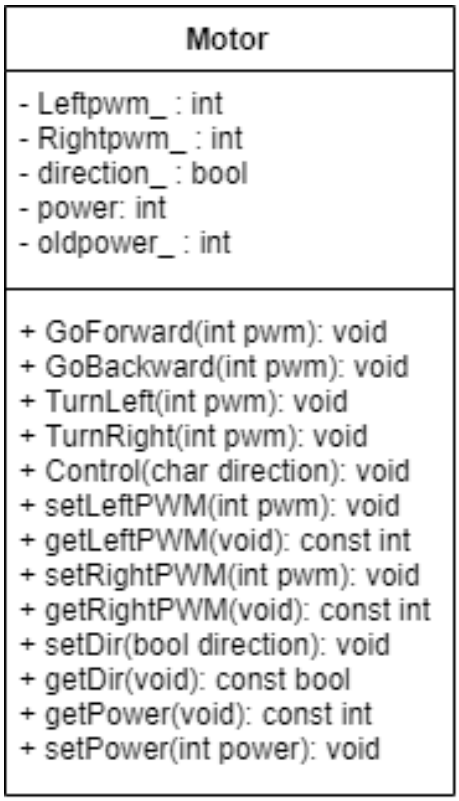
For at robotten har mulighed for at kører, kræver det en motorstying. For at styre motorene kræver det at der er nogen software, som sender to PWM signaler og to retnings signaler ud. Disse signaler vil herefter gå ind i noget hardware og tilsidst få motorern til at roterer. Dette motor modul vil hovedsageligt blive brugt af to andre klasser. Disse klasser er SPI klassen og PIDregulator klassen.

2.7.1 Modulbeskrivelse

Som tidligere nævnt, vil motor-modulet blive brugt til kontrol af motorerne. Alt reguleringen bliver styret af en anden klasse. Motor-modulet skal dog have nogle kontrol funktioner, som der vil blive kommet ind på senere.

2.7.2 Generel implementering

Modulet implementeres som en C++ klasse i PSoC-Creator, da det er PSoC'ens, som skal styre motorne. Klassen vil indeholde funktioner til styring af motorernes retning og hastighed. Udover dette vil klassen også indeholde en timer, som skal sørge for at motorerne automatisk "slukker" når der ikke bliver sendt et kontrol signal. Funktionerne kan ses i det nedenstående klassediagram.



Figur 28: Klassediagram over Motor-Klassen

Funktioner på det ovenstående klassediagram beskrives i næste afsnit.

2.7.3 Funktionbeskrivelser

Dette afsnit vil beskrive funktionerne i Motor-klassen, med beskrivelser af funktionernes parametre, returværdier og formål.

- **GoForward(*int pwm*)**
  Parametre: PWM værdi på 0-100 %.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at få motorerne til at kører fremad med en specifik PWM.

- **GoBackward(*int pwm*)**
  Parametre: PWM værdi på 0-100 %.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at få motorerne til at kører bagud med en specifik PWM.

- **TurnLeft(*int pwm*)**
  Parametre: PWM værdi på 0-100 %.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at få motorerne til at kører til venstre med en specifik PWM.

- **TurnRight(*int pwm*)**
  Parametre: PWM værdi på 0-100 %.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at få motorerne til at kører til højre med en specifik PWM.

- **Control(*char dir*)**
  Parametre: En retning som motoreren skal køre. Den kan være 'f', 'b', 'l' og 'r'.
  Returværdi: Ingen (*void*)
  Beskrivelse: Denne funktion bliver brugt til at kontrollere motorerne. Funktionen fungere sammen med en timer så, hvis funktionen bliver kaldt konstand kører motoren hurtigere og hurtigere. Hvis funktionen ikke kaldes resetter timeren motorerne.

- **SetLeftPWM(*int PWM*)**
  Parametre: PWM værdi på 0-100 %.
  Returværdi: Ingen (*void*)
  Beskrivelse: Sætter compare værdien i timeren til at få det korrekte PWM signal og sætter en privat variable, så PWM procenten kan findes.

- **GetLeftPWM(*void*)**
  Parametre: Ingen (*void*)
  Returværdi: PWM værdi på 0-100 %.
  Beskrivelse: Retunerer PWM værdien for den private variable RightPWM_.

- **SetRightPWM(*int PWM*)**
  Parametre: PWM værdi på 0-100 %.
  Returværdi: Ingen (*void*)
  Beskrivelse: Sætter compare værdien i timeren til at få det korrekte PWM signal og sætter en privat variable, så PWM procenten kan findes.

- **GetLeftPWM(*void*)**
  Parametre: Ingen (*void*)
  Returværdi: PWM værdi på 0-100 %.
  Beskrivelse: Retunerer PWM værdien for den private variable LeftPWM_.

