

AARHUS UNIVERSITY SCHOOL OF ENGINEERING

AUTONOM FORFØLGELSESDRONE MED SMARTPHONE APPLIKATION

BACHELORPROJEKT
ELEKTROINGENIØR

Arkitektur & Design

Projekt nr: 16114

Jonas Risager Nielsen - 201270337

Benedikt Wiese - 201310362

Mathias Poulsen - 201370945

Vejleder

Torben Gregersen

Aarhus University School of Engineering

15. december 2016

Versionshistorie for Systemarkitektur

Version	Dato	Beskrivelse
0.0	06.09.2016	Opsætning af dokumentet.
0.1	06.09.2016	Tilføjet domænemodel for det samlede system.
0.1.1	06.09.2016	Tilføjet domænemodel for dronen.
0.1.2	07.09.2016	Tilføjet BDD og IBD.
0.1.3	09.09.2016	Tilføjet Printdesign og Signal oversigt + Mindre rettelser til domænemodellerne.
0.1.4	09.09.2016	Rettet efter møde i; Domænemodeller, BDD, IBD, Signal oversigt.
0.1.5	19.09.2016	Tilføjet UC1-2 for server Logical view og UC1-2+7-8 for drone Logical view.
0.1.6	19.09.2016	Tilføjet Implementation view for dronen.
0.1.7	29.09.2016	Tilføjet Transmitter/Receiver setup i drone Impl view. Rettet server logical view til.
0.1.8	03.10.2016	Tilføjet Process- Data- Deployment- Security- og Implementation view for serveren.
0.1.9	06.10.2016	Rettet Logical view for dronen til når slutningen af sprint 2.
0.1.10	06.10.2016	Logical view tilføjet for applikation.
0.2	07.10.2016	Endt sprint 2.
0.2.1	11.10.2016	Logical view tilføjet for serveren UC 3 + 6. Logical view tilføjet for dronen UC 3+6 og 9+11.
0.2.2	14.10.2016	Logical view tilføjet for drone UC 3, 6 9 og 11.
0.2.3	24.10.2016	Tilføjet Arduino Nano til domænemodel, BDD og IBD.
0.2.4	26.10.2016	Logical view rettet for dronen i forhold til sprint 3.
0.2.5	26.10.2016	Process view tilføjet for dronens nuværende tråde.
0.2.6	27.10.2016	Rettet logical view for serveren efter ny feature.
0.2.7	27.10.2016	Implementation view for systemets Raspberry Pi.
0.3	27.10.2016	Endt sprint 3.

0.3.1	01.11.2016	Domænemodeller, BDD og IBD ændret grundet vejledermøde.
0.3.2	01.11.2016	Serverens Logical view til UC 4,5 og 10 tilføjet.
0.3.3	01.11.2016	Udkast til dronens Logical View UC 4,5 og 10.
0.3.4	04.11.2016	Process view rettet til efter ændringer i dronens tråde.
0.3.5	04.11.2016	Udkast til applikationens Logical View UC 4,5 og 10.
0.3.6	08.11.2016	Tilføjet GPS og Bluetooth i Deployment view.
0.3.7	08.11.2016	Ændret Logical, Data, Implementation view for serveren grundet ny feature.
0.3.8	09.11.2016	Domænemodeller, BDD og IBD ændret grundet vejledermøde.
0.3.9	17.11.2016	Logical view rettet for dronen i forhold til sprint 4.
0.3.10	18.11.2016	Tilføjet følgende afsnit til dronens implementation view: Seriel kommunikation til FlightControlleren og Nano'en. GPS samt tilt kompensation.
0.3.11	18.11.2016	Tilføjet følgende afsnit til dronens implementation view: Regulering og funktioner. Derudover er der ændret i afsnittet omkring biblioteker.
0.3.12	18.11.2016	Opdateret Logical View for applikationen, Implementation-, Data- og Process view for applikationen tilføjet.
0.4	18.11.2016	Endt sprint 4.
0.4.1	21.11.2016	Tilføjet stop signal beskrivelse i security view.
0.4.2	22.11.2016	Tilføjet Communicator i implementation view.
0.5.0	05.12.2016	Ordforklaring tilføjet.
0.5.1	07.12.2016	Revideret.
1.0	15.12.2016	Aflevering.

Ordforklaring

Forkortelse	Forklaring
3G, ISP, Internet	Disse har samme betydning i dette dokument. De bliver alle brugt til at beskrive internetforbindelsen.
3S	3-cellet batteri.
Ω	Ohm.
A	Ampere.
API	Application Programming Interface.
ASE	Aarhus University School of Engineering.
BDD	Block Definition Diagram.
BLE	Bluetooth Low Energy.
CPU	Central Processing Unit.
DC	Direct Current.
Eavesdropping	At lytte med på kommunikationen uden tilladelse.
ESC	Electronic Speed Controller.
GND	Stelforbindelse.
GPIO	General-purpose input/output.
GPS	Global Positioning System.
HTML	Hyper Text Markup Language.
HTTP	HyperText Transfer Protocol.
HTTPS	HyperText Transfer Protocol Secure.
I2C	Inter-Integrated Circuit.
IBD	Internal Block Diagram.
IN	Input.
LED	Light Emitting Diode.
LiPo	Lithium Polymer Battery.

LoS	Line of Sight.
IOT	Internet of things.
IP	Internet Protokol.
Man-in-the-middel Attack	Et angrab på kommunikationen hvor beskeden i hemmelighed retransmitterer beskeden og muligvis ændre på beskeden.
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor.
OUT	Output.
P-led	Proportional-led.
PC	Personal Computer.
PID	Proportional–integral–derivative.
PWM	Pulse-width modulation.
RGB	Red, Green, Blue.
RPi	Raspberry Pi 3 Model B.
Shadow BAC-X1	Navnet på dronen, der udvikles i dette projekt.
SQL	Structured Query Language.
SSH	Secure Shell.
TCP	Transmission Control Protocol.
UART	universal asynchronous receiver/transmitter.
URL	Uniform Resource Locator.
USB	Universal serial bus.
V	Volt.
XML	Extensible Markup Language.

Indholdsfortegnelse

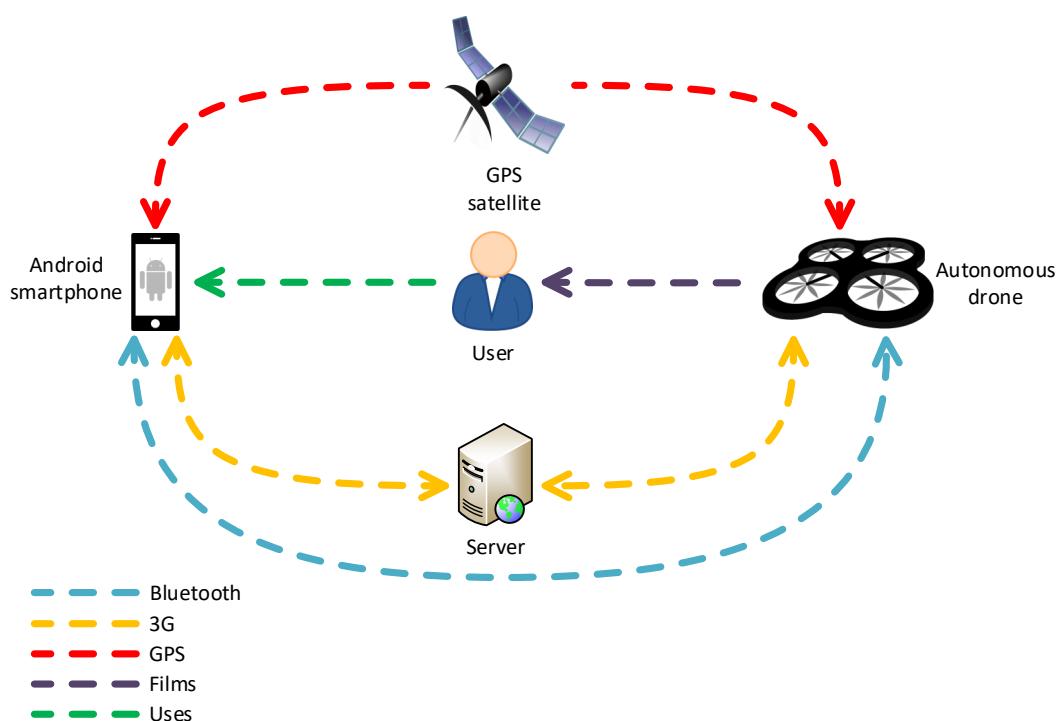
Kapitel 1 Arkitektur og Design - Shadow BAC-X1	1
1.1 Domænemodeller	2
1.1.1 Domænemodel - Samlet system	2
1.1.2 Domænemodel - Drone	3
Kapitel 2 Hardware arkitektur og design	5
2.1 Block definitions diagram	5
2.1.1 Server	6
2.1.2 Smartphone	6
2.1.3 Drone	6
2.2 Internal block diagram	8
2.3 Signal oversigt	9
2.3.1 Smartphone	9
2.3.2 Server	9
2.3.3 Drone	10
2.4 Print design	17
2.4.1 Droneinterface	17
2.4.2 Stepmotor	18
2.4.3 Spændingsregulator	19
2.5 Stykliste	20
2.6 Mekanisk Design	21
2.6.1 Dronens opbygning	21
2.6.2 Mekanisk design af kamera	22
2.7 Implementering	23
2.7.1 Print implementering	23
2.7.2 AeroQuad Cyclone stel	24
2.7.3 Batteri	24
Kapitel 3 Software arkitektur og design	25
3.1 N+1 View	25
3.2 Logical View	26
3.2.1 System sekvensdiagram	26
3.2.2 Server	28
3.2.3 Drone	35
3.2.4 Applikation	64
3.3 Process View	85
3.3.1 Server	85
3.3.2 Drone	85
3.3.3 Applikation	89
3.4 Data View	92

3.4.1	Server	92
3.4.2	Drone	94
3.4.3	Applikation	94
3.5	Deployment View	95
3.5.1	Besked struktur	95
3.5.2	Kommunikation mellem systemets blokke	96
3.6	Security View	98
3.6.1	Website sikkerhed	98
3.6.2	Login	98
3.6.3	Dronen	99
3.7	Implementation View	100
3.7.1	Server	100
3.7.2	Drone	103
3.7.3	DroneControllerUnit	107
3.7.4	Application	121

Arkitektur og Design - Shadow BAC-X1

1

Dette afsnit viser arkitekturen og designet i det samlede system. På figur 1.1 ses en oversigt over systemet og hvordan de forskellige elementer forbinder til hinanden.



Figur 1.1: System Oversigt

Det ses på figur 1.1 at brugeren benytter applikationen, som sørger for kommunikationen imellem brugeren og dronen enten via 3G eller Bluetooth. Både applikationen og dronen benytter GPS systemet.

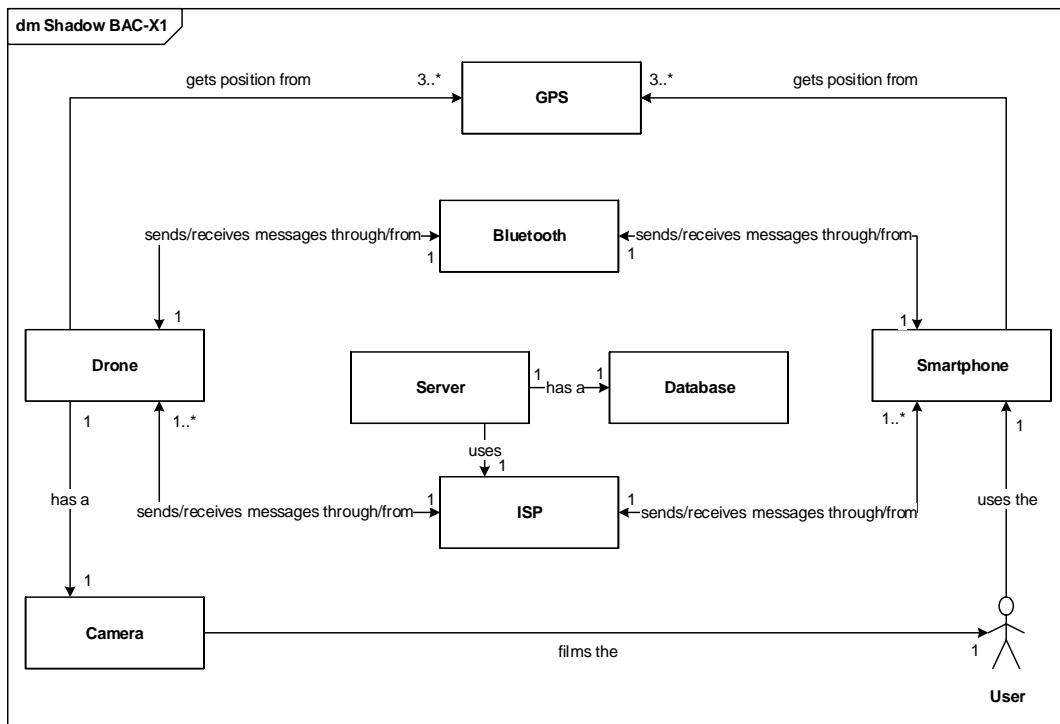
Yderligere beskrivelse af systemet kan findes i afsnit 1.2 Systembeskrivelse og i afsnit 1.3 Systemoversigt i Kravspecifikation i bilag.

1.1 Domænemodeller

Domænemodeller kan betragtes som bindeleddet mellem kravspecifikationen og dette arkitektur- og designdokument. Domænemodeller tilbyder et overblik over systemet i en generel form og i mindre blokke. I dette projekt er der udarbejdet to domænemodeller henholdsvis for det samlede system og dronen.

