Aarhus School of Engineering

Electronic Engineering

E4PRJ

Detaljeret Software Design

Author:
Nicolai Glud
Johnny Kristensen
Rasmus Lund-Jensen
Mick Holmark
Jacob Roesen

6. december 2012

Indholdsfortegnelse

Kapitel 1 Indledning **4**
1.0.1 Formål . . . 4
1.0.2 Reference dokumentation . . . 4

Kapitel 2 Kontrolinterface **5**
2.0.3 Modulets ansvar . . . 5
2.0.4 Klassediagram . . . 5
2.1 Metode- og klassebeskrivelser . . . 7
2.1.1 MainWindow . . . 7
2.1.2 Kontrolinterface . . . 11
2.1.3 DataServer . . . 11
2.1.4 manuDialog . . . 12
2.1.5 Styringsmodul . . . 13
2.1.6 VBTE . . . 14
2.1.7 Sensor . . . 14
2.1.8 RS232 . . . 14

Kapitel 3 Databasen **16**
3.0.9 Modultes Ansvar . . . 16
3.0.10 Klassediagrammer . . . 16
3.0.11 Globale variabler . . . 18
3.0.12 Funktionsbeskrivelser . . . 18
3.1 Design . . . 21
3.1.1 Server . . . 21

Kapitel 4 SM **22**
4.1 Klassens ansvar . . . 22
4.2 Klassediagram . . . 22
4.3 Variabler . . . 23
4.4 Funktionsbeskrivelser . . . 23
4.4.1 Init . . . 23
4.4.2 Levelsensor . . . 23
4.4.3 autoReg . . . 23
4.4.4 I2C_Kom . . . 24
4.4.5 KI_KOM . . . 24
4.5 Eventuelle Sekvensdiagrammer og state machines . . . 25

Kapitel 5 VBTE **26**
5.1 Modulets ansvar . . . 26
5.2 Klassediagram . . . 26
5.3 Globale variabler . . . 27

5.4 Metode- og klassebeskrivelser 27
5.4.1 Init 27
5.4.2 Valve 27
5.4.3 Dist 28
5.4.4 I2Chandle 28
5.4.5 PSoC-API::ADC 29
5.4.6 State Machine 29
5.4.7 Timing Diagram 30
5.4.8 Interrupt rutiner 30
5.5 Hovedvindue 31
5.6 Manuel regulering af hældning 32
5.7 Automatisk regulering af hældning 32
5.8 Termineringsdialog 32

Indledning 1

Dette dokument beskriver det detaljerede SW-design for BROS, som er fastlagt ud fra dokumenterne kravspecifikation og systemarkitektur.

1.0.1 Formål

Formålet med dokumentet er:

- At fastlægge systemets detaljerede softwarestruktur udfra kravene specificeret i kravsspecifikationen. Derudover beskrivelsen af softwarekomponenterne og deres grænseflader beskrevet i systemarkitektur-dokumentet.
- At fastlægge systemets softwareklasser og deres indbyrdes interaktioner.
- At beskrive de enkelte klassers vigtigste metoder.

1.0.2 Reference dokumentation

- Kravspecifikation for projektet.
- Systemarkitektur-dokument.

Kontrolinterface 2

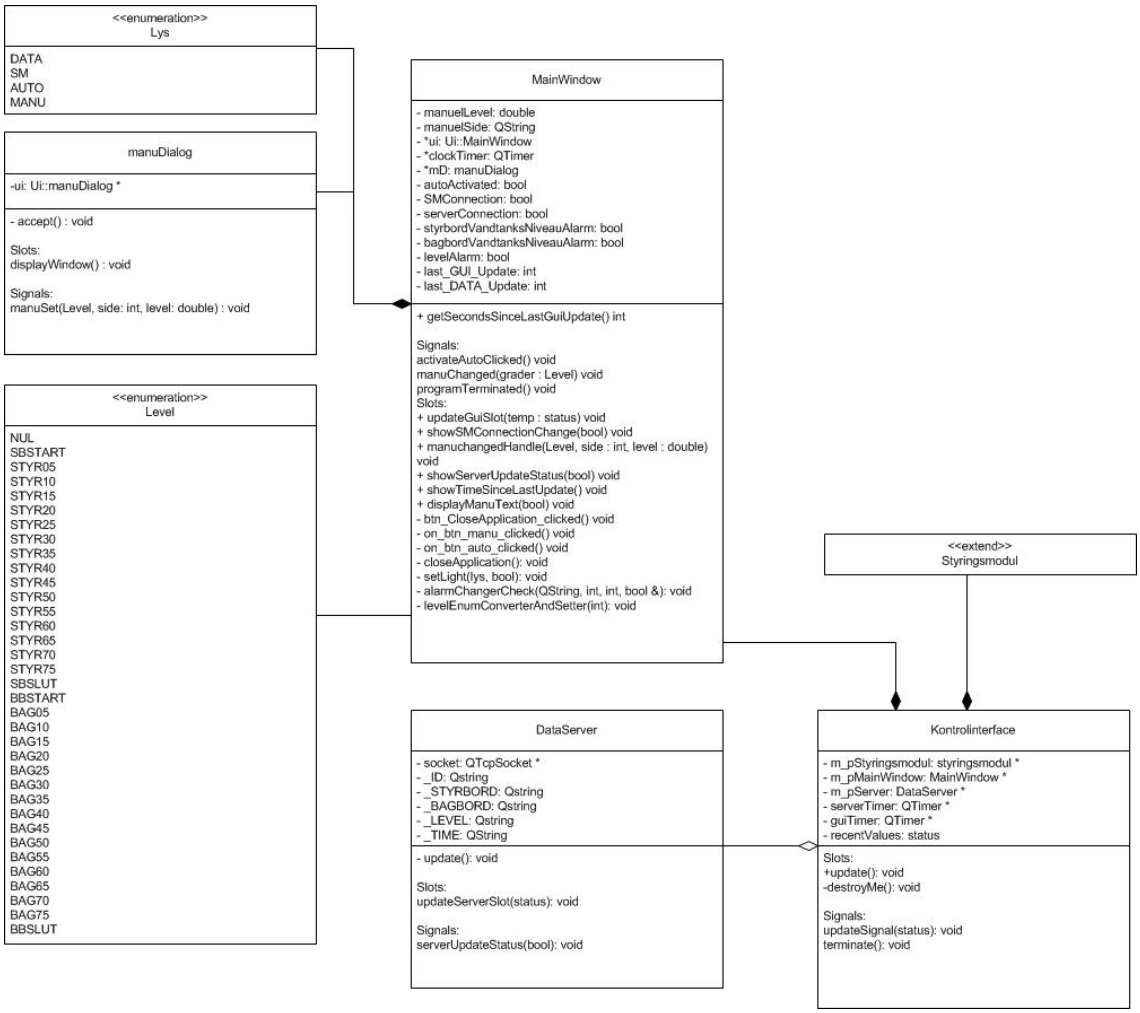
Nedenfor følger design af software til Kontrolinterfacet. Dette er lavet på baggrund af kravspecifikation og systemarkitektur.