- **SetDir(*int PWM*)**
  Parametre: PWM værdi på 0-100 %.
  Returværdi: Ingen (*void*)
  Beskrivelse: Sætter direction, hvilket kan være 'f' og 'b', i den private variable direction_.

- **GetDir(*void*)**
  Parametre: Ingen (*void*)
  Returværdi: PWM værdi på 0-100 %.
  Beskrivelse: Retunerer værdien for den private variable direction_.

- **SetPower(*int PWM*)**
  Parametre: Power værdi på 0-100.
  Returværdi: Ingen (*void*)
  Beskrivelse: Sætter en power værdi som timeren bruger til at se på, hvor hurtigt motoerne skal kører. Værdien bliver gemt i en privat variable power_.

- **GetPower(*void*)**
  Parametre: Ingen (*void*)
  Returværdi: Power værdi.
  Beskrivelse: Retunerer værdien for den private variable power_.

2.7.4 Attributbeskrivelse

Dette afsnit vil beskrive attributterne i Motor-klassen og, hvad deres formål er.

- **int Leftpwm_**
  Formål: At gemme den nuværende værdi på PWM signalet, som bliver brugt på den venstre motor.

- **int Rightpwm_**
  Formål: At gemme den nuværende værdi på PWM signalet, som bliver brugt på den højre motor.

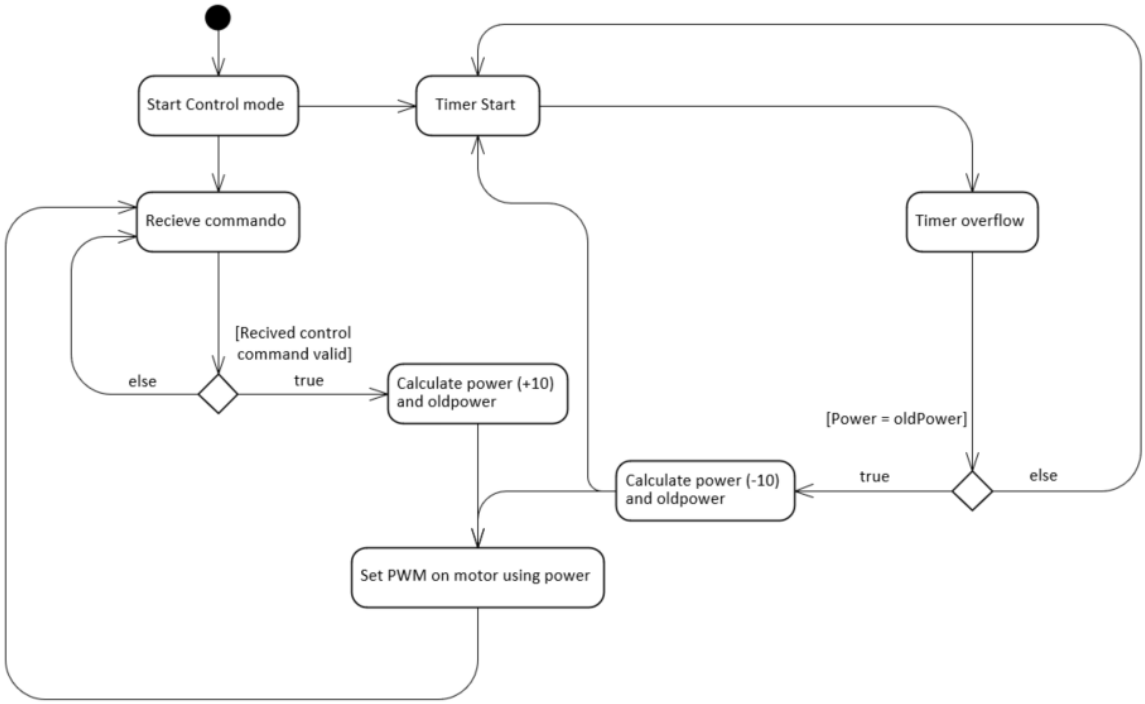
- **char direction_**
  Formål: At gemme den retning som motorerne kører.

- **int power_**
  Formål: At gemme den værdi som bliver trukket op hver gang control bliver kaldt. Denne værdi vil blive trukket ned, når timeren løber ud og power_ = oldpower_.

- **int oldpower_**
  Formål: At gemme den gamle værdi af power_ for at kunne se om power_ har ændret sig.

2.8 Aktivitets diagram

Da control() funktionen bruger en timer til at lave en ramp op og ned effekt, er der blevet lavet et aktivitets diagram for denne. Aktivitets diagrammet kan ses på figur 29 herunder.