1.1.1 Domænemodel - Samlet system

På figur 1.2 ses systemets overordnede domænemodel. Domænemodellen kan senere bruges til at danne grundlag for systemets applikationmodeller.

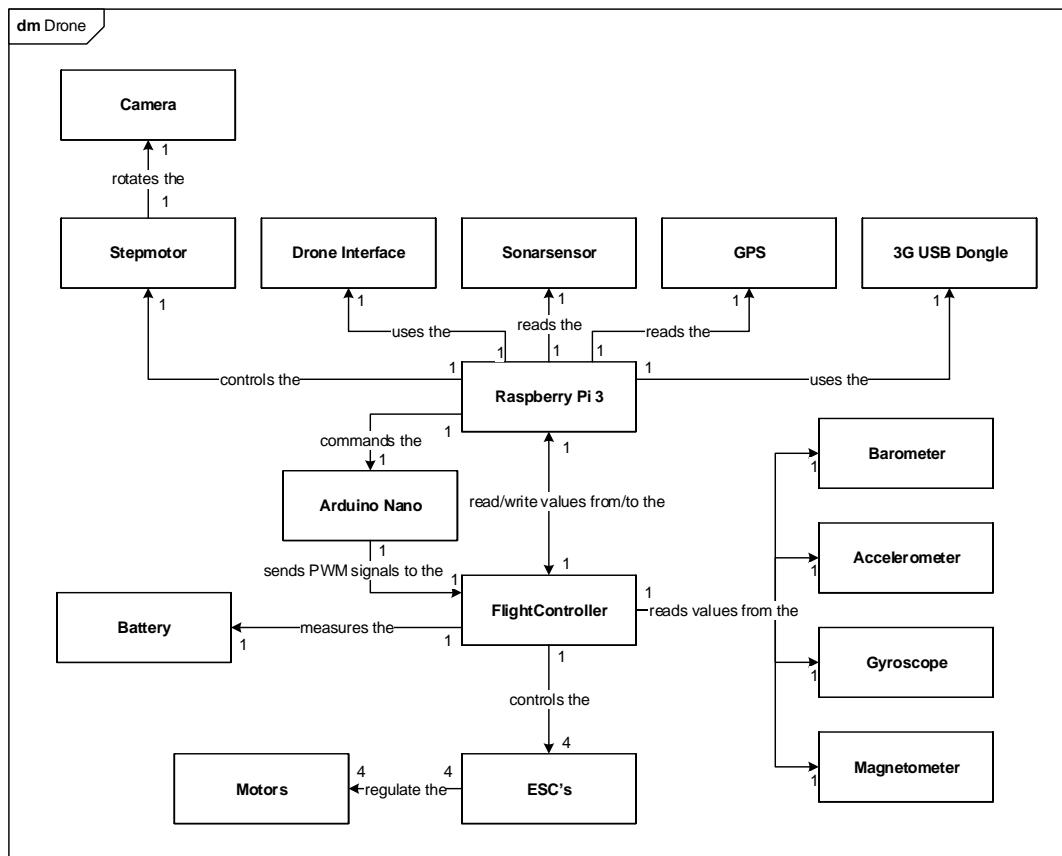


Figur 1.2: Domænemodel - Samlet system

Brugeren benytter applikationen på mobiltelefonen til at sende beskeder til dronen enten gennem Bluetooth eller ISP. ISP'en benytter systemets server, når der skal sendes og modtages beskeder. Både applikationen og dronen henter deres position fra GPS systemet og dronens kamera filmer brugeren.

1.1.2 Domænemodel - Drone

På figur 1.3 ses domænemodellen for dronen alene.



Figur 1.3: Domænemodel - Drone

Denne domænemodel skal læses for at give et overblik over systemet. FlightController'en læser værdierne fra sine sensorer og måler også spændingen på batteriet. Denne styrer ligeledes signalerne til ESC'erne, som regulerer motorerne og derved dronen. Raspberry Pi 3'en og FlightController'en har en kommunikation hvor der læses og skrives værdier. Raspberry Pi 3'en har flere funktioner tilknyttet andre blokke.

- Den styrer stepmotoren, som igen drejer kameraet.
- Den benytter dronens interface.
- Den aflæser afstandssensoren.
- Den finder dronens GPS lokation.
- Den oprettholder en 3G-forbindelse igennem en 3G-Dongle.
- Den fortæller Arduino Nano'en hvilke PWM signaler dronen skal have.

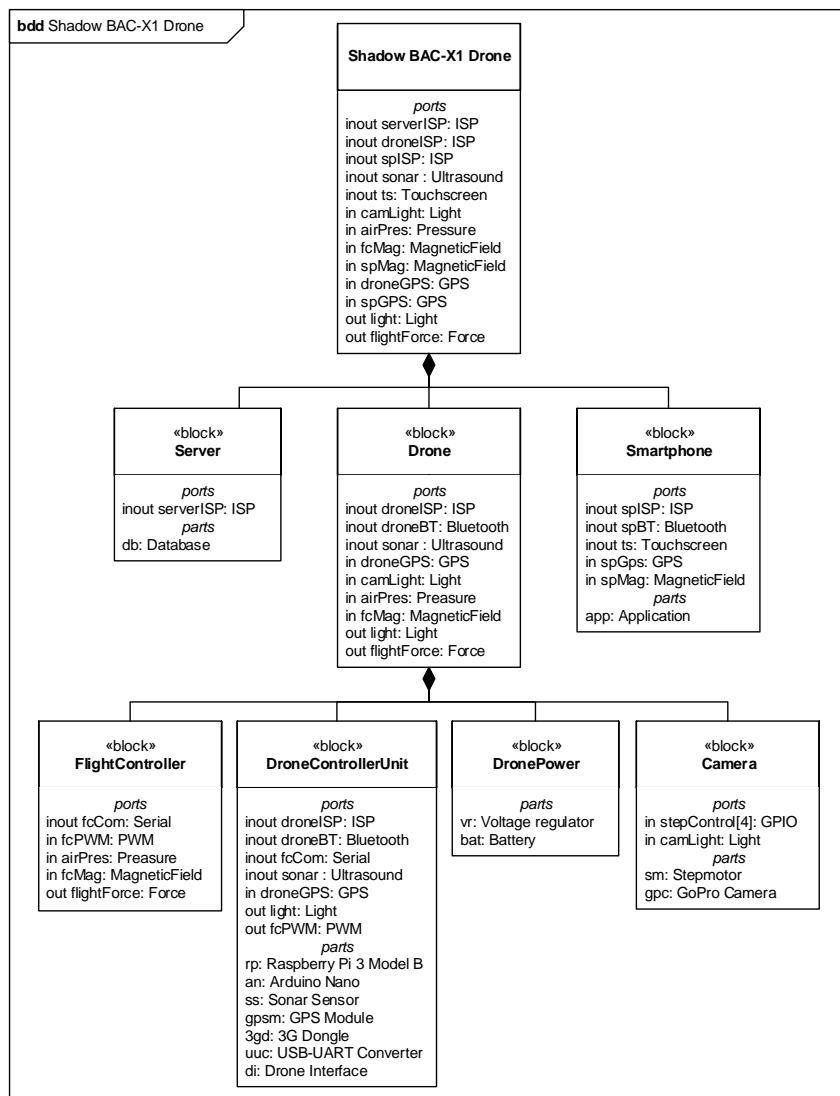
På denne måde kan hele dronens funktionalitet læses ud fra domænemodellen. Denne kan senere benyttes når der skal udarbejdes applikationsmodeller for systemet.

Hardware arkitektur og design

2

2.1 Block definitions diagram

På figur 2.1 ses et BDD for hele systemet.



Figur 2.1: BDD - hele systemet

Det ses på figur 2.1 hvad hele systemet indeholder. Hele systemet hedder Shadow BAC-X1 Drone og er opdelt i tre forskellige blokke; Server, Drone og Smartphone.

2.1.1 Server

Serveren skal indeholde en database. Det er igennem serveren, at kommunikationen imellem dronen og smartphonens applikationen sker.

2.1.2 Smartphone

Smartphonens indeholder interfacet til brugeren, som er en touchskærm. Brugeren interagerer med applikationen, som benyttes til at styre dronen med.

2.1.3 Drone

Dronen indeholder fire blokke:

2.1.3.1 FlightController

FlightController'en består af en række forskellige dele. Heriblandt en Arduino Mega 2560 med et AeroQuad v2.1.3 shield, en række sensorer og nogle motorer. Da dette dog er et færdigt system, bliver det vist som en blackbox og den interne kommunikation i FlightController blokken bliver ikke beskrevet nærmere. Den har dog et serielt interface til DroneControllerUnit'en og forskellige in- og outputs fra motorer og sensorer. Ved hjælp af det serielle interface udlæses de følgende sensorer på FlightController'en:

- Magnetometeret bruges til at måle styrken og retningen af jordens magnetiske felt.
- Accelerometeret bruges til at måle accelerationen, altså vibrationen og bevægelsen i en vilkårlig retning. Accelerometeret kan kun måle hurtigheden af bevægelsen.
- Barometeret bruges til at måle lufttrykket.

2.1.3.2 DroneControllerUnit

DroneControllerUnit'en indeholder:

- Raspberry Pi 3 Model B
- Arduino Nano
- Sonarsensor
- GPS-Module
- 3G-Dongle
- USB-UART converter
- Drone interface

En Raspberry Pi 3 Model B bruges til at styre dronen. Det er den, der agere modtager på dronen, således at der kan kommunikeres med dronen. Raspberry Pi 3 Model B skal ydermere lave alle beregninger vedrørende flyvningen. Denne fortæller Arduino Nano'en hvilke PWM signaler der skal sendes til FlightController'en.

Sonarsensoren skal bruges til at måle afstanden ned til jorden. Den skal på den måde sørge for at dronen ikke kommer for tæt på jorden, medmindre den skal lande. Sonarsensoren skal fungere i samspil med barometeret til at styre højden. GPS-modulet skal bruges til at finde dronens lokation. Ud fra sin egen position og brugerens position kan dronen udregne sin næste position.

3G-modemet skal bruges til at forbinde dronen til internettet. Det er via denne at dronen forbinder til serveren og modtager kommandoer fra brugeren. USB-UART konverteren bruges til at forbinde GPS'en til Raspberry Pi'en gennem en USB port.

Drone interfacet består af forskellige LED'er, der er nærmere beskrevet i interface afsnittet 1.10 i Kravspecifikationen.

2.1.3.3 Camera

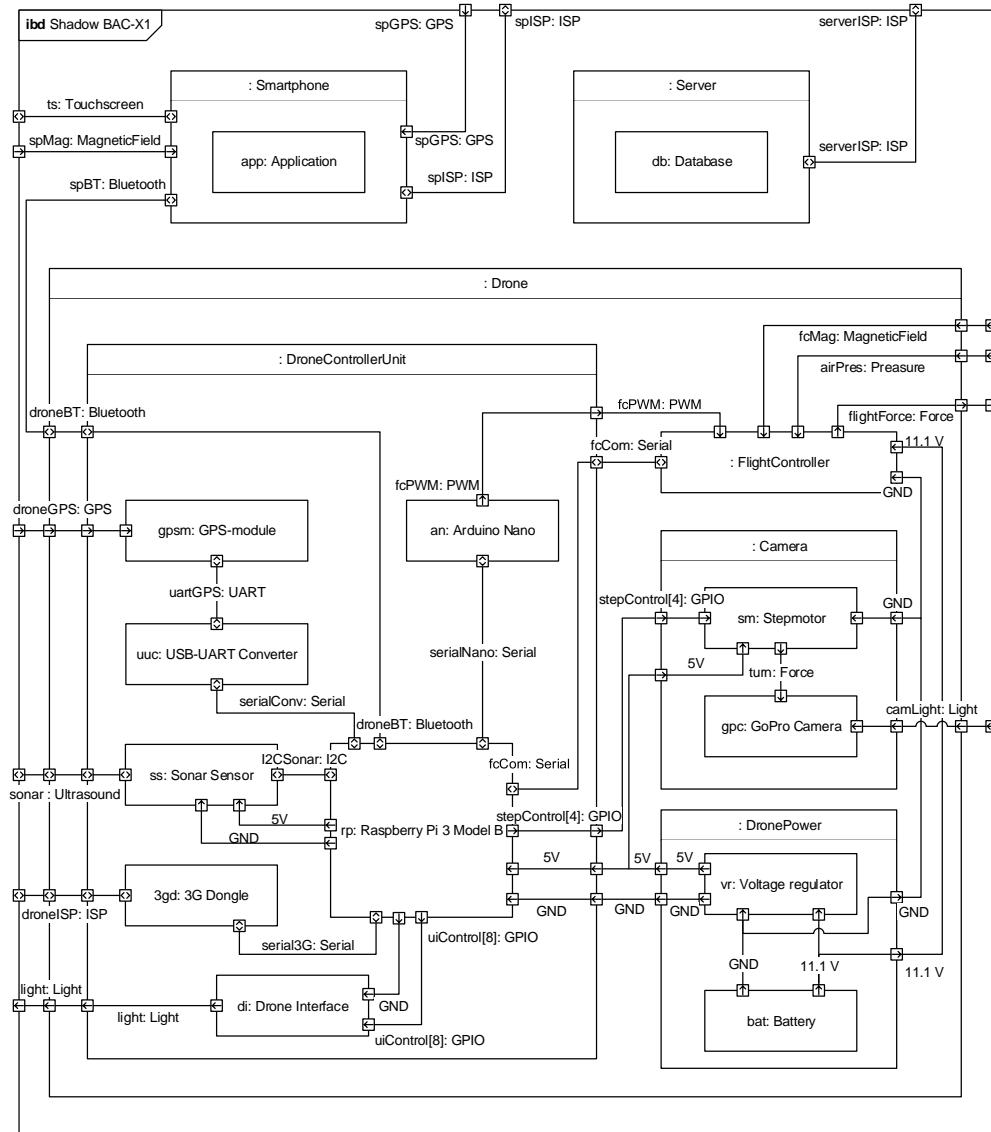
Kamera blokken består af selve GoPro kameraet og en stepmotor, der kan dreje kameraet i den ønskede position baseret på en vinkel.