2.0.3 Modulets ansvar

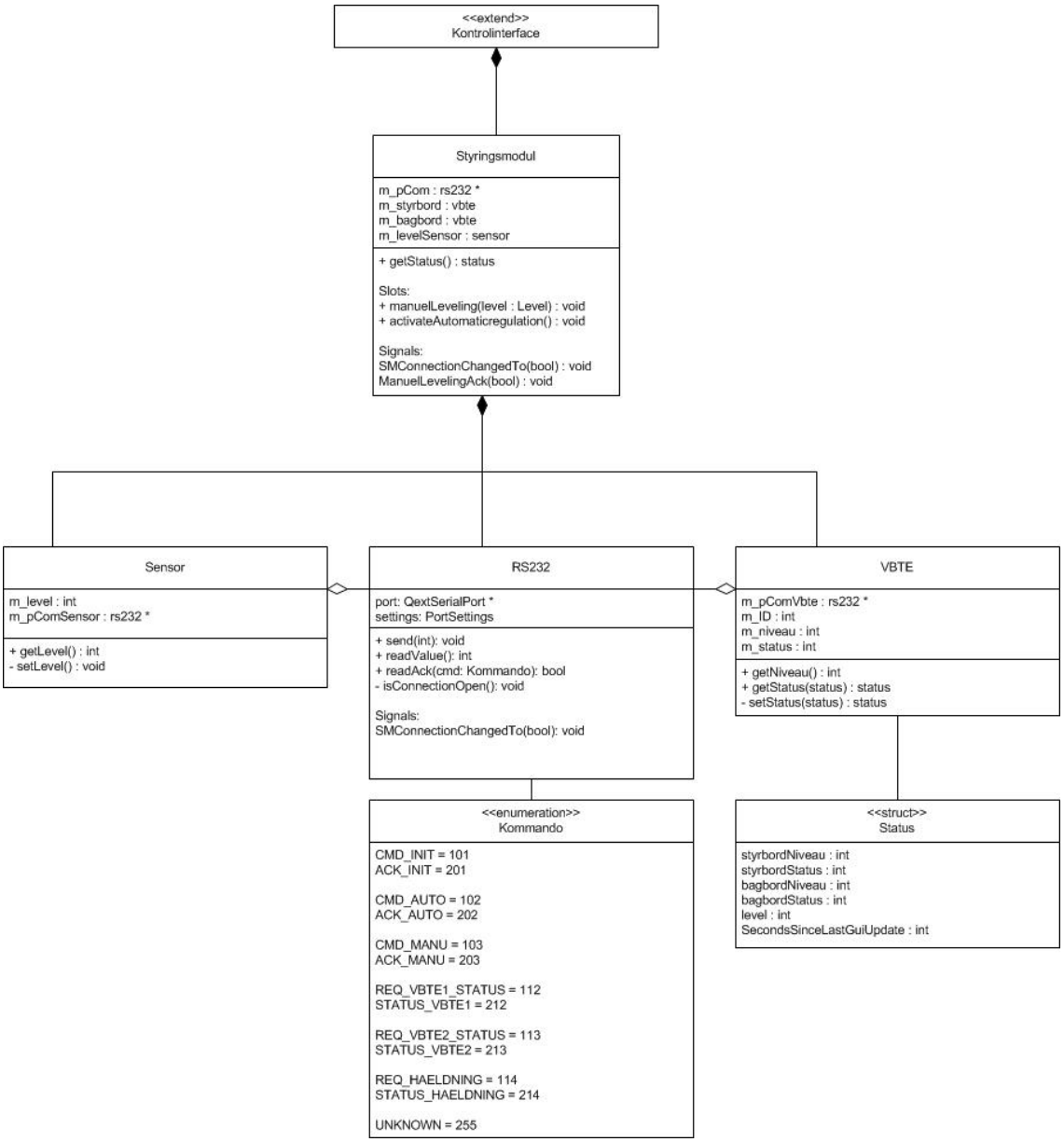
Kontrolinterfacet er brugerens primære kontaktflade til systemet. Programmet indeholder en brugergrænseflade der opfylder kravene i Kravspecifikationen. Her kan der også ses en prototype på den grafiske brugergrænseflade. Kontrolinterfacet står for at modtage inputs fra brugeren. Disse inputs sendes som kommandoer til Styringsmodulet. Det er også herfra at Kontrolinterfacet modtager de værdier, som sidenhen vises på den grafiske brugergrænseflade. Kontrolinterfacet står også for kommunikationen til den eksterne database. Her sendes en række parametre om skibet og dets status.

2.0.4 Klassediagram

Nedenfor ses klassediagrammet for Kontrolinterfacet. Bemærk at klassediagrammet er delt op i to. Skæringsstedet er mellem Kontrolinterface-klassen og Styringsmodul-klassen og er markeret med «extend».



Figur 2.1. På figuren ses klassediagrammet for KI - Kontrolinterface-delen



Figur 2.2. På figuren ses klassediagrammet for KI - Styringsmodul-delen

2.1 Metode- og klassebeskrivelser

2.1.1 MainWindow

Ansvar

Denne klasse indeholder de funktioner der er skrevet til Qt-formen MainWindow.ui hvori selve det grafiske er opbygget. Klassen indeholder de funktioner der anvendes i forbindelse med den grafiske brugergrænseflade. Det være sig når der kommer et input, eller der skal opdateres nogle værdier på skærmen.

Funktionsbeskrivelser

```
int getSecondsSinceLastGuiUpdate();
```

Beskrivelse:	En simpelt get-funktionen der returnerer værdien af klasseattributten *last_GUI_Update.*
Parametre:	Ingen
Returværdi:	`int secondsSinceLastGuiUpdate`

```
void updateGuiSlot(status temp);
```

Beskrivelse:	Bliver kaldt når GUI'en skal opdateres. Den modtager parameterent *temp* som er en struct af typen status. Ud fra denne struct hives de værdier ud, som skal vises på GUI'en. Værdierne vises ved de set-funktioner der er tilknyttet de anvendte widgets (og dermed en del af Qt-frameworket.) Når værdierne er opdateret vises det som en aktivitet i aktivitetsloggen
Parametre:	`status temp`
Returværdi:	Ingen

```
void showSMConnectionChange();
```

Beskrivelse:	Kaldes hvis SM-forbindelsen til styringsmodulet ændres fra forbundet til mistet forbindelse eller omvendt. Det udløser en aktivitet i aktivitetsloggen. Derudover skiftes lyset på gui'en. Parameteren state er den status som forbindelsen har ændret sig til.
Parametre:	`bool state`
Returværdi:	Ingen

```
void manuChangedHandle(Level samlet, int side, double level);
```

Beskrivelse:	Kaldes når brugeren har ændret i indstillingen til den manuelle hældning. Som parametre modtages hvilken side man ønsker at skibet skal hælde til (int side), hvor meget det skal hælde (double level) samt de to informationer samlet i en enum, Level samlet. Funktionen emitter signalet "manuChanged(Level temp). Det sætter også klasseattributerne manuelSide, manuelLevel samt autoActivated til deres rette værdier. Til sidst kaldes funktionen displayManuText(true)

Parametre:	`Level samlet`
	`int side`
	`double level`

Returværdi:	Ingen

```
void showServerUpdateStatus(bool state);
```

Beskrivelse:	Kaldes hver gang signalet DataServer::serverUpdateStatus() udsendes. Funktionen undersøger om forbindelsen har ændret sig ved at sammenligne med attributen serverConnection. Hvis forbindelsen har ændret sig udsendes dette som en aktivitet. Lyset ændres også således at det passer ved hjælp af setLight(DATA, serverConnection. Hvis vi modtager "true"vil last_DATA_Update opdateres til således til den nuværende værdi af sekunder siden epoch.
Parametre:	Ingen
Returværdi:	Ingen

```
void showTimeSinceLastUpdate();
```

Beskrivelse:	Kaldes hvert sekund. Funktionen opdaterer antallet af sekunder siden sidste overførelse af data til serveren eller til SM. Når tiden er længere end tiden mellem hver opdatering vil dette tal skifte til rødt.
Parametre:	Ingen
Returværdi:	Ingen

```
void displayManuText(bool show);
```