Figur 29: Aktivitetsdiagram over control() funktionen

2.9 Modul test

For at være sikre på at motorklassen fungerede efter hensigten blev klassen testet. Måden som motorklassen blev testet på var ved hjælp af enten forloops eller en UART.
For loopsne blev brugt til at teste GoForward, GoBackward, TurnLeft og TurnRight, hvor en counter blev ved med at sætte hastigheden på motoren højere og højere, hvorefter den startede forfra.
UART'en blev brugt til at teste control funktionen, da denne funktion kræver at blive kaldt flere gange for at få motoreren til at kører hurtigere.

De resterede funktioner blev testet igennem control, GoForward, GoBackward, TurnLeft og TurnRight, da de bliver brugt til at sætte eller hente værdier fra private variabler.

2.10 Switches

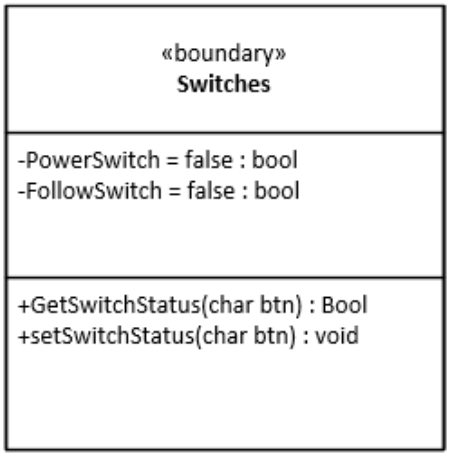
For at robotten kan starte op og at den kan følge efter en person, bliver der nød til at være et interface på robotten. Dette interface er på iFollow to switches, som henholdsvis sørger for at power on og gå i follow mode.

2.10.1 Modulbeskrivelse

Dette modul har til opgave og hente data ind fra de to switches, gemme deres værdier og returnere værdien. Modulet vil for det meste blive brugt at main, men kan også blive brugt af andre moduler.

2.10.2 Generel implementering

Modulet implementeres som en C++ klasse i PSoC-Creator, da det er PSoC'ens, som skal læse status og ændre status på de input. Klassen vil indeholde funktioner til at gemme status og returnere den gemte data. Funktionerne kan ses i det nedenstående klassediagram.



Figur 30: Klassediagram over switches-Klassen

Funktioner på det ovenstående klassediagram beskrives i næste afsnit.

2.10.3 Funktionbeskrivelser

Dette afsnit vil beskrive funktionerne i Switches-klassen, med beskrivelser af funktionernes parametre, returværdier og formål.

- **GetSwitchStatus(*char button*)**
  Parametre: Button som enten kan være 'p' eller 'f'. **Returværdi:** bool - Switches status.

Beskrivelse: Bruges til at se status på knapperne. Hvis knappen bliver trykket ind vil status ændre sig en gang.

- **SetSwitchStatus(*char button*)**
  Parametre: Button som enten kan være 'p' eller 'f'. **Returværdi:** Ingen (*void*)
  Beskrivelse: Bruges til at sætte status på knapperne. Hvis knappen bliver trykket ind vil status ændre sig en gang.

2.10.4 Attributbeskrivelse

Dette afsnit vil beskrive attributterne i Switches-klassen og, hvad deres formål er.

- **bool PowerSwitch_**
  Formål: At gennem den nuværende status på PowerSwitch. Knapper fungere som et kip relæ, så de holder status indtil de bliver trykket på igen.

- **bool FollowSwitch_**
  Formål: At gennem den nuværende status på FollowSwitch. Knapper fungere som et kip relæ, så de holder status indtil de bliver trykket på igen.

2.11 Modul test

For at sikre switches klassen virkede som den skulle var det nødvendigt, at teste funktionerne. Dette blev gjort ved hjælp af en enkelt LED, som blev tændt, hvis der blev trykket på en knap.
Ved denne test blev alle funktioner testet da set funktionen blev testet ved at sætte værdien af den private variable og get blev brugt til at hente info om LED'en skulle tændes.

2.12 LED

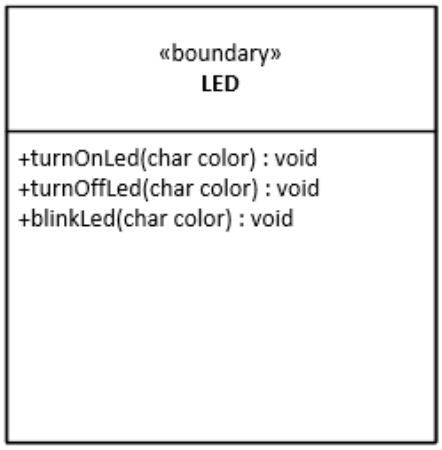
For at give brugeren af iFollow informationer om, hvilken status robotten er i eller om der er et problem sidder der to LED'en på robotten. For at disse LED'en fungere er der blevet skrevet en klasse, der styrer dem.