2.1.3.4 DronePower

Drone power blokken består af et batteri og en spændingsregulator. Blokken er ansvarlig for at levere strøm til de andre blokke på dronen.

2.2 Internal block diagram

På figur 2.2 ses et IBD over det samlede system. Se afsnit 2.3 for en uddybning af diagrammet.



Figur 2.2: IBD - Drone

2.3 Signal oversigt

I de følgende tabeller vil de enkelte blokke og signaler i figur 2.2 blive forklaret.

2.3.1 Smartphone

<i>Smartphone</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbeskrivelse
Smartphonen fungerer som brugerens interface til systemet. Denne benytter 3G og Bluetooth.	ISP	IN/OUT	spISP	Internetforbindelse til database server.
	Bluetooth	IN/OUT	spBT	Dataforbindelse mellem smartphonen og dronen.
	Touchscreen	IN/OUT	ts	Smartphonens touch-skærm.
	GPS	IN	spGPS	GPS data til smartphonen.
	Magnetic-Field	IN	spMag	Magnetisk felt, der måles af sensor.

Tabel 2.1: Signaler - Smartphone

2.3.2 Server

<i>Server</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbeskrivelse
Serveren er bindeleddet i systemet imellem dronen og smartphonen.	ISP	IN/OUT	serverISP	Internetforbindelse til serveren.

Tabel 2.2: Signaler - Server

2.3.3 Drone

<i>Drone</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbeskrivelse
Dronen kommunikerer med serveren, smartphone og flyver i forhold til kommandoer.	ISP	IN/OUT	droneISP	Dronens internettadgang.
	Bluetooth	IN/OUT	droneBT	Bluetooth forbindelsen til smartphone.
	Ultrasound	IN/OUT	sonar	Ultralyd fra sonarsensoren.
	GPS	IN	droneGPS	GPS signal til dronen.
	Light	IN	camLight	Omgivelseslys, der optages af kameraet.
	Pressure	IN	airPres	Airtryk, der bliver målt af sensor.
	Magnetic-Field	IN	fcMag	Magnetisk felt, der måles af sensor.
	Light	OUT	light	LED lys fra Drone Interface.
	Force	OUT	flightForce	Kraften der produceres af dronens motorer.

Tabel 2.3: Signaler - Drone

2.3.3.1 DroneControllerUnit

<i>DroneControllerUnit</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbeskrivelse
DroneControllerUnit'ens ansvar er at kommunikere med serveren og smartphone og reagerer på input fra forskellige sensorer.	5V	IN	5V	5V spændingsforsyning.
	GND	IN	GND	Stelforbindelse.
	ISP	IN/OUT	droneISP	Dronens internetadgang.
	Bluetooth	IN/OUT	droneBT	Bluetooth forbindelsen til smartphone.
	Serial	IN/OUT	fcCom	Seriell forbindelse til Flight-Controlleren.
	Ultrasound	IN/OUT	sonar	Ultralyd fra sonarsensoren.
	GPS	IN	droneGPS	GPS signal til dronen.
	Light	OUT	light	LED lys fra DroneInterface.
	GPIO [4]	OUT	stepControl	Fire GPIO forbindelser til Stepmotoren.
	PWM	OUT	fcPWM	PWM signal til Flight-Controlleren.

Tabel 2.4: Signaler - DroneControllerUnit

<i>Raspberry Pi 3 Model B</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
Raspberry Pi 3'en står for at opretholde kommunikationen imellem brugeren og dronen samt kommunikation med FlightController'en. Denne skal også kunne navigere dronen efter GPS-koordinater.	5V	IN	5V	5V spændingsforsyning.
	GND	IN	GND	Stelforbindelse.
	5V	OUT	5V	Spændingsforsyning til sensorer.
	GND	OUT	GND	Stelforbindelse til sensorer.
	Bluetooth	IN/OUT	droneBT	Bluetooth forbindelsen til smartphone.
	Serial	IN/OUT	serialNano	Forbindelse til Arduino Nano.
	Serial	IN/OUT	serial3G	Forbindelse til 3G dongle.
	Serial	IN/OUT	serialConv	Forbindelse til USB-Converteren.
	Serial	IN/OUT	fcCom	Forbindelse til Flight Controlleren.
	I2C	IN/OUT	I2CSonar	Forbindelse til Sonar Sensoren.
	GPIO [4]	OUT	stepControl	Fire GPIO forbindelser til Stepmotoren.
	GPIO [8]	OUT	uiControl	Otte GPIO forbindelser til Drone Interface.

Tabel 2.5: Signaler - Raspberry Pi 3 Model B

<i>Arduino Nano</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
Arduino Nano'en står for at modtage UART signaler fra Raspberry Pi'en og oversætte disse til et PWM signal til FlightController'en	Serial	IN/OUT	serialNano	Forbindelse til Raspberry Pi. Signalet indeholder også 5V og GND.
	PWM	OUT	fcPWM	PWM signal til Flight-Controlleren.

Tabel 2.6: Signaler - Arduino Nano

<i>Drone Interface</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
Drone Interfacet giver brugeren feedback omkring dronens tilstand.	GND	IN	GND	Stelforbindelse fra Raspberry Pi'en.
	GPIO [8]	IN	uiControl	GPIO signaler til LED'er på Interfacet.
	Light	OUT	light	LED lys der viser dronens status.

Tabel 2.7: Signaler - Drone Interface

<i>3G Dongle</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
3G Dongle'en er Raspberry Pi'en interface med internettet.	ISP	IN/OUT	droneISP	Donglens forbindelse med internettet.
	Serial	IN/OUT	serial3G	Den Serielle forbindelse mellem Raspberry Pi'en og donglen. Indeholder også 5V og GND.

Tabel 2.8: Signaler - 3G Dongle

<i>Sonar Sensor</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
Sonar sensoren er Raspberry Pi'en sensor til at måle afstanden til jorden.	5V	IN	5V	Spæningsforsyning fra Raspberry Pi.
	GND	IN	GND	Stelforbindelse fra Raspberry Pi.
	I2C	IN/OUT	I2CSonar	Forbindelse mellem Sonar Sensoren og Raspberry Pi'en.
	Ultrasound	IN/OUT	Sonar	Ultralyd, der bliver sendt ud og optages fra sensoren.

Tabel 2.9: Signaler - Sonar Sensor

<i>USB-UART Converter</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
Converterer Raspberry Pi'ens USB signal til et UART signal til GPS modulet og omvendt.	Serial	IN/OUT	serialConv	Forbindelsen til Raspberry Pi'en.
	UART	IN/OUT	uartGPS	Forbindelsen til GPS-modulet.

Tabel 2.10: Signaler - USB-UART Converter

<i>GPS Module</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
Modtager GPS signal og sender det videre til USB-UART Konverteren.	UART	IN/OUT	uartGPS	Forbindelsen til USB-UART Konverteren.
	GPS	IN	droneGPS	GPS signalet der modtages.

Tabel 2.11: Signaler - GPS Module

2.3.3.2 FlightController

<i>FlightController</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
FlightController'en interfacer med Arduino Nano'en og styrer dronens motorer gennem de tilhørende ESC'er.	11.1V	IN	11.1V	11.1V spændingsforsyning.
	GND	IN	GND	Stelforbindelse.
	Seriell	IN/OUT	fcCom	Forbindelse til Raspberry Pi'en.
	PWM	IN	fcPWM	PWM signalet fra Arduino Nano'en.
	Magnetic-Field	IN	fcMag	Magnetisk felt, der optages af Flight-Controllerens sensorer.
	Pressure	IN	airPres	Lufttryk, der optages af Flight-Controllerens sensorer.
	Force	OUT	flightForce	Kraften der produceres af dronens motorer.

Tabel 2.12: Signaler - FlightController

2.3.3.3 Camera

<i>Camera</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
Kamera modulet består af stepermotoren og selve kameraet.	5V	IN	5V	5V spændingsforsyning.
	GND	IN	GND	Stelforbindelse.
	GPIO [4]	IN	stepControl	Kontrol signaler fra Raspberry Pi.
	Light	IN	camLight	Omgivelseslys, der optages af kameraet

Tabel 2.13: Signaler - Camera

<i>Stepmotor</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
Stepmotoren står for at roterer det monterede kamera.	5V	IN	5V	5V spændingsforsyning.
	GND	IN	GND	Stelforbindelse.
	GPIO [4]	IN	stepControl	Kontrol signaler fra Raspberry Pi.
	Force	OUT	turn	Kraften der drejer kameraet.

Tabel 2.14: Signaler - Stepmotor

<i>GoPro Camera</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
Kamera der filmer brugeren.	Force	IN	turn	Kraften fra stepermotoren, der drejer kameraet
	Light	IN	camLight	Omgivelseslys, der optages af kameraet

Tabel 2.15: Signaler - GoPro Camera

2.3.3.4 DronePower

<i>DronePower</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
Giver forsyning til de andre komponenter på dronen.	5V	OUT	5V	5V spændingsforsyning.
	11.1V	OUT	11.1V	11.1V spændingsforsyning.
	GND	OUT	GND	Stelforbindelse

Tabel 2.16: Signaler - DronePower

<i>Voltage Regulator</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
Regulerer 11.1V til 5V.	11.1V	IN	11.1V	11.1V spændingsforsyning som input.
	GND	IN	GND	Stelforbindelse.
	5V	OUT	5V	5V spændingsforsyning som output.
	GND	OUT	GND	Stelforbindelse

Tabel 2.17: Signaler - Voltage Regulator

<i>Battery</i>				
Funktionsbeskrivelse	Signaler	IN/OUT	Signalnavn	Signalbekrивelse
11.1V LiPo batteri.	11.1V	OUT	11.1V	11.1V spændingsforsyning.
	GND	OUT	GND	Stelforbindelse

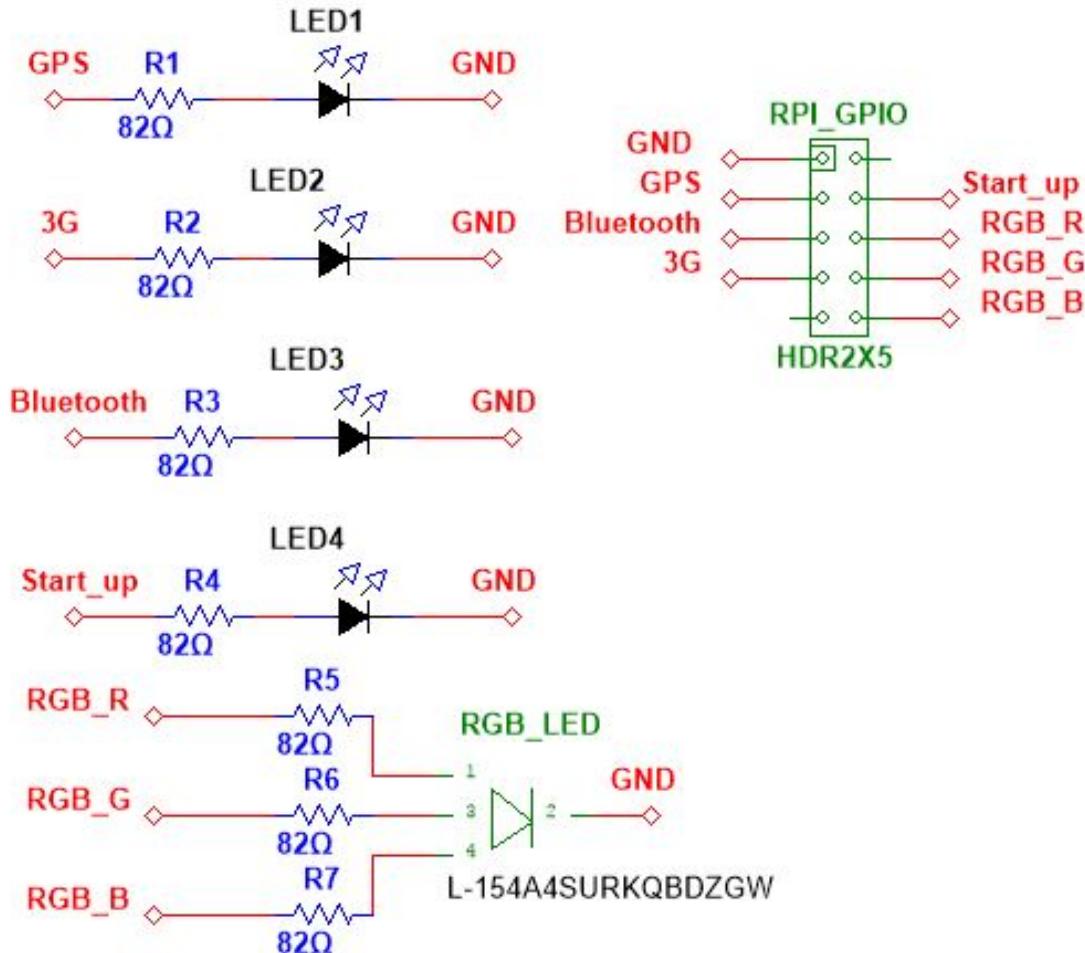
Tabel 2.18: Signaler - Battery

2.4 Print design

Denne sektion vil indeholde en beskrivelse af printdesignene. Der vil også være en beskrivelse af de enkelte komponenter som er valgt.