Beskrivelse:	Viser eller skjuler teksten med indstillingen af manuel hældning afhængig af parameteret show.
Parametre:	Ingen
Returværdi:	Ingen

```
void activateAutoClicked();
```

Beskrivelse:	udsendes når der er blevet trykket på knappen *activateAutoClicked*
Parametre:	Ingen
Returværdi:	Ingen

```
void activateAutoClicked();
```

Beskrivelse:	udsendes når der er blevet trykket på knappen *activateAutoClicked*
Parametre:	Ingen
Returværdi:	Ingen

```
void manuChanged(Level grader);
```

Beskrivelse:	Udsendes når der er blevet ændret en manuel indstilling.
Parametre:	`Level grader`
Returværdi:	Ingen

```
void programTerminated();
```

Beskrivelse:	Udsendes når programmet er blevet lukket ned.
Parametre:	Ingen
Returværdi:	Ingen

`void btn_CloseApplication_clicked();`

Beskrivelse:	Kaldes når luk-knappen på GUI'en er blevet trykket. Udsender signalet programTerminated() hvis brugeren bekræfter valget
Parametre:	Ingen
Returværdi:	Ingen

`void on_btn_manu_clicked();`

Beskrivelse:	Kaldes når der bliver trykket på knappen for manuel hældning. Viser dialogen "manuDialog".
Parametre:	Ingen
Returværdi:	Ingen

`void on_btn_auto_clicked();`

Beskrivelse:	Kaldes når der trykkes på *Automatisk Hældnings-knappen*. Aktiverer automatisk styring og deaktiverer den manuelle.
Parametre:	Ingen
Returværdi:	Ingen

`void setLight(lys id, bool state);`

Beskrivelse:	Sætter lyset i forhold til parametrene. lys er en enum der bestemmer hvilket element lyset skal ændres for. state er om lyset skal være tændt eller ej
Parametre:	`lys id` `bool state`
Returværdi:	Ingen

`void alarmChangerCheck(QString sentence, int critical_point, int value, bool &e`

Beskrivelse:	Tester om alarmerne har ændret sig. Sentence er starten af den sætning der skrives i aktivitetsloggen. critical_point er det kritiske punkt for det emne der arbejdes på. Value er den værdi den har. Earlier_state er hvilket stadie alarmen var i tidligere.

Parametre:	`QString sentence`
	`int critical_point`
	`int value`
	`bool &earlier_state`

Returværdi:	Ingen

`void levelEnumConverterAndSetter(int level);`

Beskrivelse:	Konverterer en integer baseret på Level-enumeratoren om til en side og en vinkel.
Parametre:	`int level`
Returværdi:	Ingen

2.1.2 Kontrolinterface

Ansvar

Dette er hovedklassen hvori selve programmet lever. Oprettelsen af et objekt af denne klasse er derfor også det eneste der sker i main.

Funktionsbeskrivelser

```
void update();
```

Beskrivelse:	Får en status-struct fra SM-klassen udfyldt med de nuværende værdier for systemet. Tilføjer antal sekunder siden sidste gui-update ved hjælp af getSecondsSinceLastGuiUpdate. Denne struct udsendes med signalet updateSignal(status recentValues)
Parametre:	Ingen
Returværdi:	Ingen

```
void updateSignal(status recentValues);
```

Beskrivelse:	Signalet der sendes når gui og server skal updateres. Indeholder en struct med alle relevante værdier.
Parametre:	`status recentValues`
Returværdi:	Ingen

```
void terminate();
```

Beskrivelse:	Udsendes når brugeren ønsker at terminere programmet.
Parametre:	Ingen
Returværdi:	Ingen

```
void destroyMe();
```

Beskrivelse:	Muliggør nedlæggelse af klassen med et funktionskald.
Parametre:	Ingen
Returværdi:	Ingen

2.1.3 DataServer

Ansvar

Klassen står for al kommunikation med serveren via en TCP-forbindelse. Forbindelse oprettes og nedlægges hver gang der tages kontakt. DataServer-objektet opdaterer serveren når den får ordre om det fra Kontrolinterface-klassen.

Funktionsbeskrivelser

`void onDelete();`	
Beskrivelse:	Kaldes i destruktoren. Sender en besked til serveren om at programmet termineres .
Parametre:	Ingen
Returværdi:	Ingen

`void updateServerSlot(status recentValues);`	
Beskrivelse:	Kaldes når signalet updateSignal(status recentValues) udsendes. Funktionen sørger så for at der via TCP-forbindelsen bliver udsendt de relevante værdier til databasen.
Parametre:	`status recentValues`
Returværdi:	Ingen

`void serverUpdateStatus(bool status);`	
Beskrivelse:	Udsendes når databasen er blevet opdateret. Parameteren "status"indikerer hvorvidt overførelsen var succesfuld eller ej. Der ventes ikke noget svar fra databasen.
Parametre:	`bool status`
Returværdi:	Ingen

2.1.4 manuDialog

Ansvar

manuDialog-klassen står for håndtering af det vindue der åbnes ved tryk på *Manuel Hældningsregulering-knappen.*

Funktionsbeskrivelser

`void manuSet(Level degrees, int side, double level);`	
Beskrivelse:	Udsendes i funktionen accept(). Der medsendes de data som brugeren har valgt på dialogen.

Parametre:	`Level degrees`
	`int side`
	`double level`

Returværdi:	Ingen

```
void accept();
```

Beskrivelse:	Kaldes når der trykkes på "OK-knappen på dialogen. Det indtastede omdannes til en værdi i forhold til enumeratoren "Level". Dialogen lukkes og signalet manuSet(...) udsendes

Parametre:	`Level degrees`
	`int side`
	`double level`

Returværdi:	Ingen

2.1.5 Styringsmodul

Ansvar

Klassen har samme rolle som det fysiske styringsmodul har i systemet. Det giver kontrolinterfacet adgang til sensor-værdier og vandstandsniveauer igennem sine underklasser, VBTE og Sensor.

Funktionsbeskrivelser

```
status getStatus();
```

Beskrivelse:	Indhenter værdierne for systemet fra VBTE'er og Hældningssensor. Disse værdier sættes ind i structen temp som så returneres.
Parametre:	Ingen
Returværdi:	`status recentValues`

```
void manuelLeveling(Level level);
```

Beskrivelse:	Sender kommando og vinkel til PSoC over RS232 vha. objektet m_pCom.
Parametre:	`Level level`
Returværdi:	Ingen

```
void activateAutomaticRegulation();
```

Beskrivelse:	Sætter hvorvidt automatisk regulering skal være aktiveret eller ej til styringsmodulet.
Parametre:	Ingen
Returværdi:	Ingen

```
void SMConnectionChangedTo(bool status);
```

Beskrivelse:	Udsendes hver gang der har været en overførelse. Status indikerer hvorvidt overførelsen var succesfuld eller ej.
Parametre:	`bool status`
Returværdi:	Ingen

```
void ManuelLevelingAck(bool status);
```

Beskrivelse:	Udsendes når der er blevet sendt kommandoen CMD_MANU til styrings-modulet. Bool status indikerer hvorvidt overførelsen var succesfuld ej.
Parametre:	`bool status`
Returværdi:	Ingen

2.1.6 VBTE

Ansvar

Håndterer værdierne for hver sin vandballasttankenhed. Kommunikerer til det fysiske styringsmodul ved hjælp af RS232-klassen.

Funktionsbeskrivelser

```
int getNiveau();
```

Beskrivelse:	Kalder setNiveau og returnerer værdien af niveau
Parametre:	Ingen
Returværdi:	int niveau()

```
status getStatus(status temp);
```

Beskrivelse:	Skaffer status vha. rs232-objektet og sætter niveau og status i structen temp som herefter returneres.
Parametre:	status temp()
Returværdi:	status temp()

2.1.7 Sensor

Ansvar

Håndterer værdierne for hældningssensoreren. Kommunikerer til det fysiske styringsmodul ved hjælp af RS232-klassen.