2.12.1 Modulbeskrivelse

Dette modul har til opgave og skrive til de to LED'er, som sidder sammen med de to knapper på robotten. LED'erne kan enten blinke eller konstant tændt.

2.12.2 Generel implementering

Modulet implementeres som en C++ klasse i PSoC-Creator, da det er PSoC'ens, som tænde, slukke eller blinke med LED'erne. Klassen vil indeholde funktioner til at sætte status på LED'erne. Funktionerne kan ses i det nedenstående klassediagram.



Figur 31: Klassediagram over LED-Klassen

Funktioner på det ovenstående klassediagram beskrives i næste afsnit.

2.12.3 Funktionbeskrivelser

Dette afsnit vil beskrive funktionerne i LED-klassen, med beskrivelser af funktionernes parametre, returværdier og formål.

- **turnOnLed(*char Color*)**
  Parametre: Color er farven på den LED som skal tændes. Dette kan enten være 'r' eller 'g'.

Returværdi: Ingen (*void*)
Beskrivelse: Bruges til at tænde for en specifik LED.

- **turnOffLed(*char Color*)**
  Parametre: Color er farven på den LED som skal slukkes. Dette kan enten være 'r' eller 'g'.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at slukke for en specifik LED.

- **blinkLed(*char Color*)**
  Parametre: Color er farven på den LED som skal blinke. Dette kan enten være 'r' eller 'g'.
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at blinke med en specifik LED.

2.13 Modul test

For at være sikre på at LED klassen virkede blev alle funktionerne testet i et main program for sig selv. I dette main program blev funktionerne kaldt så LED'en først blev tændt, herefter blev den sat til at blink og tilsidst blev den slukket igen. Dette blev gjort med begge LED'er.

2.14 rpiSPI

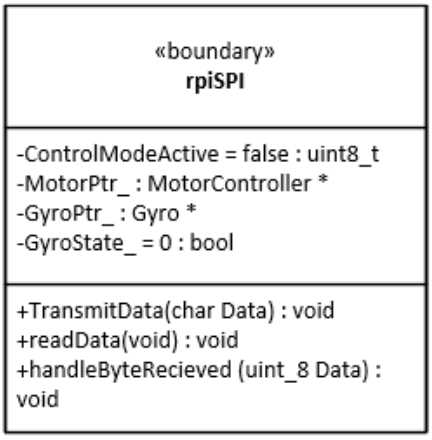
For at iFollow kan kommunikere med RPI'en er det nødvendigt at implementere en klasse, som sørger for kommunikation imellem disse to. Denne klasse sørger for at sende, modtage og handle den modtagende data.

2.14.1 Modulbeskrivelse

Dette modul har til opgave at kommunikere med RPI'en og agere på de data som den modtager.

2.14.2 Generel implementering

Modulet implementeres som en C++ klasse i PSoC-Creator, da det er PSoC'ens, som skal kommunikere med RPI'en. Funktionerne kan ses i det nedenstående klassediagram.



Figur 32: Klassediagram over rpiSPI-Klassen

Funktioner på det ovenstående klassediagram beskrives i næste afsnit.

2.14.3 Funktionbeskrivelser

Dette afsnit vil beskrive funktionerne i rpiSPI-klassen, med beskrivelser af funktionernes parametre, returværdier og formål.

- **TransmitData(*uint8_t Data*)**
  Parametre: Data er det, som ønskers transmitteret til RPI'en **Returværdi:** Ingen (*void*)
  Beskrivelse: Denne funktion bruges til at sende data til RPI'en.

- **readData(*void*)**
  Parametre: Ingen (*void*)
  Returværdi: Returnere enten 0, 'c' eller 'o'
  Beskrivelse: Denne funktion bruges til at læse, hvad der bliver sendt fra RPI'en. Grunden til at den har en returværdi er grundet mainprogrammet skal bruge info om kontrolmode er aktiv eller ej.

- **handleByteRecieved(*uint8_t byteReceived*)**
  Parametre: byteReceived er den data som er modtaget og den skal handle.
  Returværdi: Returne enten 0, 'c' eller 'o'
  Beskrivelse: Denne funktion bruges til at handle de modtagede data. Dette er i form af kommandoer til motorerne eller om kontrolmode er aktiv. Dette er også grunden til at funktionen returnere en værdi, da main programmet så ved at den er i kontrol mode eller ej.

2.14.4 Attributbeskrivelse

Dette afsnit vil beskrive attributterne i rpiSPI-klassen og, hvad deres formål er.

- **uint8_t ControlModeActive_**
  Formål: Denne variable sørger for at holde styr på om systemet er i kontrolmode eller ej.

- **MotorController * MotorPtr_**
  Formål: Denne variable er en pointer til motorklassen, som kontrollere motorerne. Denne variable er med da klassen skal sende de data videre som den modtager.