2.4.1 Droneinterface

Figur 2.3 viser et kredsløbsdesign af droneinterfacet.



Figur 2.3: Droneinterfaceprint

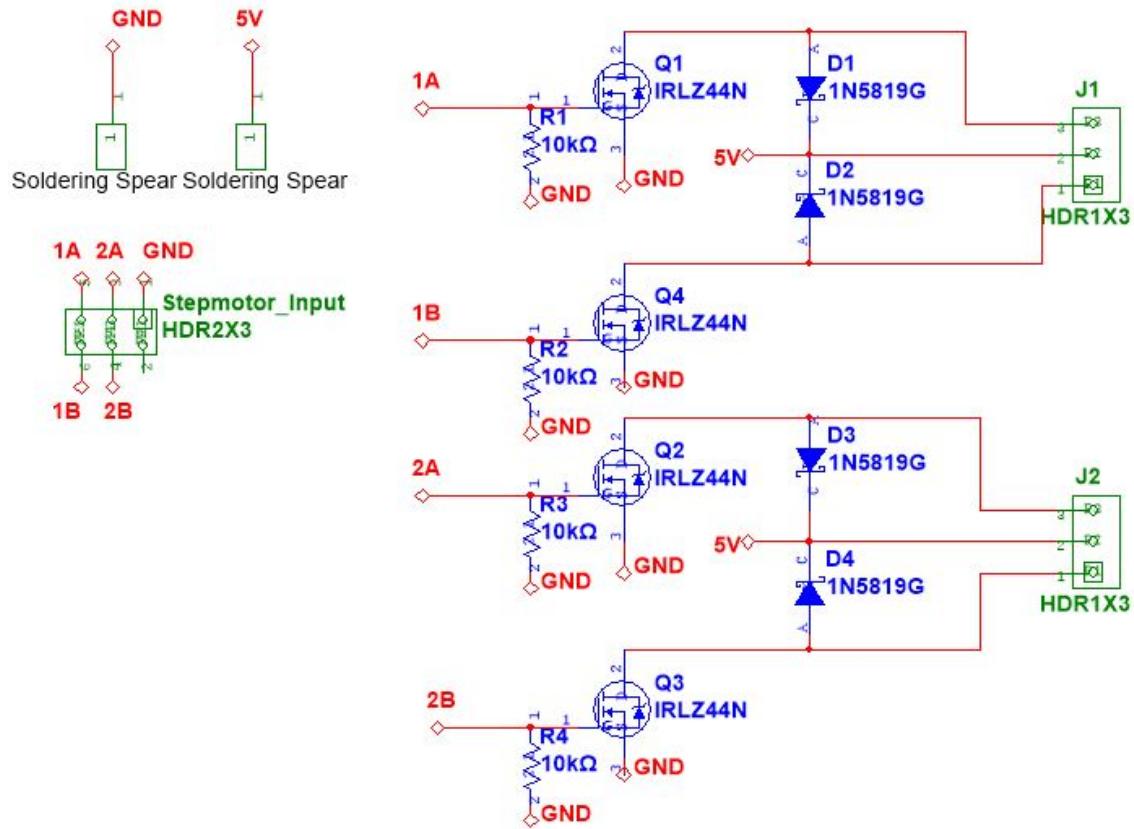
Der er valgt en modstand på 82Ω for at leve en strøm på ca. 15 mA til LED'erne. Dette er gjort ud fra følgende beregning:

$$\frac{3.3V - 2.2V}{0.015A} = 73.3\Omega$$

Dette er gjort for hver LED. RGB dioden fungerer som tre LED'er, derfor er der sat en modstand på 82Ω foran hvert ben på RGB dioden. Alle pins på dette print drives af Raspberry Pi'en.

2.4.2 Stepmotor

Figur 2.4 viser et kredsløbsdesign af stepmotor-printet.



Figur 2.4: Stepmotorprint

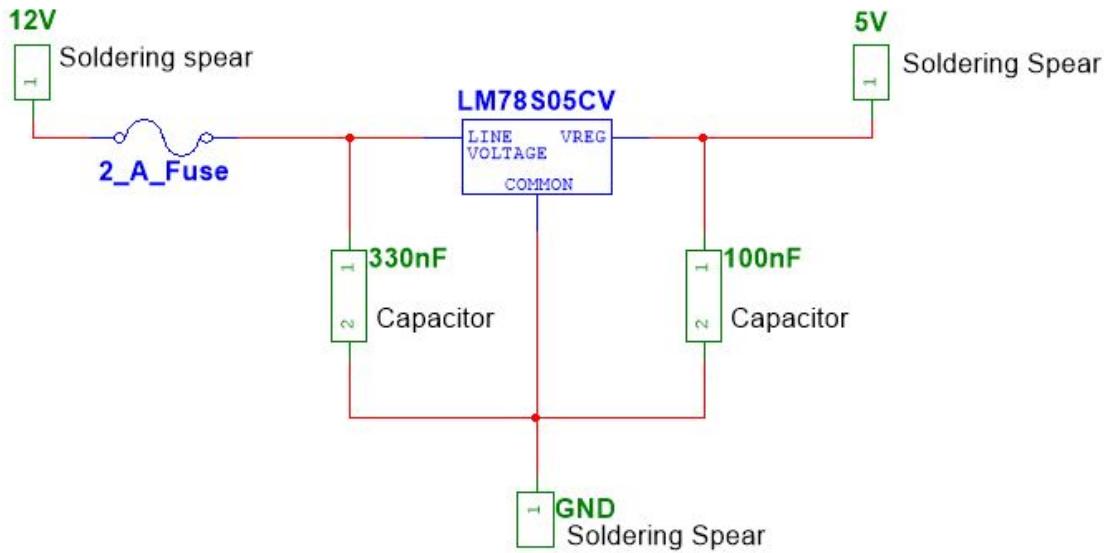
Stepmotor-printet skal bruges til at styre stepmotoren. Dette er nødvendigt, da Raspberry Pi'en ikke kan levere en høj nok strøm til at trække motoren. På printet ses det, at stepmotoren bliver forsynet af en 5V forsyning, som kommer fra spændingsregulatoren. Raspberry Pi'en bruges til at styre de fire MOSFET's. MOSFET'ene der bruges hedder IRLZ44N.

Når Raspberry Pi'en sender et højt signal ind på gate-benet af MOSFET'en åbner denne op for at der kan løbe en strøm imellem source- og drain-benet på MOSFET'en. Derved kan der løbe en strøm fra 5V forsyningen, igennem stepmotoren og ned i ground. Når Raspberry Pi'en sætter et lavt signal, altså 0V, på gate benet vil MOSFET'en lukke for at der kan løbe en strøm imellem drain- og source-benet. Modstanden på 10kOhm, sidder som en pull-down modstand. Denne modstand gør, at GPIO-benet fra Raspberry Pi'en der sidder på gate-benet holdes lavt, når Raspberry Pi'en sætter det til et lavt signal.

Dioden sørger for at beskytte MOSFET'en mod en høj strøm. Dette er dog lidt overflødig i denne sammenhæng, da motoren ikke ville kunne trække en så høj strøm, som vil kunne skade MOSFET'en. MOSFET'en kan klare en strøm på op til 47A kontinuerligt.

2.4.3 Spændingsregulator

Figur 2.5 viser et kredsløbsdesign af spændingsregulatoren.



Figur 2.5: Spændingsregulatorprint

Spændingsregulator-printet skal benyttes til at regulere 12V fra batteriet ned til 5V, således at stepmotoren, RPi'en og de komponenter der er forsynes af RPi'en, kan forsynes. I Analysen afsnit 1.11 er der lavet et estimat for, hvor stort et strømforbrug der vil være på dronen. Ud fra de komponenter der skal forsynes af spændingsregulatoren, er spændingsregulatoren, LM78S05CV, valgt [1]. Denne regulator kan leve op til 2 ampere. Der er indsatt en sikring på 2 ampere for at sikre systemet mod en kortslutning.

2.5 Stykliste

Dette afsnit vil indeholde en færdig liste med komponenter der er indkøbt og brugt i den færdige drone.

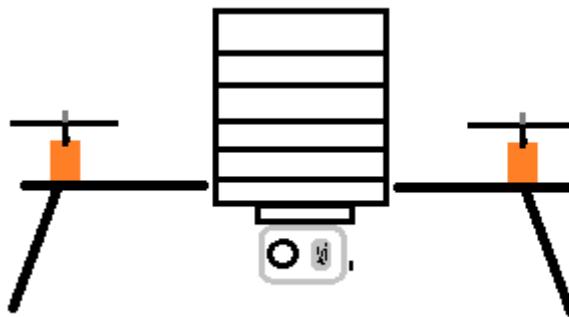
- Raspberry Pi 3
- Arduino Nano
- Arduino Mega 2560
- AeroQuad v2.1.3 shield til Arduino Mega 2560
- AeroQuad Cyclone Frame pakke
- Lipo 3-cell Batteri
- DC-DC Step-Down 3A spændingsregulator.
- Huawei 3533 USB 3G dongle
- Adafruit Ultimate GPS Logger Shield
- Pulse Engineering W4000G197 Antenne
- 5V stepmotor - 28BYJ-48
- Stepmotorprint
 - 4x IRLZ44N mosfet
 - 4x 10k Ω modstand
 - 4x 1N5819G diode
- Droneinterfaceprint
 - 2x L-154A4SURKQBDZGW RGB LED
 - Grøn LED
 - Gul LED
 - Rød LED

2.6 Mekanisk Design

Denne sektion omhandler det mekaniske design på dronen. Dette indebærer hvordan dronen er bygget op, hvor komponenterne fra afsnit 2.5 skal placeres og hvordan kameraet fungerer.

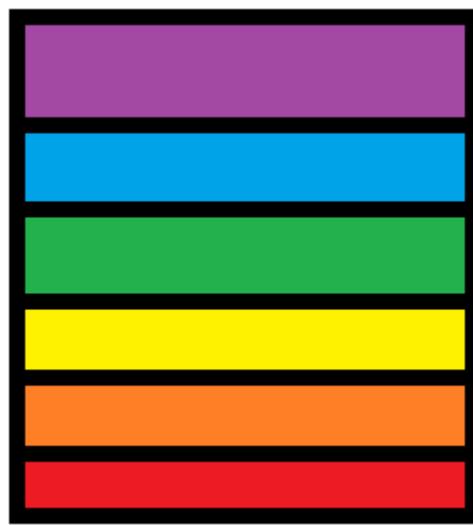
2.6.1 Dronens opbygning

Selve dronen er bygget som anvist på AeroQuads hjemmeside [2]. På figur 2.6 ses en skitse af dronen. Her kan det ses, at centrummet af dronen er bygget op af runde skiver, med plads i mellem hver skive. I pladsen imellem hver skive, kan dronens elektroniske dele placeres.



Figur 2.6: Skitse af dronen

Det er nødvendigt at placere de elektroniske dele på bestemte niveauer, da nogle af de elektroniske dele har nogle specifikke krav i forhold til placeringen.



Figur 2.7: Opdelingen af pladser med farver

På figur 2.7 er der indsatt farver på hvert niveau. På det røde niveau er batteriet og kameraet placeret. Det er nødvendigt, at placere kameraet på undersiden af dronen, for at det har

det bedste udsyn til brugeren.

På det orange niveau er ESC'erne placeret. Disse er placeret her, da de skal sidde tæt på ledningerne fra motorerne.

På det gule niveau, er dronens powerbox placeret. Powerboxen fungerer som en stikdåse og giver derved mulighed for at resten af systemet på dronen kan forsynes fra batteriet. ESC'erne skal forbindes til powerboxen, hvilket gør det nærliggende at placere powerboxen tæt på disse. Derudover er printet til styring af stepermotoren og spændingsregulatoren placeret på dette niveau.

På det grønne niveau er FlightController'en placeret samt Arduino Nano'en. Da Nano'en skal kommunikere med FlightController'en er det nærliggende af placere den lige ved siden af. FlightController'en er placeret på dette niveau, da denne skal forbindes til ESC'erne.

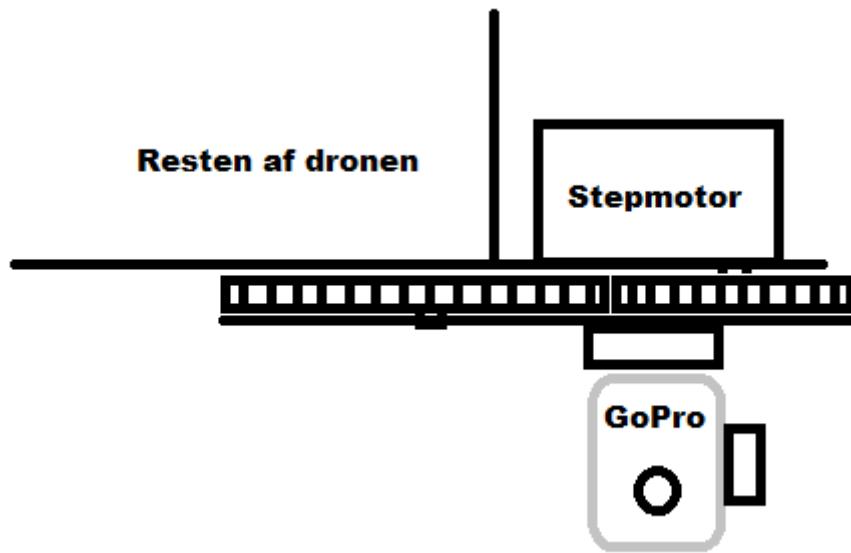
På det blå niveau er Raspberry Pi 3'en placeret. Denne skal forbindes til både Nano'en, FlightController'en og GPS-modulet.