Funktionsbeskrivelser

```
int getLevel();
```

Beskrivelse:	Skaffer værdien af hældningen på skibet vha. rs232-objektet og returnerer det.
Parametre:	Ingen
Returværdi:	int level

2.1.8 RS232

Ansvar

Håndterer kommunikationen til det fysiske styringsmodul ved protokollen der ses i enumeratoren "Kommando".

Funktionsbeskrivelser

```
void send(int cmd);
```

Beskrivelse:	Sendefunktion. Sender den medsendte integer.
Parametre:	int cmd
Returværdi:	Ingen

```
int readValue();
```

Beskrivelse:	Modtagerfunktion. Returnerer den læste værdi.
Parametre:	Ingen
Returværdi:	int receivedValue

```
bool readAck(Kommando cmd);
```

Beskrivelse:	Kalder readValue() og sammenholder dennes returværdi med den værdi der er medsendt som parameter (cmd). Returnerer hvor vidt de to var identiske.
Parametre:	Kommando cmd
Returværdi:	bool status

```
void SMConnectionChangedTo(bool status);
```

Beskrivelse:	Når der har været en overførelse udsendes dette signal. Status indikerer hvorvidt overførelsen var succesfuld eller ej.
Parametre:	bool status
Returværdi:	Ingen

```
void SMConnectionChangedTo(bool status);
```

Beskrivelse:	Når der har været en overførelse udsendes dette signal. Status indikerer hvorvidt overførelsen var succesfuld eller ej.
Parametre:	bool status
Returværdi:	Ingen

```
bool isConnectionOpen();
```

Beskrivelse:	Tester om der er forbindelse til PSoC
Parametre:	Ingen
Returværdi:	bool connectionState

Databasen 3

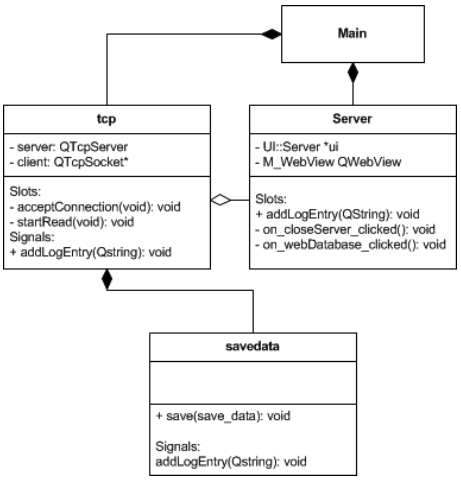
Nedenfor følger design af software til databasen og dens interface. Dette er lavet på baggrund af kravspecifikation og systemarkitektur.

3.0.9 Modultes Ansvar

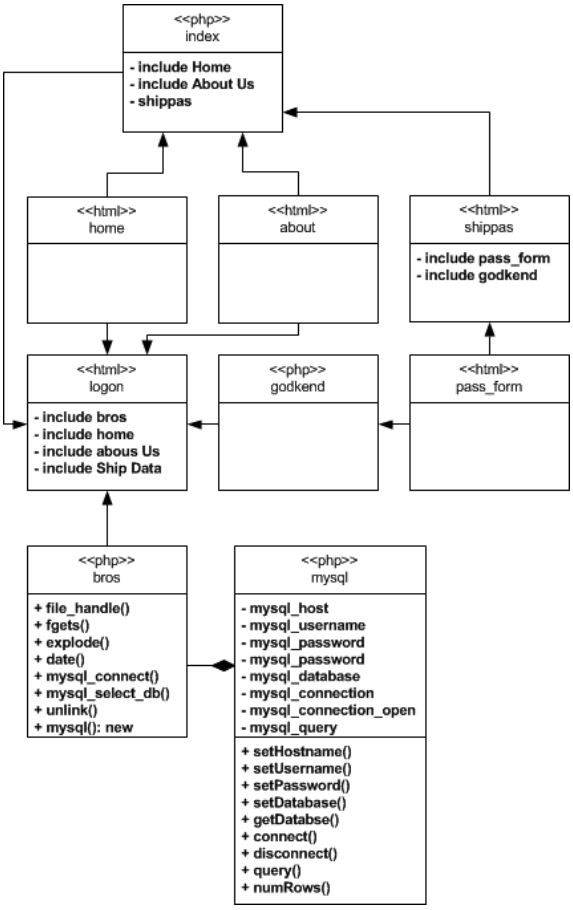
Databasen er her hvor havne terminalens personale kan aflæse data fra skibet. Disse data er sendt fra KI. Programmerne indeholder brugergrænseflader der opfylder kravene, beskrevet i kravspecifikationen. Her kan der også ses en prototyper på brugergrænsefladen. Database modulet har 3 dele. Severen, Web-siden og en mySQL database.
Severen står for kommunikationen immellem KI og Databasen. Severen modtager data fra KI og lager disse i en tekst fil
Web-siden giver brugeren mulighed for at se info om BROS samt at logge sig ind i BROS database hvorfra at data om skibe der er tilsluttet systemet kan aflæses. Web-sidens 3 vigtigste funktioner er at gemme ny data til mySQL databasen, slette den tekst fil som severen lavede og vise data for brugeren. For at håndtere web-sider der benytter sig af php (web-programmering) kræves der en web server som er i stand til at håndtere dette. En af de mest udbredte er apache serveren som også benyttes for denne webside. **mySQL databasen** er en database som er installeret på computeren. Alle data som er sendt fra KI er lageret i mySQL databasen.

3.0.10 Klassediagrammer

Nedenfor ses klassediagrammerne for databasen. Bemærk database modulet er lavet som en server del og en web del



Figur 3.1. Klassedigram for databasens severer



Figur 3.2. Klassedigram for databasens web-side.

1

[1] FiXme Note: check med kim, beskriv «html» og «php»

3.0.11 Globale variabler

3.0.12 Funktionsbeskrivelser

Server

Denne header står for at starte serveren og starte GUI for korte informationer til brugeren. Alle informationer om server start, connection o datamodtagelse vil blive udskrevet her.

`Void addLogEntry( QString )`

Står for at udskrive beskeder fra tcp klassen til GUI
Parametre: Ui::Server *ui;
Returværdi:

`Void on_closeServer_clocked( )`

Denne funktion håndtere luk knappen. Ved tryk knappen vil brugeren blive bedt om at svare ja eller nej til a
Parametre: Ingen
Returværdi:Ingen

`Void on_webDatabase_clicked( )`

Denne funktion står for at håndtere den direkte adgang til den web baserede database. Ved tryk vil brugeren
Parametre: QWebView* m_pWebView
Returværdi:Ingen

update

Denne header står for at håndtere de struct's der benyttes til at at save data til log filen.