- **Gyro * GyroPtr_**
  Formål: Denen variable er en pointer til gyro klassen, da iFollow skal sende data tilbage til RPI'en om den er væltet eller ej.

- **uint8_t GyroState_**
  Formål: Denne variable er til at indeholde den state som gyroklassen retunerer.

2.14.5 Aktivitets diagram

For at give et godt overblik over, hvordan readData fungere er der blevet lavet et aktivitetsdiagram over funktionen. Diagrammet kan ses på figur 33 herunder.

Figur 33: Aktivitetsdiagram for rpiSPI-Klassen

2.15 Modul test

For at være sikre på at rpiSPI klassen fungerede som den skulle var det nødvendigt at teste den med en anden enhed, som kunne sende en SPI besked til den.
Der blev derfor opsat et testprogram på en anden microcontroller, som kunne sende forskellige beskeder til PSoC'en og printe svaret ud på en terminal.
Herefter blev PSoC'en sat sammen med motor'eren og klassen blev testet sammen med motorklassen.

2.16 PIDRegulator

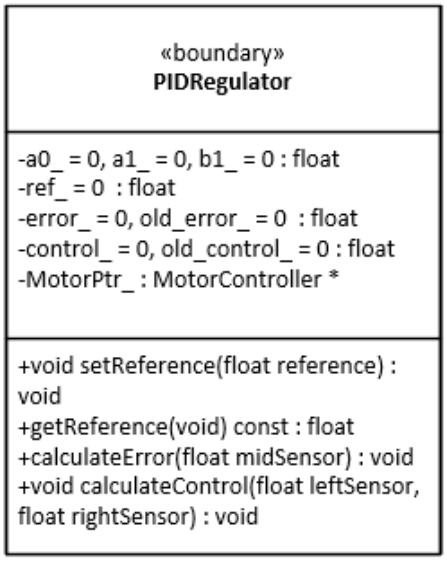
For at iFollow kan følge efter en person er det nødvendigt at implementere en regulator, som kan regulere afstanden. Derfor er det nødvendigt med en Regulator klasse.

2.16.1 Modulbeskrivelse

Dette modul har til opgave at regulere afstanden fra iFollow til et objekt det bevæger sig. Udover dette skal klassen også kunne finde ud af at dreje med objektet, hvis det går rundt om et hjørne.

2.16.2 Generel implementering

Modulet implementeres som en C++ klasse i PSoC-Creator, da det er PSoC'ens, som styre motorene og derfor er den som skal regulere på dem. Funktionerne kan ses i det nedenstående klassediagram.



Figur 34: Klassediagram over RPI-Klassen

Funktioner på det ovenstående klassediagram beskrives i næste afsnit.

2.16.3 Funktionbeskrivelser

Dette afsnit vil beskrive funktionerne i LED-klassen, med beskrivelser af funktionernes parametre, returværdier og formål.

- **setReference(*float reference*)**
  Parametre: Reference paramteren er den ønskede afstand som regulatoren skal opretholde.
  Returværdi: Ingen (*void*)
  Beskrivelse: Denne funktion skal sørge for at sætte referencen i regulatoren.

- **getReference(*void*)**
  Parametre: Ingen (*void*)
  Returværdi: Returnerer reference
  Beskrivelse: Sørger for at retunere værdien af den private variable reference_

- **calculateError(*float midSensor*)**
  Parametre: Sensor data for den sensor, som sidder i midten.
  Returværdi: Ingen (*void*)
  Beskrivelse: Udregner fejlen imellem referencen og den målte værdi

- **calculateControl(*float leftSensor, float rightSensor*)**
  Parametre: Sensor data fra både højre og venstre sensor
  Returværdi: Ingen (*void*)
  Beskrivelse: Udregner, hvor hurtigt og hvilken vej iFollow skal kører, for at følge med et objekt.

2.16.4 Attributbeskrivelse

Dette afsnit vil beskrive attributterne i PIDRegulator-klassen og, hvad deres formål er.

- **float a0_, a1_, b1_**
  Formål: Dette er regulatorens værdier som bliver ganget på fejl, gammel fejl, control og gammel control.

- **float ref_**
  Formål: Dette er regualtoren reference som bliver brugt til at udregne fejlen.

- **float error_, old_error**
  Formål: Variabler til at indehold fejlen og den gamle fejl

- **float control_, old_control**
  Formål: Variabler til at indehold control signalet og det gamle controlsignal

- **MotorController *MotorPtr_**
  Formål: Pointer til motorklassen så regulatoren kan styre motorerne med det beregnede kontrol signal.