På det lilla niveau er GPS-modulet placeret. Da GPS-modulet kun skal forbindes til Raspberry Pi 3'en, er dette placeret på øverste niveau, lige over Raspberry Pi 3. GPS modulets antennen skal placeres på toppen af dronen således denne kan nå satelitterne.

Dronens interface er placeret på ydersiden af dronen. Dette ligger derfor ikke på et specielt niveau. Derudover er 3G-donglen fæstnet til en af armene ud til en af motorerne.

2.6.2 Mekanisk design af kamera

Kameraet skal kunne dreje omkring dronens y-akse. For at opnå dette, benyttes der en stepermotor med et tandhjul. Tandhjulet passer ind i et andet tandhjul, som er fæstet til en plade hvorpå kameraet sidder. Derved kan kameraet drejes vha. stepermotoren. På figur 2.8 kan der ses en skitse af dette.



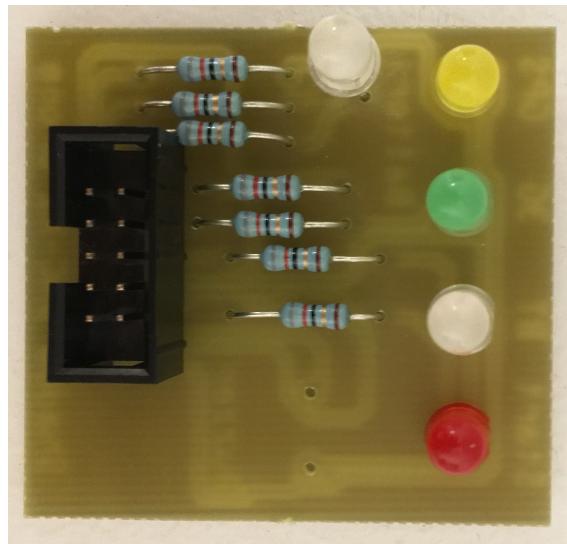
Figur 2.8: Skitse af kamerastyring

2.7 Implementering

Dette afsnit omhandler implementeringen af hardwaren i projektet. Dette omfatter prints, batteri og dronens stel.

2.7.1 Print implementering

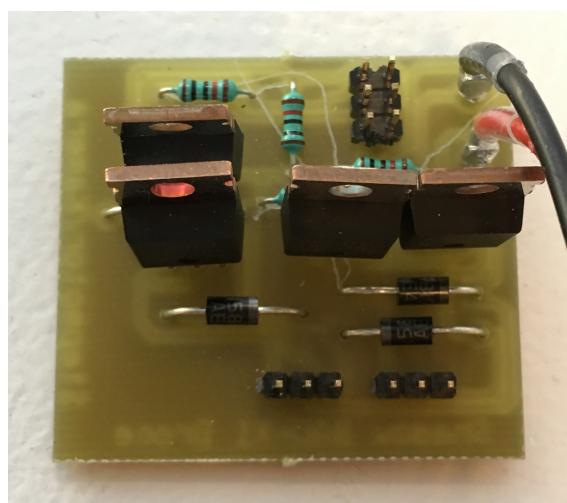
Printene blev produceret af Ingeniørhøjskolen. De blev leveret med kobber baner og huller til komponenter. På figur 2.9 ses det færdige interface-print.



Figur 2.9: Interface-print

På printet er der to huller der ikke bliver brugt, da det var planlagt at implementere en buzzer. Det blev dog forkastet. Det blev ikke valgt at fremstille et nyt print da det ikke har indflydelse på printets resterende funktionalitet.

På figur 2.10 ses det færdige stepmotor-print.



Figur 2.10: Stepmotor-print

Spændingsregulator-printet blev kasseret, da det blev erfaret under implementeringen, at det var fejldimensioneret. Fejldimensioneringen gjorde, at spændingsregulatoren blev for varm, hvilket gjorde at udgangsspændingen faldt. Da Raspberry Pi'en er meget følsom overfor en underspænding, betød det at Raspberry Pi'en genstartede. Dette er ikke hensigtsmæssigt. Det blev derfor valgt at bestille en færdig spændingsregulator. Der blev bestilt en DC-DC Step-Down [3]. Denne blev også analyseret i Analysen i afsnit 1.6.4.

2.7.2 AeroQuad Cyclone stel

AeroQuad Cyclone stellet består af det fysiske stel, batteri, powerbox, ESC'er, motorer, og propeller. Dette kan samles på mange forskellige måder og i forskellige konfigurationer. På AeroQuad's hjemmeside [2] findes en manual, som kan fører brugeren gennem opsætningen af dronen. Her er der også en mulighed for at brugeren kan lave dronen, som denne gerne vil have den. I dette projekt er det valgt at opsætte dronen i en konfiguration med 4 motorer. Dette kaldes for en Quad X opsætning.

2.7.3 Batteri

I systemet benyttes LiPo batterier. I dette tilfælde skal der være ekstra opmærksomhed omkring brugen af disse. Når LiPo batterier skal oplades eller aflades skal dette gøres med en speciel charger. I dette projekt er der benyttet en Turnigy Accucell-8150 [4] charger, som også kan balance batterierne. Fremgangsmåde med LiPo batterierne er følgende:

- Når batteriet skal bruges oplades dette.
- Må ikke aflades til lavere end 3V per celle
- Efter endt brug er det en god ide at sætte batteriet i storage mode.

Når et LiPo batteri oplades er tommelfinger reglen at man ikke på lade med mere end 1C. C er her kapaciteten af batteriet. Så benyttes et batteri på 4200mA må der altså maksimalt oplades med 4.2A. Hvis batteriet aflades til mindre end 3V per celle kan der ske skader på batteriet, som gør denne ustabil og i risikozonen for brand. For ikke at skade batteriet er det vigtigt at sætte batteriet i storage mode når det ikke skal bruges. Batterierne skal opbevares ved stuetemperatur.

Batteriet skal med jævne mellemrum balanceres, da dette sikre at cellerne har den samme værdi og dermed også sikre stabiliteten af batteriet.[5]

Software arkitektur og design 3

3.1 N+1 View

N+1 er en model til at udvikle softwaren i et given system [6]. Dette gøres ved at dele processen op i forskellige View's. I dette projekt har vi benyttet os af de følgende View's; Logical-, Process-, Data-, Deployment-, Security- og Implementation View. Hver enkelt View vil herunder kort blive beskrevet.

Logical View

I dette View opdeles systemet i CPU'er. Alle systemets Use Case's gennemgås for hver CPviewU og de relevante konceptuelle klasser findes og tilhørende sekvensdiagrammer udarbejdes. Baseret på sekvensdiagrammerne udledes der statiske klassediagrammer. I slutningen af dette View kan de enkelte statiske klassediagrammer samles til et.

Process View

Dette View omhandler de forskellige processer i de enkelte CPU'er. Hvordan forskellige processer interagerer med hinanden på samme CPU. Her kan det også blyses nærmere hvilke processer der er tråde og hvilke der ikke er.

Data View

I dette View bliver dataet i systemet beskrevet. Dette kan være hvordan en given CPU håndterer data omkring f.eks. GPS-koordinater. I hvilken struktur gemmes de og hvordan tilgås de. Her kan ligeledes være forklaringer omkring databaser, logs og hvorvidt dataet bliver gemt i flash-hukommelse eller på anden vis.

Deployment View

Dette View omhandler protokollerne mellem de forskellige CPU'er. Her skal de forskellige protokoller som benyttes forklares. Eventuelle protokoller man selv har defineret skal også forklares her.

Security View

Dette View omhandler den sikkerhed som ligger i systemet. Dette kan være brugeropsætninger eller brugen af krypteret data.

Implementation View

I Implementation View skal det forklares for læseren hvordan systemet skal bygges og opsættes, såfremt kildekoden er tilgængelig. Dette afsnit beskriver også særlige tiltag i projektet.

3.2 Logical View

I dette View skal CPU'ens konceptuelle klasser opstilles. Dette gøres på baggrund af Use Case's og domænemodeller. Igennem applikationsmodellen, som er et værktøj til at udarbejde dette View, findes først de overordnede klasser til CPU'en baseret på en given Use Case. Herefter opstilles et eller flere sekvensdiagrammer, som fortæller hvordan disse klasser interagerer med hinanden. Baseret på sekvensdiagrammet kan et statisk klassediagram opstilles til hver Use Case. Disse kan til sidst samles til et komplet statisk klassediagram for den pågældende CPU.

Der identificeres 3 typer af klasser i applikationens modellen.

- Controller
- Boundary
- Domain

Controller

Controller klassen står for at håndtere gennemførelsen af Use Case'en. Denne står for kommunikationen med Boundary klasser og Domain klasser.

Boundary

Dette er grænsefladerne til systemet. Dette kan være et interface til brugeren eller en klasse, som håndterer kommunikationen med en aktør. Dette kunne f.eks være et GPS modul.

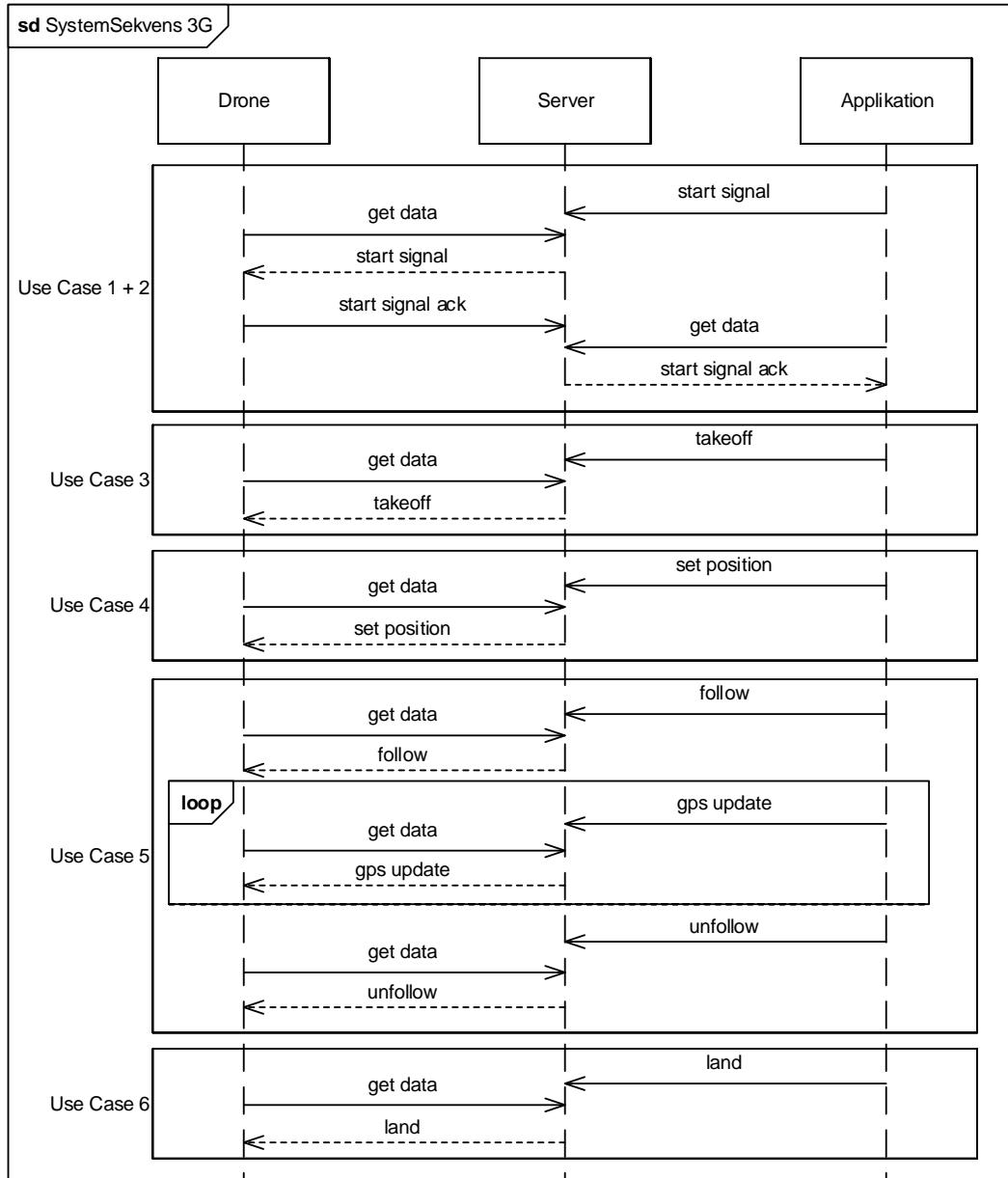
Domain

Domain klasser indeholder systemets vedvarende data, dette kunne f.eks være en database, indstillinger eller på andre måde data, som skal kunne forblive.