`Void addLogEntry( QString )`

Står for at udskrive beskeder fra tcp klassen til GUI
Parametre: Ui::Server *ui;
Returværdi:

tcp

Denne header står for tcp forbindelsen. Socket oprettes og connection adgang gives. Når data bliver modtaget blvier denne gemt i den eksterne log fil ship.txt

`Void addLogEntry( QString )`

Står for at udskrive beskeder fra tcp klassen til GUI
Parametre: Ui::Server *ui;
Returværdi:

`Void acceptConnection( void )`

Beskrivelse:	Står for at acceptere forbindelse fra KI og connecte.
Parametre:	QTcpServer server
	QTcpSocket* client
Returværdi:	

`Void startRead( void )`

Beskrivelse:	Læser data fra socket. Data til fil og GUI
Parametre:	QTcpSocket* client
Returværdi:	

savedata

Denne header står for at handtere lagring af data modtaget fra skibet. Den lager dataerne i ship.txt

`int save( save_data )`

Står for at udskrive beskeder fra tcp klassen til GUI
Parametre:
Returværdi:

Wep-side

Web-siden står for at fremvise skibs data grafisk for terminal personalet. Desuden står den for at lager data og loade data fra mySQL databasen.

bros

`file_handle( )`

Beskrivelse:	Læse hvilken adresse databasen ligger på
Parametre:	
Returværdi:	

`fgets ( )`

Beskrivelse:	Læse hvilken adresse databasen ligger på
Parametre:	
Returværdi:	

`explode( )`

Beskrivelse:	Læse hvilken adresse databasen ligger på
Parametre:	
Returværdi:	

`date( )`

Beskrivelse:	Opdatere dato og tid når der gemmes til mySQL databasen
Parametre:	
Returværdi:	

`mysql_connect( )`

Beskrivelse: Læse hvilken adresse databasen ligger på
Parametre:
Returværdi:

`mysql_select_db( )`

Beskrivelse: Læse hvilken adresse databasen ligger på
Parametre:
Returværdi:

`unlink( )`

Beskrivelse: Læse hvilken adresse databasen ligger på
Parametre:
Returværdi:

`new mysql( )`

Beskrivelse: Læse hvilken adresse databasen ligger på
Parametre:
Returværdi:

mysql

`setHostName( )`

Beskrivelse: Læse hvilken adresse databasen ligger på
Parametre:
Returværdi:

`setUserName( )`

Beskrivelse: Skriver brugernavn til databasen
Parametre:
Returværdi:

`setPassword( )`

Beskrivelse: Skriver password til databasen
Parametre:
Returværdi:

`setDatabase( )`

Beskrivelse: Fortæller mySQL hvilken database der skal benyttes
Parametre:
Returværdi:

`getDatabase( )`
Beskrivelse:Tager fat i databasen
Parametre:
Returværdi:

`connect( )`	
Beskrivelse:	Står for at samle localhost, username, password, database og connecte til databasen
Parametre:	
Returværdi:	

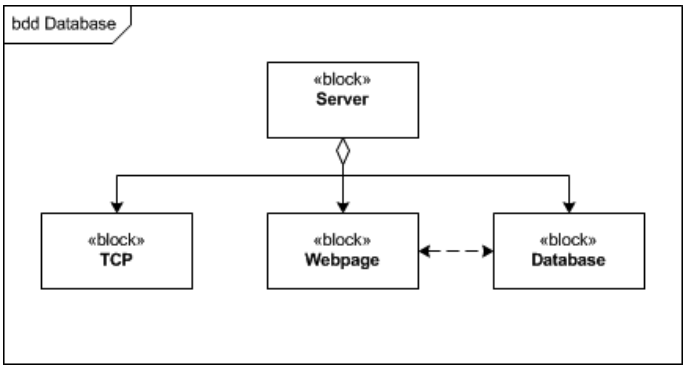
`disconnect( )`	
Beskrivelse:	Lukker database forbindelsen
Parametre:	
Returværdi:	

`query( )`	
Beskrivelse:	Opretter database kø og skriver data til skærm
Parametre:	
Returværdi:	

`numRows( )`	
Beskrivelse:	Checker hvor mange rækker der findes i databasen bruges desuden til at udskrive om databasen er tom.
Parametre:	
Returværdi:	

3.1 Design

3.1.1 Server



Figur 3.3. BDD server

SM 4

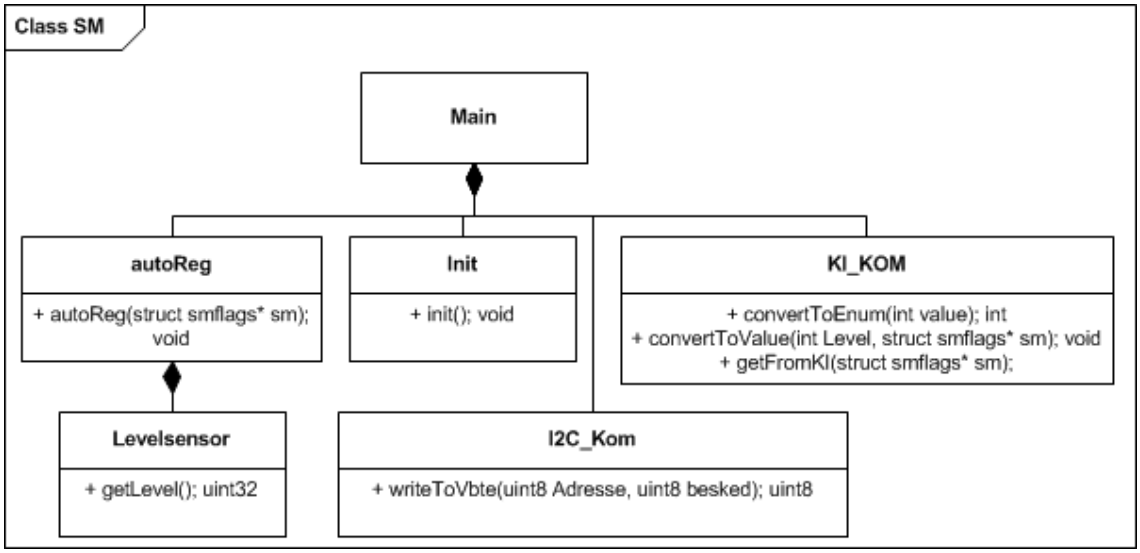
Dette afsnit beskriver designet af styringsmodulet, SM.

4.1 Klassens ansvar

Styringsmodulet har til ansvar at holde styr på levelsensoren og værdierne fra VBTE. Den kommunikere med KI og VBTE med indbyggede API'er fra Cypress PSoC 5 biblioteker.

4.2 Klassediagram

Nedenfor ses klassediagrammet for SM. Bemærk at koden dog er i C men for overblikket er der lavet klassediagram.



Figur 4.1. På figuren ses klassediagrammet for SM

4.3 Variabler

Variabel	Beskrivelse
`autoflag`	Denne variable er et flag der holder styr på automatisk regulering .
`manuflag`	Et flag til at holde styr på manuel regulering.
`levelVal`	En variable med vores level værdi.
`VBTE1Niveau og VBTE2Niveau`	Holder styr på vandniveauet i ballasttanke i %.
`VBTE1Status og VBTE2Status`	Holder styr op tilgængelighed for VBTE1 og 2.
`vinkelVal`	Indeholder værdien for manuel regulering.

Alle variabler er indkapslet i en struct navngivet "smflags".