2.17 Modul test

For at være sikre på at PIDregulator klassen fungerede som den skulle opsatte vi et teste sammen med arudino'en. Denne test blev lavet lige efter intergrationen, da det ellers ikke ville være muligt at sætte de to enheder sammen.
Testen vidste at klassen sagtens kunne udregne værdier og sende dem videre til motorerne, men at værdierne var forkerte. Dette mistænkte vi regulerings koefficienterne og samplingshastigheden for at være skyld i.

2.18 DatabaseApp

DatabaseApp eller Databaseapplikationen er den del af dokumentet som skal fungere som kontroller programmet. Det er her hele funktionaliteten af databaseapplikationen bliver skrevet og alle objekterne til klasserne bliver oprettet. For udviddet klassediagram for hele DatabaseApplikationen og tilhørende klasser henvises til *Systemarkitektur*-dokumentet.

2.18.1 Modulbeskrivelse

Som nævnt tidligere har dette modul til formål at være kontrollerklassen for databaseapplikations programmet hvor hovedstrukturen af programmet bliver opbyggets.

2.18.2 Generel implementering

Modulet implementeres som en C++ Main på Raspberry Pi. Koden er skrevet i editoren Visual Studio Code. Da dette modul ikke er en klasse men istedet en kontrollerklasse vil hele funktionaliteten blive opbygget i et gentagende main lykke. Denne kontrollerklasser bruger også alle de efterfølgende klasser.

2.19 GPS

GPS Klassen er den klasse som står for at oprette opjekt med funktionalitetten af åbne den fildescripter som objektet bliver lavet med, derudover er dette også klassen for funktionaliteten til at hente GPS positionen ned.

2.19.1 Modulbeskrivelse

Denne klasse er til ene formål at opdatere NMEA_GPGGA klassen med nye koordinater og GPS parametre hvor gang klassens funktion bliver kørt. Det er derfor denne klasse som har alt funktionalitet til at opdatere positioner for systemet.

2.19.2 Generel implementering

Modulet vil blive implementeret som en C++ klasse. Klassens ene funktion updateCoordinates sørger som tidligere nævnt for at opdatere GPS data i objekt klassen NMEA_GPGGA. Denne klasse bliver implementeret med en Linux fstream[6] library til at åbne og aflæse filer. Derudover bliver der også brugt strings hvor biblioteker som stdio string til at opdele dataene i GPS strengen som bliver modtaget fra GPS'en.

2.19.3 Funktionbeskrivelser

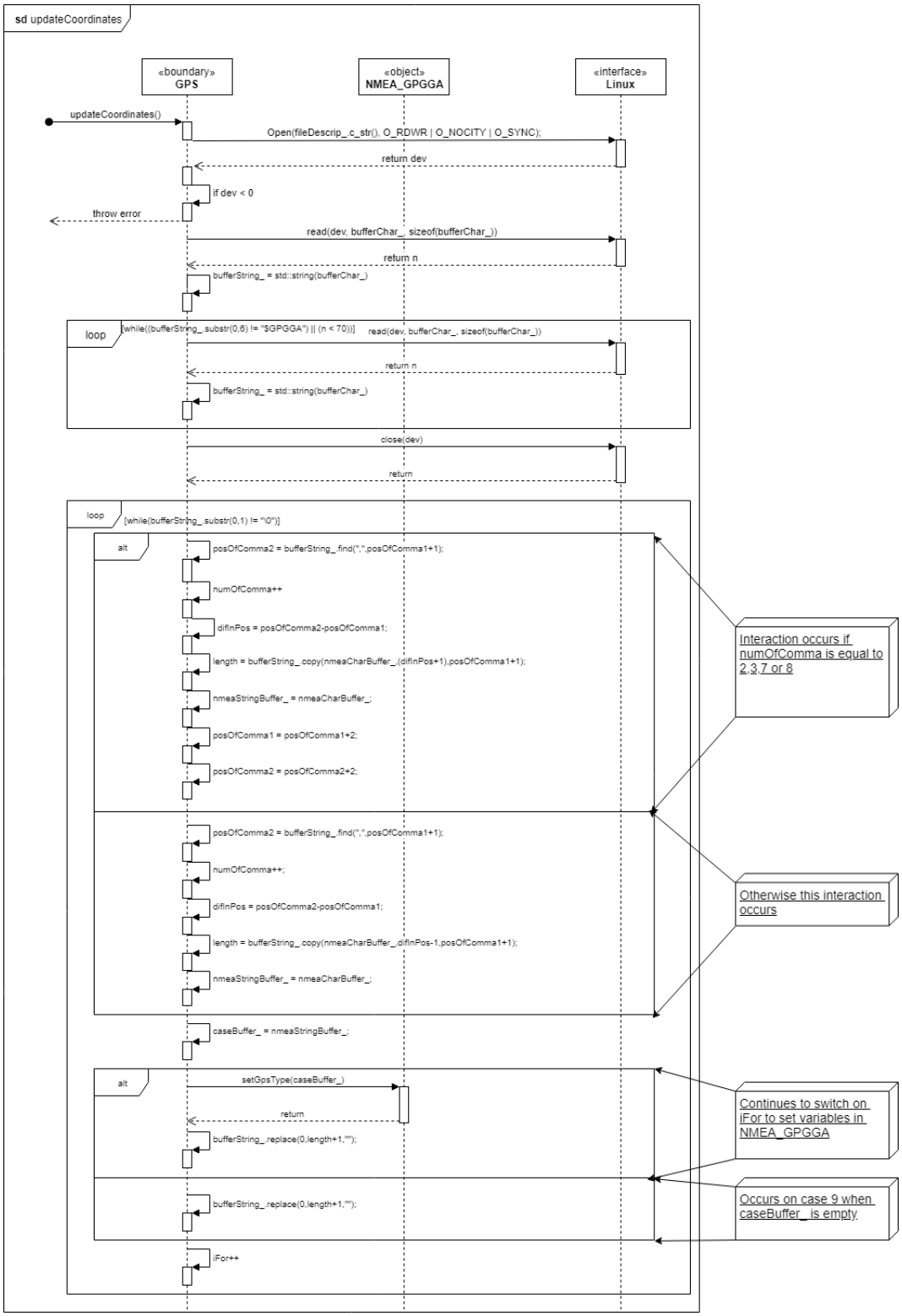
Dette afsnit vil beskrive funktionerne i GPS klassen med beskriveler af funktionernes formål.