Hvis en funktion fremkommer i flere statiske klassediagrammer vil disse efter første gang være skrevet med grå skrift, således kun de nye tilføjede funktioner er tydelige

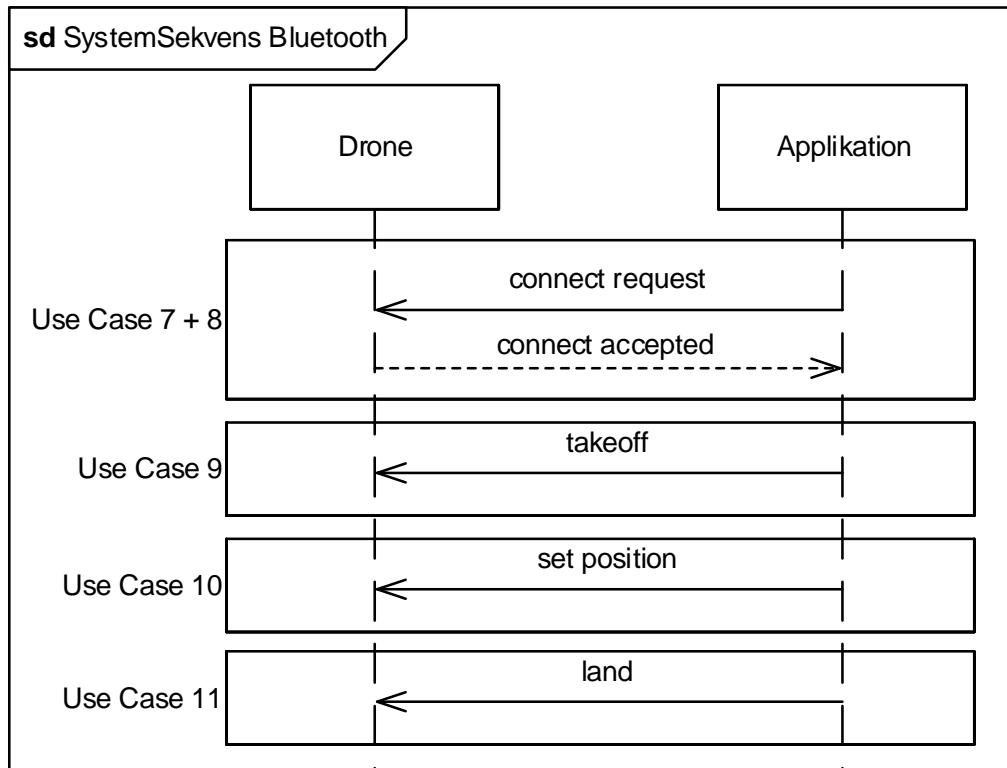
3.2.1 System sekvensdiagram

Dette afsnit opstiller hele systemet og beskriver kort alle Use Case's opdelt i 3G og Bluetooth. Sekvensdiagrammerne viser kommunikationen mellem systemets blokke på et overordnet niveau. Disse sekvensdiagrammer er ment, for at skabe et overblik over systemet før de enkelte Use Case's opdeles.



Figur 3.1: System sekvensdiagram - 3G

Figur 3.1 viser kommunikationen mellem blokke når 3G benyttes. Det ses at alle beskederne sendes til serveren hvorefter disse kan hentes efterfølgende. Use Case 3, 4 og 6 er blot en enkelt besked fra applikationen til dronen. I Use Case 5 har dronen løbende behov for GPS opdateringer fra applikationen. Disse fortsætter indtil dronen får besked på at stoppe med at følge applikationen.



Figur 3.2: System sekvensdiagram - Bluetooth

I figur 3.2 ses systemet sekvensdiagram for Bluetooth. Bluetooth fungerer således at applikationen sender en request til dronen, som denne accepterer. Herefter er det blot en enkelt besked fra applikationen til dronen.

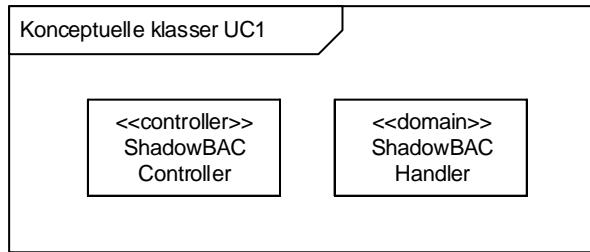
3.2.2 Server

I denne sektion vil Logical View blive udarbejdet for CPU'en, som håndterer serveren. Serveren er bindeleddet imellem dronen og applikationen, når 3G benyttes. Da serveren kun kan benyttes gennem 3G, vil følgende Use Case's ikke blive opstillet i denne sektion, da disse har med Bluetooth at gøre.

- Use Case 7 - Bluetooth Parringmode Drone
- Use Case 8 - Application Bluetooth Start
- Use Case 9 - Autonomous TakeOff (Bluetooth)
- Use Case 10 - Set Drone Position (Bluetooth)
- Use Case 11 - Autonomous Landing (Bluetooth)

3.2.2.1 Use Case 1 - System Start Drone

Ved at analysere Use Case 1 og domænemodellerne er de konceptuelle klasser fundet. Disse ses på figur 3.3.



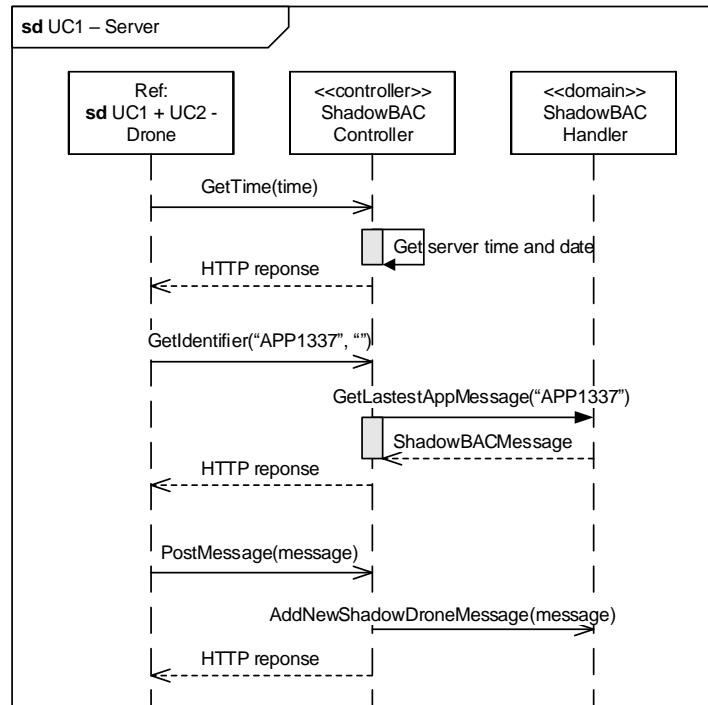
Figur 3.3: CPU Server - UC1 konceptuelle klasser

Tabel 3.1 indeholder en kort beskrivelse af de konceptuelle klasser.

Navn	Beskrivelsen
ShadowBACController	Controlleren, som står for at håndtere forskellige HTTP beskeder fra og til dronen.
ShadowBACHandler	Denne klasse står for at håndtere serverens database. At hente og gemme beskeder i databasen

Tabel 3.1: Beskrivelser af klasserne i figur 3.3

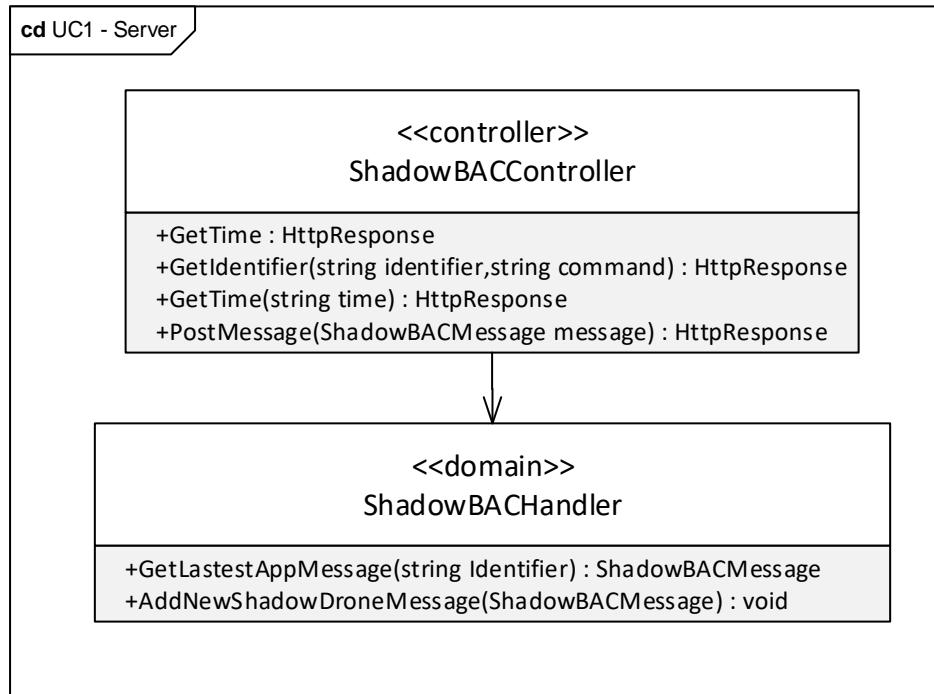
På baggrund af de konceptuelle klasser i figur 3.3 kan et sekvensdiagram, baseret på Use Case 1, laves.



Figur 3.4: CPU Server - UC1 sekvensdiagram

Serverens ansvar i denne Use Case er at håndtere de beskeder denne modtager fra dronen. Funktionerne i *ShadowBACController*'en bliver kaldt ved at dronen sender en HTTP GET eller POST besked til bestemte URL's.

Det første dronen gør er, at bede serveren om tiden og dato'en. Serveren svarer tilbage med en *ShadowBACMessage*, som denne opretter, hvor kun felterne, dato og tid, er sat. Herefter forespørger dronen den seneste besked fra applikationen "APP1337". Serveren henter den seneste besked fra databasen med netop denne *identifier* og returnerer denne til dronen. Hvis dronen modtager et validt Start-signal fra applikationen svarer denne tilbage, ved at sende en HTTP POST besked. Serveren gemmer denne besked i databasen, således at applikationen kan forespørge om denne på et senere tidspunkt.

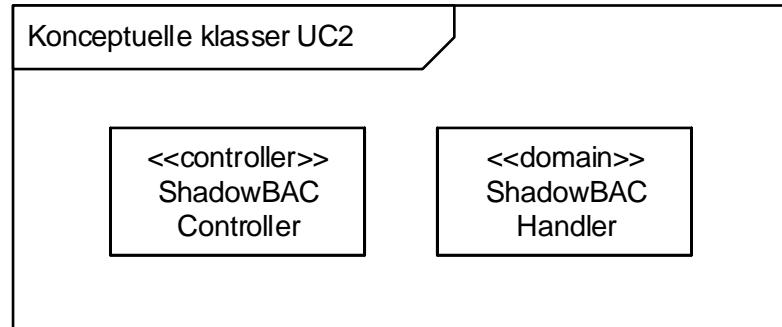


Figur 3.5: CPU Server - UC1 Klassediagram

På figur 3.5 ses det statiske klassediagram baseret på sekvensdiagrammet. De respektive funktioner er trukket ud af sekvensdiagrammet og indsat i de enkelte klasser. Funktionen *GetIdentifier* tager to parametre, men dronen sender blot en tom string med til *command*, da denne parameter ikke benyttes af dronen.

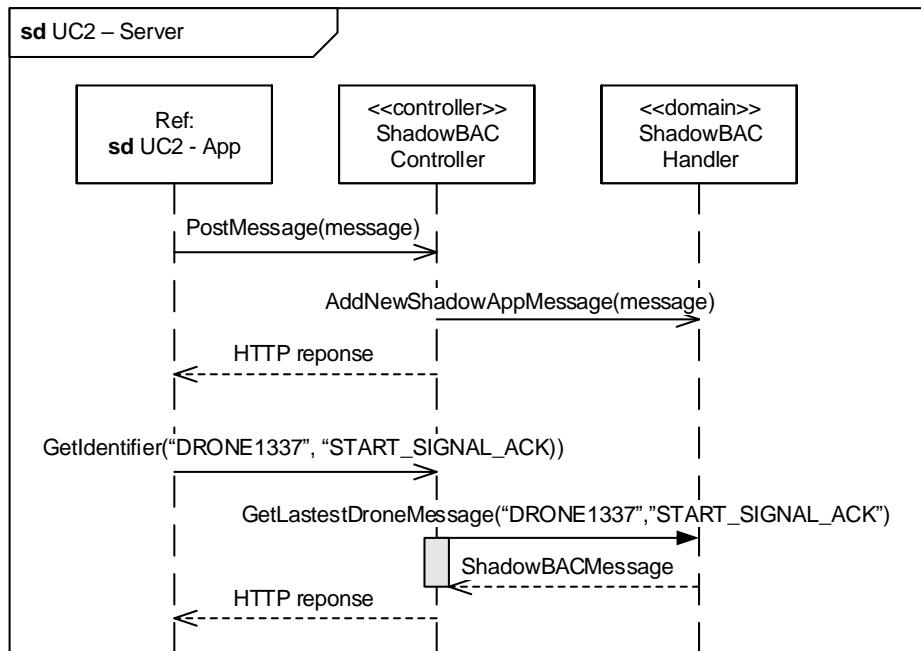
3.2.2.2 Use Case 2 - Applikation Start

Ved at analysere Use Case 2 og domænemodellerne, er de konceptuelle klasser fundet. Disse ses på figur 3.6.