4.4 Funktionsbeskrivelser

4.4.1 Init

Ansvar

Denne header har til ansvar at sørge for alle komponenter opretter og initieret. `void init( void);`

Beskrivelse:	Funktionen anvender API'et fra Cypress componenter og står for at initiere og starte vores PSoC hardware. Den sætter også et register tilhørende vores Accelerometer.
Parametre:	ingen
Returværdi:	ingen

4.4.2 Levelsensor

Ansvar

Denne header har til ansvar at hente levelværdien ind fra vores accelerometer. `uint32 getLevel( void);`

Beskrivelse:	Funktionen anvender API'et fra Cypress componenter og venter på at vores ADC henter convertere det analoge signal. Funktionskald for ADC ses i PSoC databladet.
Parametre:	ingen
Returværdi:	`uint32 levelVal`

4.4.3 autoReg

Ansvar

Denne header har til ansvar at styre automatisk regulering. `void autoReg( struct smflags* sm);`

Beskrivelse:	autoReg anvender værdier fra VBTE moduler samt KI til at holde systemet i et bestemt level. Funktionen starter med at checke på automatisk og manuel styrings flagene. Derefter kalder den getLevel agere ud fra niveauet. Funktionen vil altid tømme fra en tank før den begynder at fylde en anden.
Parametre:	`struct smflags* sm`
Returværdi:	ingen

4.4.4 I2C_Kom

Ansvar

Denne header har til ansvar at kommunikere med VBTE modulerne. `uint8 writeToVbte( uint8 Adresse, uint8 besked);`

Beskrivelse:	writeToVbte anvender I2C fra Cypress PSoC 5 API til at skrive til VBTE modulerne. Den tager adressen og beskeden man skal sende og sender til pågældende enhed. Derefter venter den på svar som den så returnere.
Parametre:	`uint8 Adresse` `uint8 besked`
Returværdi:	`uint8 VbteNiveau`

4.4.5 KI_KOM

Ansvar

Denne header har til ansvar at kommunikere med KI enheden.

```
int convertToEnum( int value);
```

Beskrivelse:	funktionen tager en level værdi ind for så at konvertere den til en Enum(integer) som den returnere.
Parametre:	`int value`
Returværdi:	`int Enum`

```
void convertToValue(int Level, struct smflags* sm);
```

Beskrivelse:	Funktionen tager en enum og en pointer som den så konvertere til en level værdi og sætter i sm structen.
Parametre:	`int Level,` `struct smflags* sm`
Returværdi:	ingen

```
void getFromKI( struct smflags* sm);
```

Beskrivelse:	Funktionen anvender UART fra Cypress PSoC 5 API'en til at modtage en besked fra KI modulet som den så vurdere og agere på. Når den har modtaget noget sender den en ack tilbage til KI modulet. Derefter handler den og hvis det er nødvendigt sender data til KI.
Parametre:	`struct smflags* sm`
Returværdi:	ingen

4.5 Eventuelle Sekvensdiagrammer og state machines

Måske kommer de senere?

VBTE 5

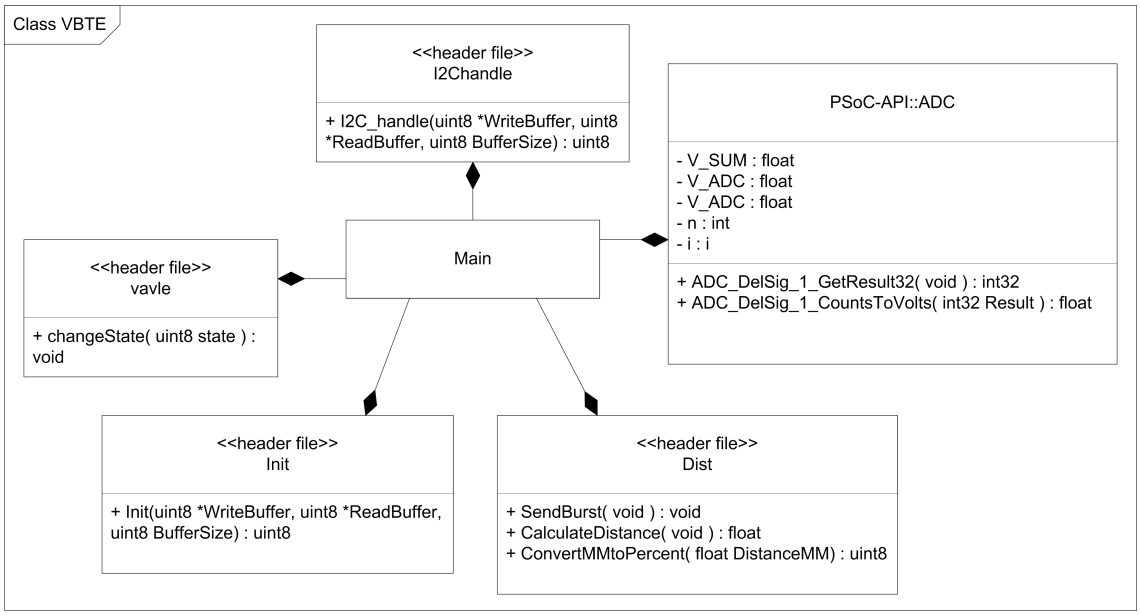
Nedenfor følger design af software til VBTE. Dette er lavet på baggrund af kravspecifikation og systemarkitektur. Bemærk der i dette design dokument blandt andet ikke er beskrevet mixer, pga osv. da deres eneste funktion er "Start();". Derudover er der en betydelig hardware del knyttet til dette modul, der refereres derfor til detaljeret hardware design for yderligere detaljer om VBTE modulet.

5.1 Modulets ansvar

Som beskrevet i systemarkitektur står VBTE'en for at måle vandniveauet i ballasttankene samt at lukke vand ind eller ud af ballasttankene. Hertil er der også en kommunikation med SM modulet indeholende instruktioner.

5.2 Klassediagram

Nedenfor ses klassediagrammet for VBTE. Bemærk at koden dog er i C men for overblikket er der lavet klassediagram.



Figur 5.1. På figuren ses klassediagrammet for VBTE **Billedet skal opdateres**

5.3 Globale variabler

Variabel	Beskrivelse
`BurstLengthVal`	Denne variabel er anvendt til at håndtere antallet af perioder burstet bliver sendt med.
`WaitBurstVar`	Bliver brugt til nonblocking delay til SendBurst funktionen.
`BurstTimerVal`	Holder på Timerens værdi når et burst er sendt.
`DistanceTimerVal`	Holder værdien på timeren når et burst er modtaget.
`CalcDistFlag`	Bliver sat når et burst er modtaget så en afstand kan blive beregnet.
`BurstFlag`	Bliver sat når et burst bliver sendt og hevet ned når et burst er modtaget. Dette sker for ikke at få flere detektioner på samme signal.

5.4 Metode- og klassebeskrivelser

5.4.1 Init

Ansvar

Denne header har til ansvar at initiere alle moduler og blokke på PSoC'en. Disse funktioner hentes fra PSoC'ens API.

Funktionsbeskrivelser

```
void Init( uint8* WriteBuffer, uint8* ReadBuffer, uint8 BufferSize )
```

Beskrivelse:	Funktionen anvender API'et fra alle PSoC blokke anvendt i designet og kalder deres start funktion. Derudover initierer den også read- og writebuffer til I2C modulet.

Parametre:	`uint8* WriteBuffer`
	`uint8* ReadBuffer`
	`uint8 BufferSize`

Returværdi:	Ingen

5.4.2 Valve

Ansvar

Denne header har til ansvar at styre ventilerne ud fra "state-variablen modtaget fra I2C_handle. Headeren benytter PSoC-API'et til kontrol registre..

Funktionsbeskrivelser

`void ChangeState( uint8 state )`

Beskrivelse:	Funktionen modtager state som indeholder indformationer om ventilerne skal være lukkede eller hvilken ventil der skal være åben. Den benytter PSoC5 API'et for kontrol register.
Parametre:	`uint8 state`
Returværdi:	Ingen

5.4.3 Dist

Ansvar

Denne header har til ansvar at sende burst, beregne afstanden samt at omregne afstanden til procent.