- **GPS()**
  Parametre: Ingen parametre (*void*). Bruger default fileDescripter_
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at oprette standart objekt med default fileDescripter til at åbne UART filen.

- **GPS(*std::string fileDescrip_*)**
  Parametre:(*fileDescripter til åbning af UART fil*)
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at oprette objekt med fileDescripter hvor GPS koordinater skal hentes fra.

- **updateCoordinates(*void*)**
  Parametre: Ingen (*void*)
  Returværdi: Ingen (*void*)
  Beskrivelse: Bruges til at åbne, opdele og indsætte GPS data i objektklassen NMEA_GPGGA.

pga. updateCoordinates og dens omfang er der valgt at lave et sekvensdiagram til hvordan denne skal fungere. Dette sekvensdiagram kan ses på figur 35



Figur 35: Sekvensdiagram for updateCoordinates()

2.20 mySQLGPS

Dette er modulet hvorpå kommunikation til at tilgå databasen fra programmet bliver implementeret. Denne klasses funktion er at oprette funktioner hvorpå der kan ses data til databasen og tjekke om hvor meget data der ligger på databasen

2.20.1 Modulbeskrivelse

Denne klasse til have funktioner til at tilslutte mySQL databasen, disconnect fra denne, sende almindelige querys, sende sikre querys, få antallet af datasæt på databasen og fjerne det ældste datasæt på mySQL databasen.

2.20.2 Generel implementering

Implementeringen af dette modul vil blive udført i C++ med brug af library fra mySQL[7] som giver funktioner til at tilgå serveren, Derudover kan det også med dette library oprettes mySQL objekter hvori attributer til databasen kan sættes.

2.20.3 Funktionbeskrivelser

Dette afsnit vil beskrives funktionerne i mySQLGPS klassen med beskrivelser af funktionernes formål.

- **mySQLGPS(*std::string DBHOST, std::string USER, std::string PASSWORD, std::string DATABASE, unsigned int PORT*)**
  Parametre: Parametre til IP-adresse, string til brugernavn og password, hvilken database og hvilken port forbindelsen skal oprettes igennem (*std::string, std::string, std::string, std::string, unsigned int*)
  Returværdi: Ingen (*void*)
  Beskrivelse: Sætter de private attributer som bruges til at oprette forbindelse til databasen

- **mysql_connect(*void*)**
  Parametre: Ingen (*void*)
  Returværdi: Ingen (*void*)
  Beskrivelse: Opretter forbindelse til databasen således at der kan med objektet sendes querys til databasen.

- **mysql_disconnect(*void*)**
  Parametre: Ingen (*void*)
  Returværdi: Ingen (*void*)
  Beskrivelse: Afslutter forbindelsen til databasen

- **mysql_sendQUERY(*std::string QUERY*)**
  Parametre: Streng som indeholder den query som der skal sendes til databasen (*std::string*)
  Returværdi: Ingen (*void*)
  Beskrivelse: Denne funktion bliver brugt til at sende data til databasen hvorpå der ikke bliver sende følsomme data.

- **mysql_COUNT(*void*)**
  Parametre: Ingen (*void*)
  Returværdi: Returnere værdien af antallet af datasæt som ligger på databasen (*Unsigned int*)
  Beskrivelse: Bruges til at finde ud af hvor mange elementer der ligger på databasen.