Figur 3.6: CPU Server - UC2 konceptuelle klasser

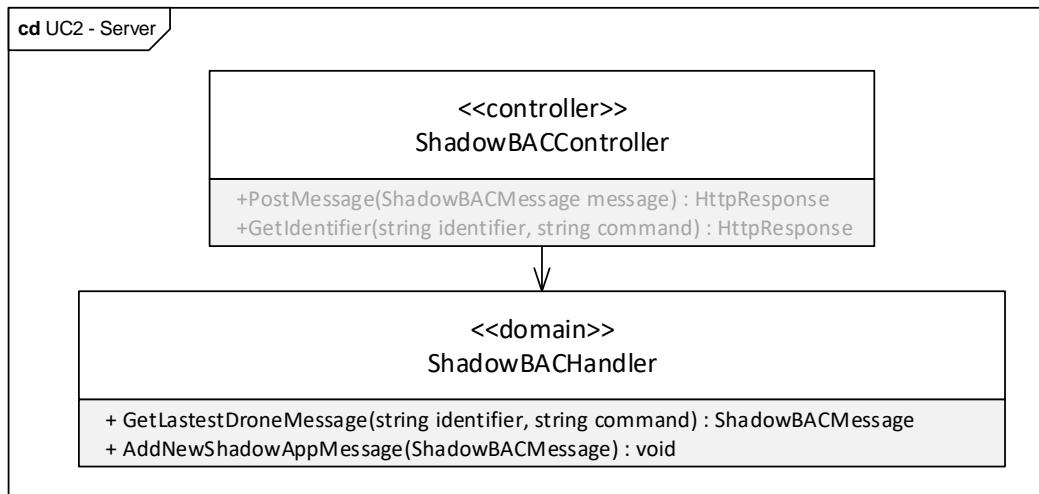
Dette er de samme konceptuelle klasser, som i Use Case 1. Disse vil derfor ikke blive forklaret yderligere, da dette er gjort i Use Case 1. På baggrund af de konceptuelle klasser i figur 3.6 kan et sekvensdiagram laves baseret på Use Case 2.



Figur 3.7: CPU Server - UC2 sekvensdiagram

Serverens ansvar i denne Use Case er at håndtere de beskeder, denne modtager fra applikationen. Funktionerne i `ShadowBACController`'en bliver kaldt ved at applikationen sender en HTTP GET eller POST besked til bestemte URL's.

Det første applikationen gør, er at sende en HTTP POST besked til serveren med et Start-signal. Serveren gemmer beskeden i databasen således, at dronen kan tilgå denne senere. Herefter henter applikationen den seneste besked fra "DRONE1337", som var en "START_SIGNAL_ACK" kommando. Såfremt Start-signalen er hentet af dronen, har denne svaret med et acknowledgement, som serveren henter fra databasen og returnerer til applikationen gennem et HTTP response.



Figur 3.8: CPU Server - UC2 Klassediagram

På figur 3.8 ses det statiske klassediagram baseret på sekvensdiagrammet. De respektive funktioner er trukket ud af sekvensdiagrammet og indsat i de enkelte klasser. Applikationen benytter i denne Use Case to parametre i funktionen *GetIdentifier*, da denne ønsker kun at få den seneste acknowledgement fra dronen.

3.2.2.3 Use Case 3 - Autonomous TakeOff

Use Case 3 benytter samme funktionalitet som tidligere beskrevet i Use Case 1 og 2. Der vil ikke blive udarbejdet konceptuelle klasser, sekvensdiagram og statisk klassediagram, da disse ville indeholde samme funktionalitet som fundet i Use Case 1 og 2.

Use Case 3 starter med at applikationen sender en besked til serveren gennem funktionen *PostMessage* præcis som i Use Case 2, se figur 3.8. Når dronen modtager et TakeOff-signal fra applikation, ved dronen at denne skal lette og flyve til højden angivet i beskedten fra applikationen. Den eneste funktion dronen benytter i denne Use Case, er funktionen *GetIdentifier*, som også fremgår i Use Case 1's statiske klassediagram, se figur 3.5.

3.2.2.4 Use Case 4 - Set Drone Position

Use Case 4 benytter samme funktionalitet som tidligere beskrevet i Use Case 1 og 2. Der vil ikke blive udarbejdet konceptuelle klasser, sekvensdiagram og statisk klassediagram, da disse ville indeholde samme funktionalitet fundet i Use Case 1 og 2.

Use Case 4 starter med at applikationen sender en besked til serveren gennem funktionen *PostMessage* præcis som i Use Case 2, se figur 3.8. Når dronen modtager et "SETPOS" signal fra applikation vil dronen henter GPS koordinatet fra beskedten. Den eneste funktion dronen benytter i denne Use Case, er funktionen *GetIdentifier*, som også fremgår i Use Case 1's statiske klassediagram, se figur 3.5.

3.2.2.5 Use Case 5 - Follow User

Use Case 5 benytter samme funktionalitet som tidligere beskrevet i Use Case 1 og 2. Der vil ikke blive udarbejdet konceptuelle klasser, sekvensdiagram og statisk klassediagram da disse ville indeholde samme funktionalitet som fundet i Use Case 1 og 2.

Use Case 5 starter med at applikationen sender en besked til serveren gennem funktionen *PostMessage* præcis som i Use Case 2, se figur 3.8. Applikationen begynder herefter at sende beskeder kontinuerligt til dronen. Når dronen modtager et Follow-signal fra applikation, vil dronen begynde kontinuerligt at hente beskeder fra serveren. Den eneste funktion dronen benytter i denne Use Case, er funktionen *GetIdentifier* til at hente beskeder fra serveren. Når dronen modtager et Unfollow-signal stopper denne med løbende at hente data fra serveren. Funktionerne benyttet af dronen i denne Use Case fremgår i Use Case 1's statiske klassediagram, se figur 3.5.

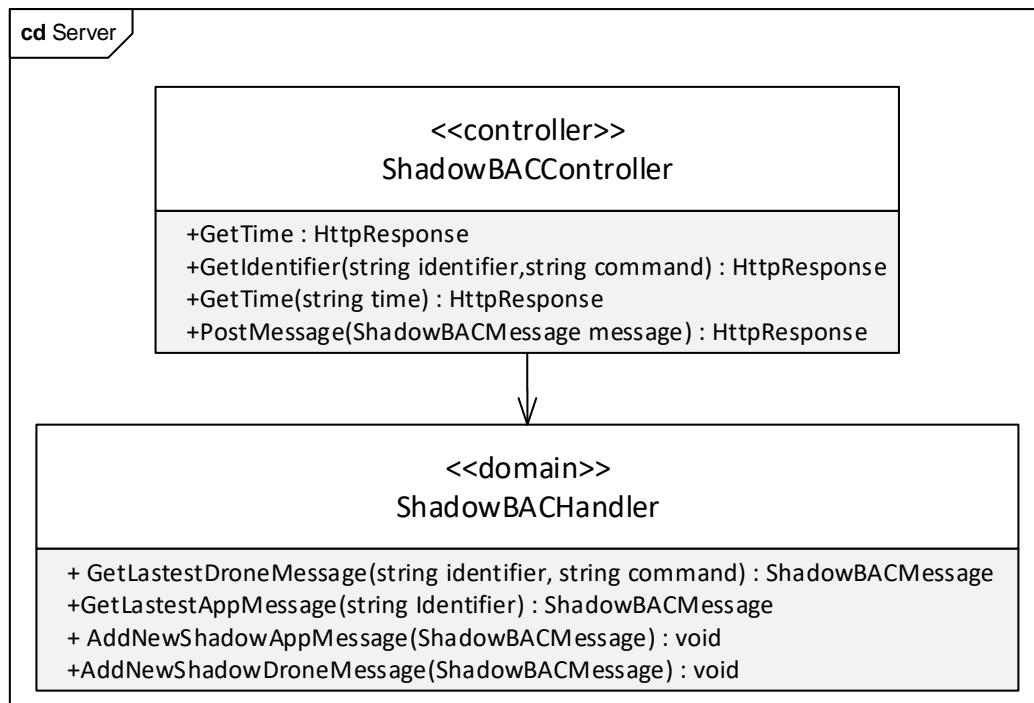
3.2.2.6 Use Case 6 - Autonomous Landing

Use Case 6 benytter samme funktionalitet som tidligere beskrevet i Use Case 1 og 2. Der vil ikke blive udarbejdet konceptuelle klasser, sekvensdiagram og statisk klassediagram, da disse ville indeholde samme funktionalitet som fundet i Use Case 1 og 2.

Use Case 6 starter med at applikationen sender en besked til serveren gennem funktionen *PostMessage* præcis som i Use Case 2, se figur 3.8. Når dronen modtager et Land-signal fra applikation, ved dronen at denne skal lande. Den eneste funktion dronen benytter i denne Use Case, er funktionen *GetIdentifier*, som også fremgår i Use Case 1's statiske klassediagram, se figur 3.5.

3.2.2.7 Samlet klassediagram for CPU'en på serveren

Baseret på de statiske klassediagrammer udarbejdet gennem serverens Logical View kan et samlet klassediagram opstilles. Dette er vist på figur 3.9.



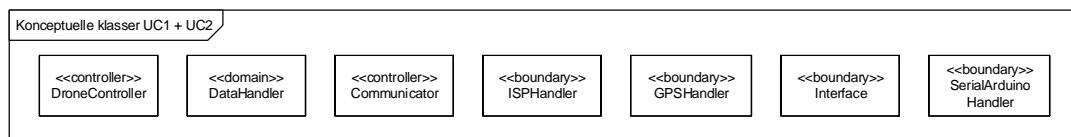
Figur 3.9: CPU Server - Samlet klassediagram for serveren

3.2.3 Drone

I denne sektion vil Logical View blive udarbejdet for den CPU, der styrer dronen. På dronen findes i alt tre CPU'er, Raspberry Pi'en, Arduino Nano'en og FlightControlleren. FlightControlleren er ikke udviklet i dette projekt og denne vil ikke blive forklaret gennem N+1. Arduino Nano'en har kun en enkelt opgave og vil ikke blive udarbejdet for alle Use Case's i systemet. Denne vil blive forklaret kort til sidst i dronens Logical View.

3.2.3.1 Use Case 1 og 2

Gennem Use Case 1, Use Case 2 og domænemodellen for dronen er der opstillet konceptuelle klasser for Use Case 1 og 2. Disse konceptuelle klasser kan ses på figur 3.10.



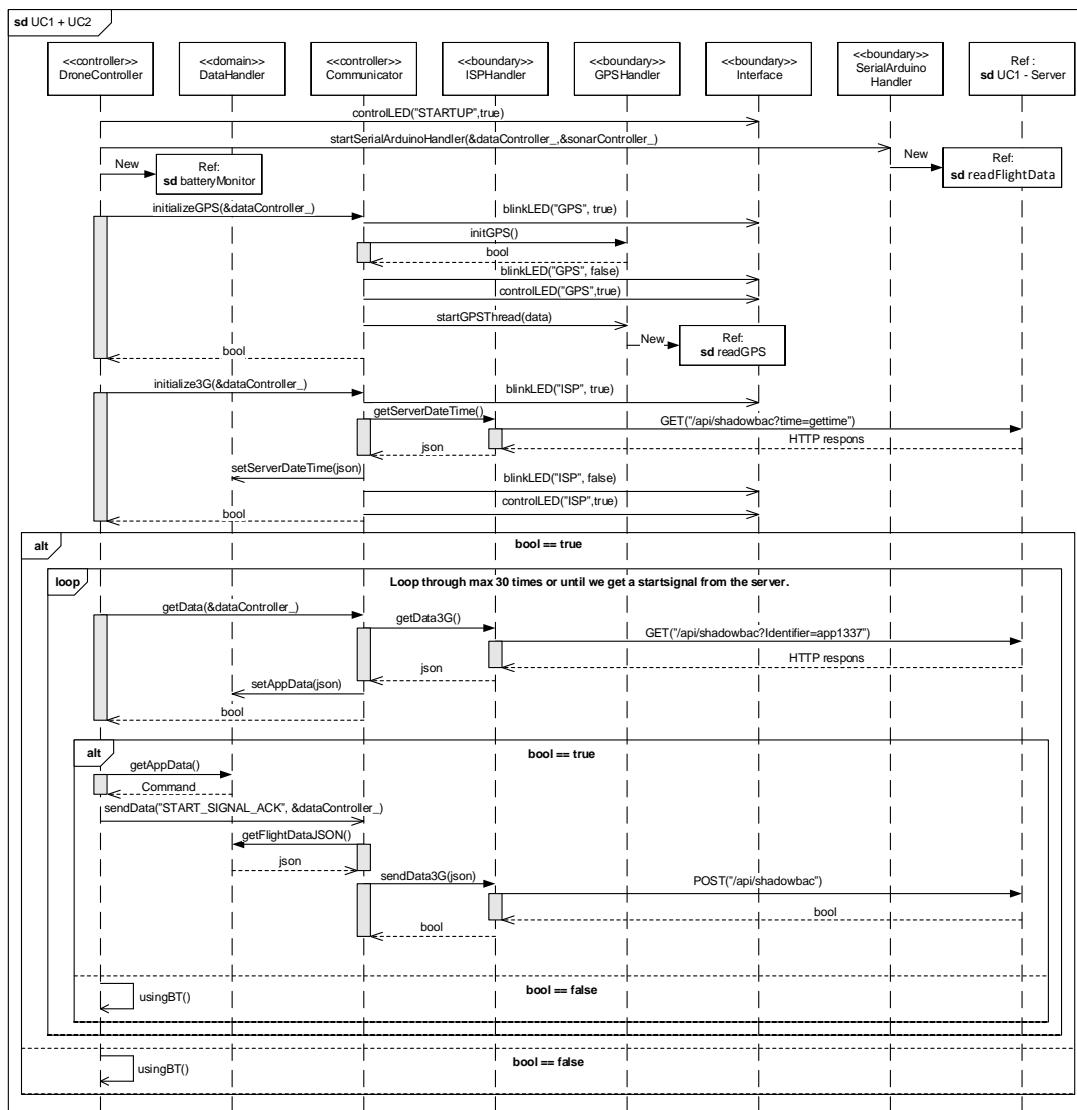
Figur 3.10: CPU Drone - UC1 og UC2 konceptuelle klasser

De konceptuelle klasser beskrives dybere i tabel 3.2.