Funktionsbeskrivelser

`void SendBurst( void )`

Beskrivelse:	Denne metode aktiverer en 40kHz clock og tæller perioderne op til 10, lukker for burstet og ligger timerens værdi ind i den globale variabel BurstTimerVal. Herefter sættes BurstFlag'et.
Parametre:	Ingen
Returværdi:	Ingen

`float CalculateDistance( void )`

Beskrivelse:	Denne metode anvender BurstTimerVal og DistanceTimerVal til at finde ud af hvor mange clocks der er gået fra burstet er blevet sendt til det igen er blevet registreret.
Parametre:	Ingen
Returværdi:	`float DistanceMM`

`uint8 ConvertMMtoPercent( float )`

Beskrivelse:	Metoden modtager afstanden i millimeter og returnerer hvor mange % der er i tanken
Parametre:	`float DistanceMM`
Returværdi:	`uint8 DistancePercent`

5.4.4 I2Chandle

Ansvar

Denne header har til ansvar at håndtere I2C. Den kigger om der er kommet noget i writebufferen. Er der kommet noget i bufferen kigger den efter hvilket tilstand det er der skal ændres til og så smider den afstanden i procent i read bufferen.

Funktionsbeskrivelser

```
uint8 I2C_handle( uint8* WriteBuffer, uint8* ReadBuffer, uint8 BufferSize )
```

Beskrivelse:	Funktionen anvender API'et fra I2C blokken i PSoC miljøet. Med disse tjekker den om der er fyldt nyt i bufferen og aflæse dette. Herfer kalder den funktionen I2C_decode til at afkode beskeden fra SM. Herefter klargøres readbufferen til at sende vandniveau tilbage.

Parametre:	`uint8* WriteBuffer`
	`uint8* ReadBuffer`
	`uint8 BufferSize`

Returværdi:	`uint8 State`

5.4.5 PSoC-API::ADC

Ansvar

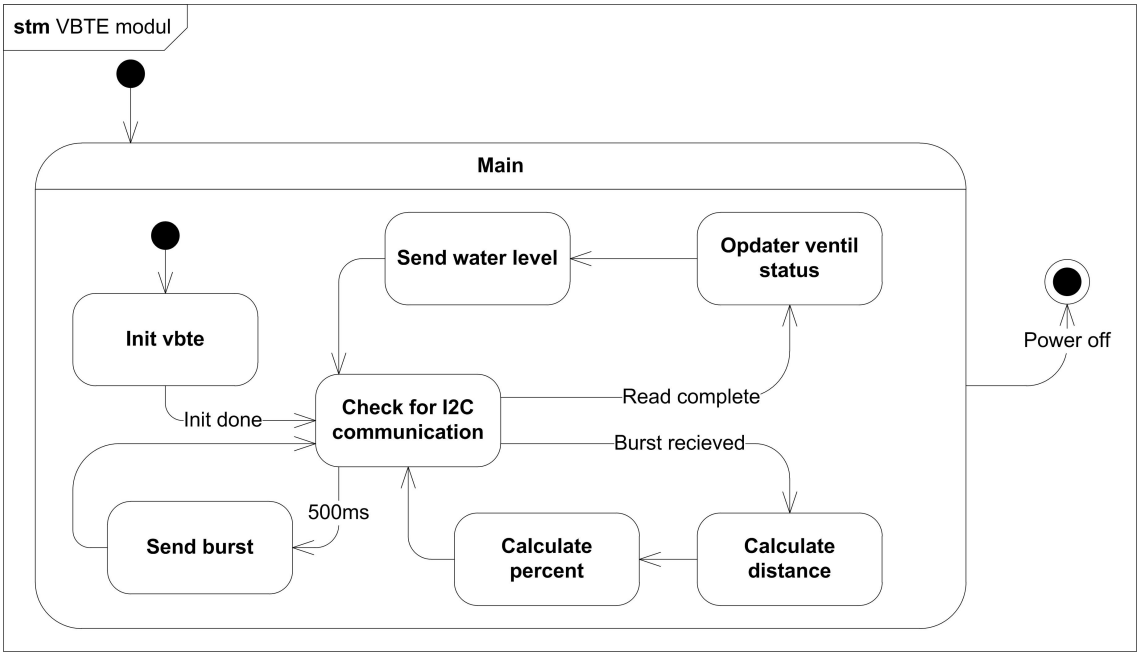
Denne header er kun beskrevet fordi der er implementeret en funktionalitet i dette API. Der er under ADC

Beskrivelse

Inde i ADC'ens interrupt header tilføjes funktionalitet så der, hver gang der bliver samplet, bliver valideret på om der er en detektion. Er der en detektion sættes flaget til udregning af afstand samt flaget for burst sættes til 0 igen. For at gøre dette anvendes funktionerne fra API'et for ADC'en.

5.4.6 State Machine

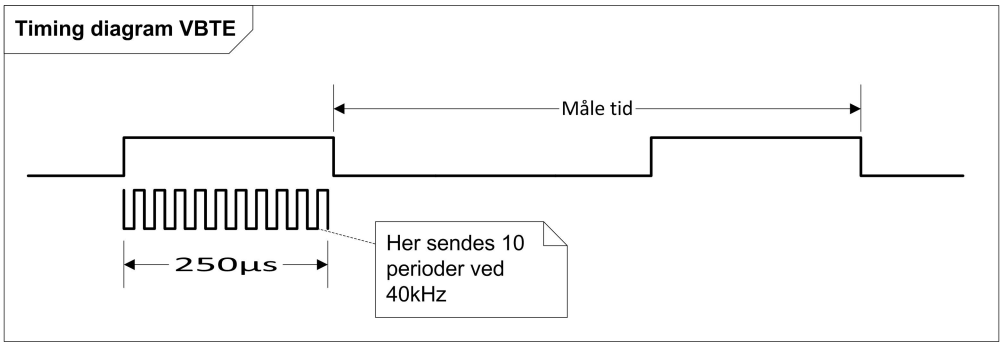
Nedenfor ses statemachine der beskriver det overordnede flow i VBTE programmet.



Figur 5.2. Statemachine for VBTE program

5.4.7 Timing Diagram

Nedenfor ses timing diagram for en ultralydspuls til afstandsmåling



Figur 5.3. Timing diagram for VBTE ultralydspuls

5.4.8 Interrupt rutiner

I dette afsnit beskrives interrupt rutinerne i VBTE programmet.

isr_dist

Isr_dist har til ansvar at tælle den globale variabel `WaitBurstVar` op. Den bliver triggeret af en clock på 1kHz. Hver gang variablen havner over 500 bliver `SendBurst();` funktionen kaldt og variablen bliver nulstillet.

isr_counter

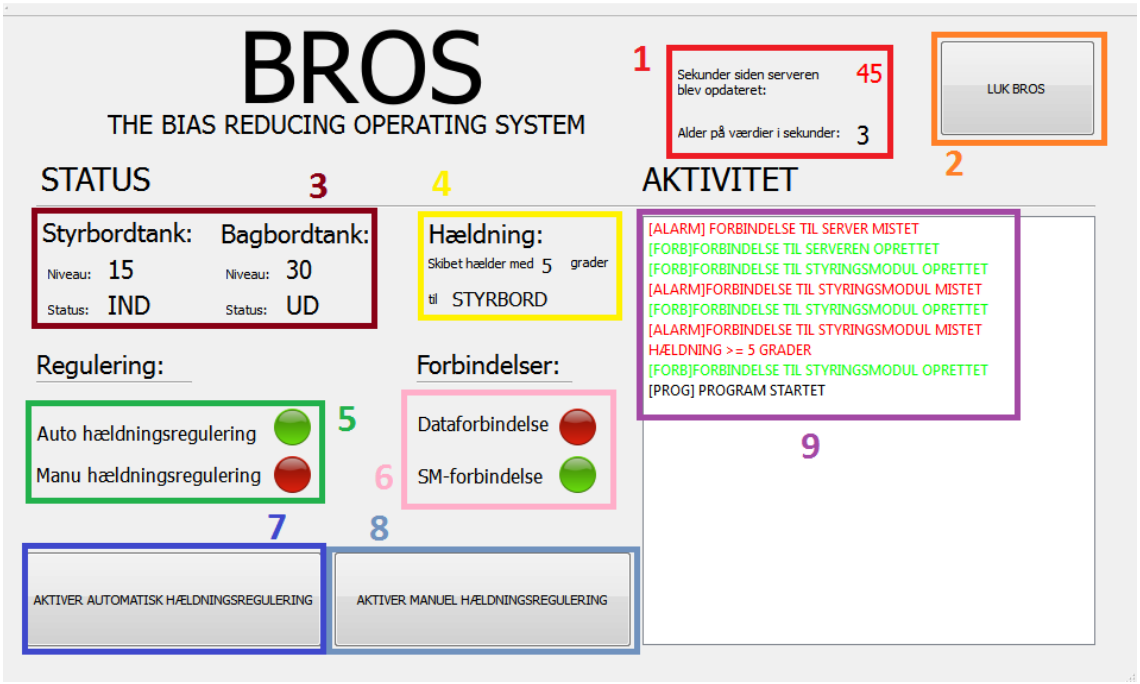
Isr_counter tæller variablen `BurstLengthVal` op. Denne anvendes til at styre at der kun bliver sendt 10 peroder i et burst.