- **mysql_send_delete_row(*void*)**
  Parametre: Ingen (*void*)
  Returværdi: Ingen (*void*)
  Beskrivelse: Sletter det ældeste datasæt som ligger på databasen.

- **mysql_secure_sendQUERY(*std::string QUERY, GPS myGPS*)**
  Parametre: Streng som indeholder query som skal sendes til databasen, GPS objekt som bruges til at tjekke data i strængen (*std::string, GPS*)
  Returværdi: Ingen (*void*)
  Beskrivelse: Funktioner bruges til at sende følsomme informationer til server og tjekker således også med objektet at dataene der sendes i queryen er korrekt i forhold til hvad der står i GPS objektet.

2.21 NMEA_GPGGA

Dette modul er det modul hvorpå data fra GPS'en bliver midlertidigt gemt i inden dette sendes til mySQL databasen.

2.21.1 Modulbeskrivelse

Som tidligere beskrevet har denne klasse som funktion til kun at opholde data inden dette sendes til mySQL serveren.

2.21.2 Generel implementering

Dette modul bliver implementeret i en C++ klasse med set og get metoder til alle de private variabler. Alle disse attributer bliver implementeret privat så der skal bruge metodekald til at ændre disse. Dette giver en sikker datastruktur til den midlertidlige klasse.

2.21.3 Funktionbeskrivelser

Alle de efterfølgende private variable har set og get metoder og disse vil ikke blive forklaret yderligere da disse bare er standart set og get metoder for private variabler

- std::string **gpsType_**;
- std::string **fixTime_**;
- std::string **Latitude_**;
- std::string **Longitude_**;
- std::string **fixQuality_**;
- std::string **numOfSats_**;
- std::string **hDilutionofPos_**;
- std::string **altitudeMeters_**;
- std::string **hOfGeoid_**;
- std::string **checkSum_**;

3 Referenceliste

[1] Funktion for formattering af driftstid. https://www.tutorialspoint.com/How-to-convert-seconds-to-HH-MM-SS-with-JavaScript. Sidst besøgt 23 Maj 2019.

[2] Skabelon-funktion for generering af canvas pushpin. https://docs.microsoft.com/en-us/bingmaps/v8-web-control/map-control-concepts/pushpins/custom-canvas-pushpin-example. Sidst besøgt 23 Maj 2019.

[3] Css - wikipedia side. https://da.wikipedia.org/wiki/CSS. Sidst besøgt 23 Maj 2019.

[4] Html - wikipedia side. https://da.wikipedia.org/wiki/HTML. Sidst besøgt 23 Maj 2019.

[5] Javascript - wikipedia side. https://da.wikipedia.org/wiki/JavaScript. Sidst besøgt 23 Maj 2019.

[6] Library for fstream til linux open. https://docs.microsoft.com/en-us/cpp/standard-library/fstream?view=vs-2019, . Sidste besøgt 23 Maj 2019.

[7] Library for mysql in c++. https://packages.debian.org/stretch/default-libmysqlclient-dev, . Sidst besøgt 23 Maj 2019.

[8] File-system pakke til node.js. https://nodejs.org/api/fs.html#fs_file_system, . Sidst besøgt 23 Maj 2019.

[9] Mysql node.js pakke. https://www.npmjs.com/package/mssql, . Sidst besøgt 23 Maj 2019.

[10] Pakke for node.js til http-requests. https://www.npmjs.com/package/request, . Sidst besøgt 23 Maj 2019.

[11] Websocket pakke til node.js for at tillade kommunikation mellem browser og webserver. https://socket.io/, . Sidst besøgt 23 Maj 2019.

[12] Pakke for spi-seriel bus adgang med node.js. https://www.npmjs.com/package/spi-device, . Sidst besøgt 23 Maj 2019.

[13] Software kode, 2019. iFollow/RapportBilag/Kode.

[14] Conrad Electronic. Datablad for DC-gear motor, 30 November 2014. iFollow/RapportBilag/Design/Motor/RB350018-2A723R.

[15] Ryan Dahl. Node.js. `https://nodejs.org/en/`. Sidst besøgt 23 Maj 2019.

[16] Microsoft. Bing maps documentation. `https://docs.microsoft.com/en-us/bingmaps/`. Sidst besøgt 22 Maj 2019.

[17] Norman S. Nise. Reducering af blok diagrammer, 2015. iFollow/RapportBilag/Design/Motor/blok_diagram_reduction.pdf.

[18] Norman S. Nise. Generel Overføringsfunktion formel, 2015. iFollow/RapportBilag/Design/Motor/Control-Systems-Engineering-Wiley-2015.pdf (side 80, ligning 2.153).

[19] Sengpielaudio. Omregner til omregning af dB til Gain. `http://www.sengpielaudio.com/calculator-gainloss.htm`, 2019.