Appendix A: Kontrolinterface

I dette appendix vil jeg gå nærmere ind på opbygningen af den grafiske brugergrænseflade på Kontrolinterfacet.

5.5 Hovedvindue

Det første vindue man ser ved programopstart er hovedvinduet, som vist på



Figur 5.4. På figuren ses hovedvinduet for Kontrolinterface-programmet

Hovedvinduets elementer	
1: Forsinkelse i sekunder	Det øverste tal fortæller tiden i sekunder siden serveren sidst er ble opdateret succesfuldt. Nedenunder udskrives tiden i sekunder si værdierne i boks tre og fire er blevet opdateret.
2: Nedlukningsknap	Anvendes til at lukke programmet. Programmet åbner dialogen som s **??**
3: Vandballasttankene	Her kan status for vandballasttankene aflæses. Niveauet er hvor fyldt tan er angivet i procent. Hvis niveauet er over 70% skrives tallet med rø Status angiver vandets flow i tanken: IND/UD/LUKKET.
4: Hældningssensor	Værdien for hældningen af skibet angives i antal grader og i hvilken s skibet hælder.
5: Reguleringsstatus	Her angives hvorvidt automatisk eller manuel hældningsregulering aktiveret. Der vil altid kun være en og kun en af disse aktiveret. De vil der altid være en rød og en grøn indikator tændt. I dette eksempe den automatisk hældningsregulering aktiveret.
6: Forbindelser	Indikerer hvorvidt der er forbindelse til Styringsmodulet og serve Dataforbindelse er rød hvis det ikke lykkedes at oprette forbindelse serveren ved sidste forsøg. SM-forbindelse er rød hvis det ikke lykkedes få de ventede svar fra Styringsmodulet. I denne situation er der forbind til styringsmodulet, men ikke serveren.
7: Automatisk reguleringsknap	Ved tryk på denne knap vil man komme til dialogen på **??** såfr automatisk styring ikke er aktiveret. Hvis den er aktiveret og man tryk på knappen vil dialogen på figur **??** fremkomme.
8: Manuel reguleringsknap	Bringer dig til dialogen på figur **??**
9: Aktivitetslog	Her udskrives vigtige hændelser i programmet med farvekoder. I d eksempel kan det ses hvordan alarmer skrives med rødt og oprett forbindelser skrives med grønt.

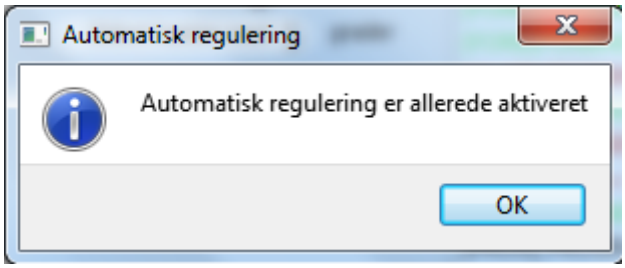
5.6 Manuel regulering af hældning

Ved tryk på knappen med teksten "AKTIVER MANUEL HÆLDNINGSREGULERING"(boks otte på figur 5.6) kommer dialogen på figur **??**

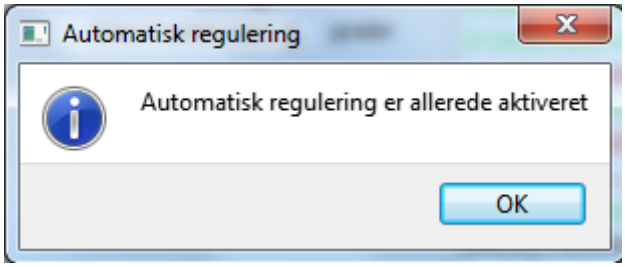
Figur 5.5. På figuren ses vinduet for manuel indstilling af vinkel

5.7 Automatisk regulering af hældning

Ved tryk på knappen med teksten "AKTIVER AUTOMATISK HÆLDNINGSREGULERING"(boks syv på figur 5.6) kommer en dialog frem. Såfremt automatisk hældnings allerede er aktiveret (som indikeret med en grøn cirkel øverst i boks fem på figur 5.6) fremkommer dialogen på figur **??** frem. Hvis automatisk hældning ikke er aktiveret fremkommer dialogen på figur **??** frem.



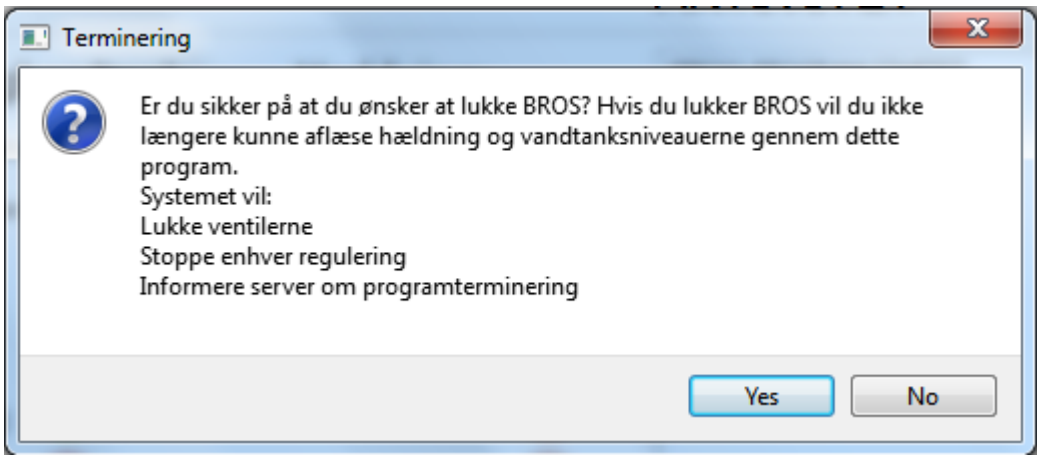
Figur 5.6. Ved tryk på AKTIVER AUTOMATISK HÆLDNINGSREGULERING når automatisk hældningsregulering allerede er aktiveret



Figur 5.7. Ved tryk på knappen i boks syv på figur 5.6 når automatisk hældningsregulering ikke er aktiveret

5.8 Termineringsdialog

Denne advarsel fremkommer når man trykker på knappen i boks to på figur 5.6.



Figur 5.8. Advarsel før nedlukning af BROS

Rettelser

Note: check med kim, beskriv «html» og «php» 17