Navn	Beskrivelsen
DroneController	Dette er main-controlleren på dronen. Den sætter alle processor igang.
DataHandler	Denne klasse står for al lagring og håndtering af data på dronen.
Communicator	Denne klasse står for at håndtere al ekstern kommunikation på dronen. Dette indebærer kommunikationen over Bluetooth, 3G og GPS.
ISPHandler	Denne klasse står for kommunikationen over 3G-modemet.
GPSHandler	Denne klasse står for at komunikere med og konfigurere GPS-modulet.
Interface	Denne klasse står for at styre interfacet på dronen.
SerialArduinoHandler	Denne klasse står for at håndtere den serielle kommunikation fra Raspberry Pi'en ud til både FlightControlleren og Arduino Nano'en.

Tabel 3.2: Beskrivelser af klasserne i figur 3.10

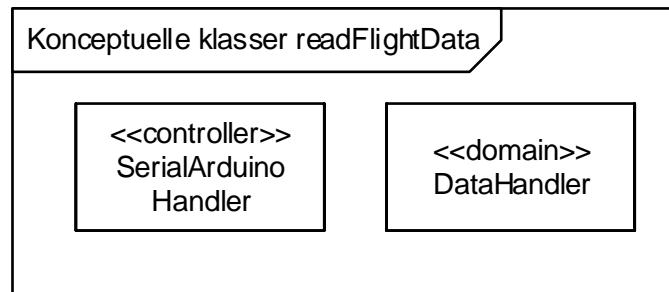
Ud fra de konceptuelle klasser kan der laves et sekvensdiagram. Sekvensdiagrammet dækker både Use Case 1 og Use Case 2 på dronesiden. Sekvensdiagrammet ses på figur 3.11.



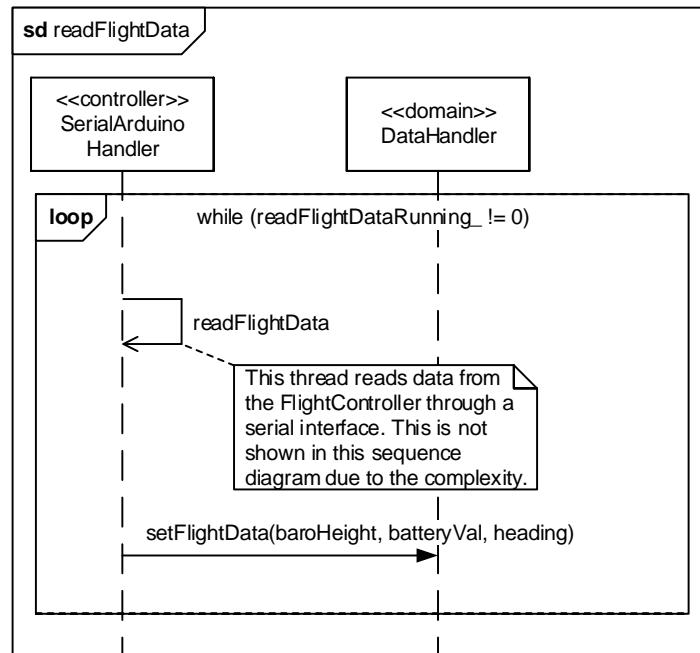
Figur 3.11: CPU Drone - UC1 og UC2 Sekvensdiagram

Først startes kommunikationen til systemets Arduino'er gennem *SerialArduinoHandler*'en. Herefter startes tråden til at monitorerer batteriets spænding. Sekvensdiagrammet er baseret på at der er 3G forbindelse. Dog er det specificeret hvad der sker hvis *initialize3G()* eller *getData(&dataController_)* går galt. Loopet vil hvert andet sekund kigge på det modtagede data fra serveren, og så validere på om et Start-signal er modtaget. Er Start-signalet ikke modtaget efter 30 loops, et minut, så vil systemet gå i Bluetooth mode. Hvis systemet benytter 3G kaldes funktionen *using3G()*, hvis Bluetooth benyttes, *usingBT()*.

De tre tråde der oprettes gennem Use Case 1 og 2 bliver vist i de følgende tre afsnit. De konceptuelle klasser til disse tråde er allerede tidligere forklaret og vil derfor ikke blive forklaret igen.

FCThread

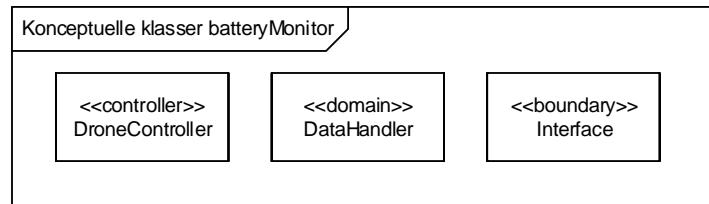
Figur 3.12: CPU Drone - Konzeptuelle klasser FlightControllerThread



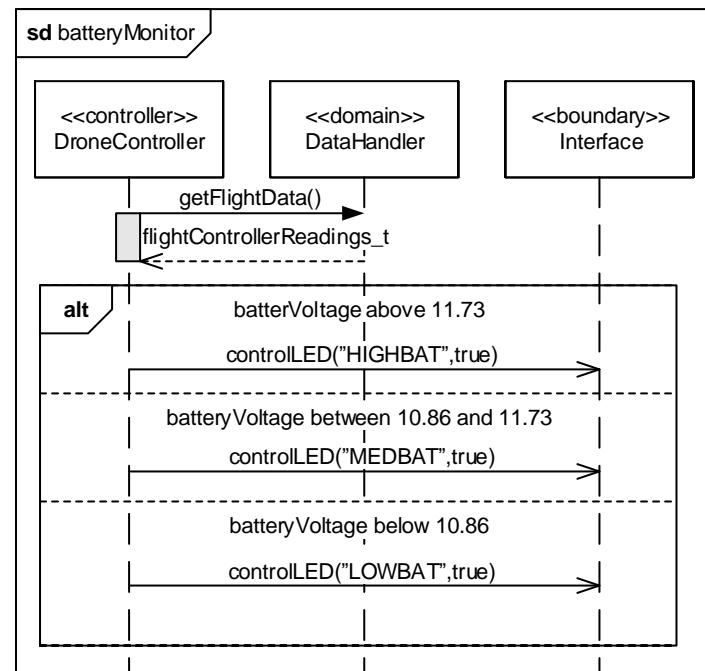
Figur 3.13: CPU Drone - Sekvens diagram FlightControllerThread

Denne tråd læser værdier fra FlightController'en gennem den serielle kommunikation. Denne tråd gemmer disse data i `DataHandler`'en således at andre klasser eller tråde kan tilgå dataet.

BatteryMonitorThread

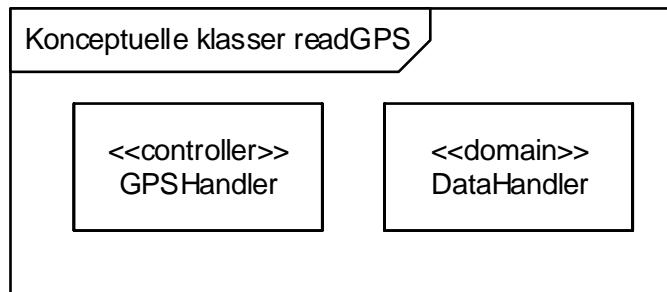


Figur 3.14: CPU Drone - Konzeptuelle klasser BatteryMonitor

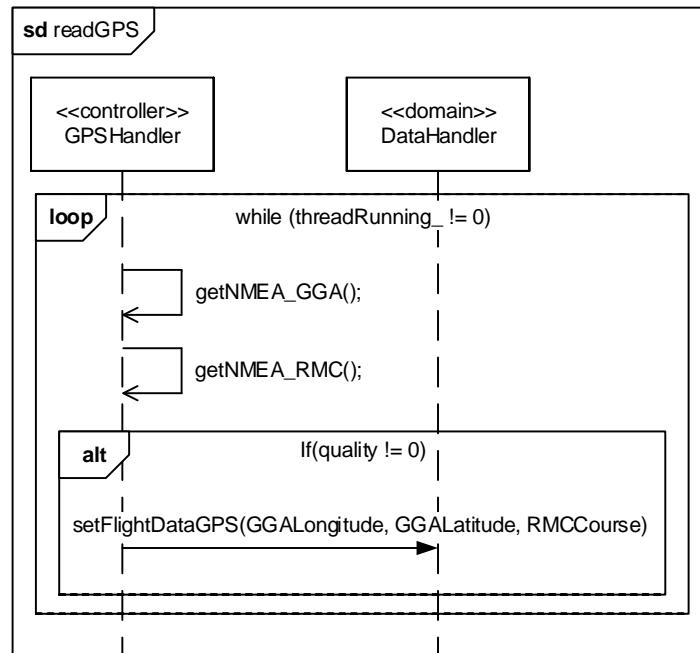


Figur 3.15: CPU Drone - Sekvensdiagram BatteryMonitor

Denne tråd henter batterispænding fra *DataHandler*'en og baseret på denne lyser RGB dioden med en given farve.

GPSThread

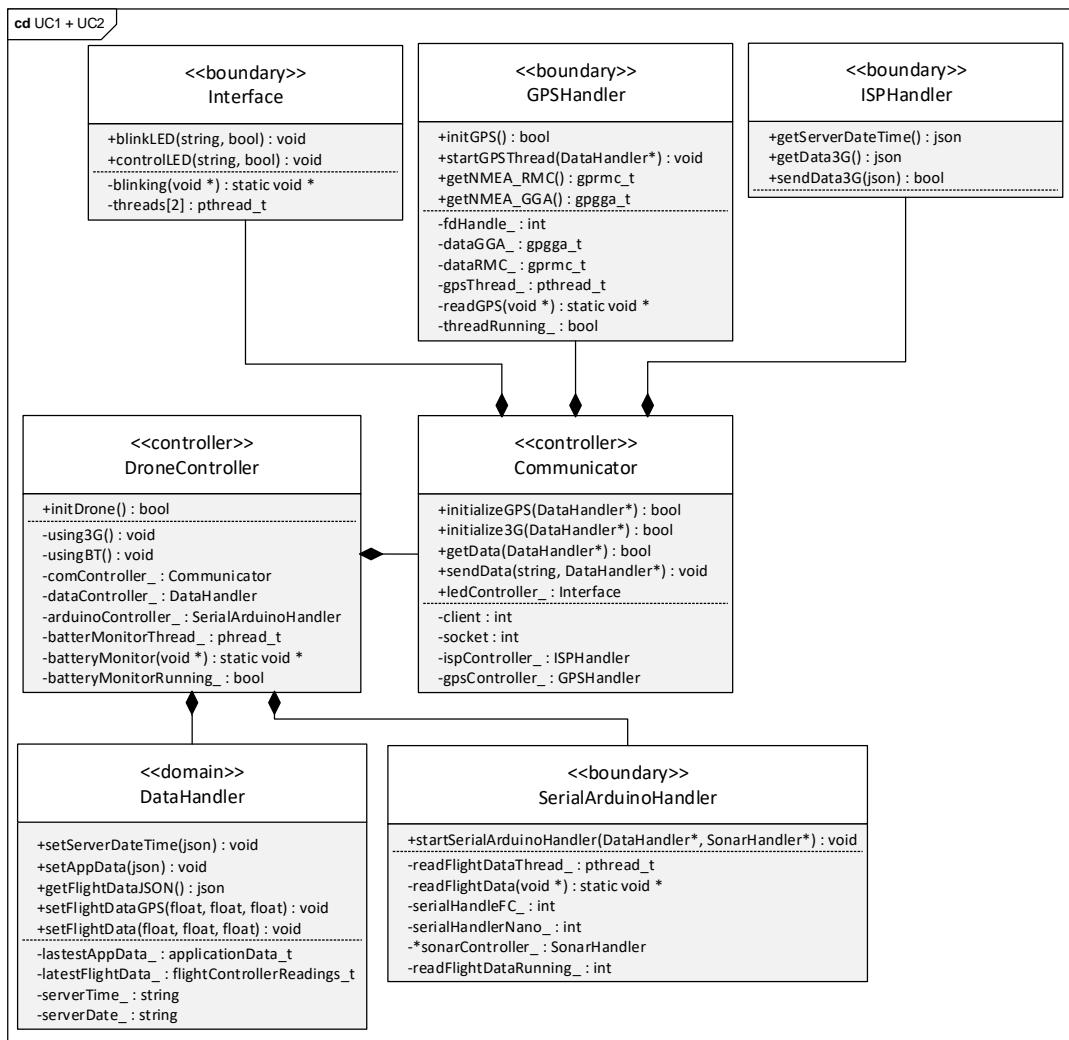
Figur 3.16: CPU Drone - Konceptuelle klasser GPS



Figur 3.17: CPU Drone - Sekvens diagram GPS

GPS tråden læser først en GPGGA besked og dernæst en GPRMC besked fra dronens GPS modul. Den gemmer herefter relevante data i *DataHandler*'en.

Baseret på sekvensdiagrammerne kan der opstilles et statisk klassediagram for Use Case 1 og 2 samt de tilhørende tråde. Dette ses på figur 3.18

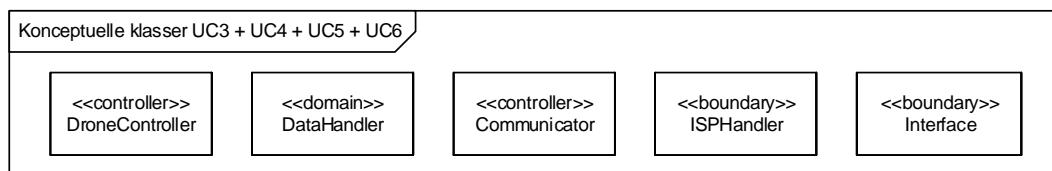


Figur 3.18: CPU Drone - UC1 og UC2 Klassediagram

Det statiske klassediagram viser alle funktionerne der benyttes i systemet på dronen i Use Case 1 og 2 samt de tilhørende tråde.

3.2.3.2 Using3G

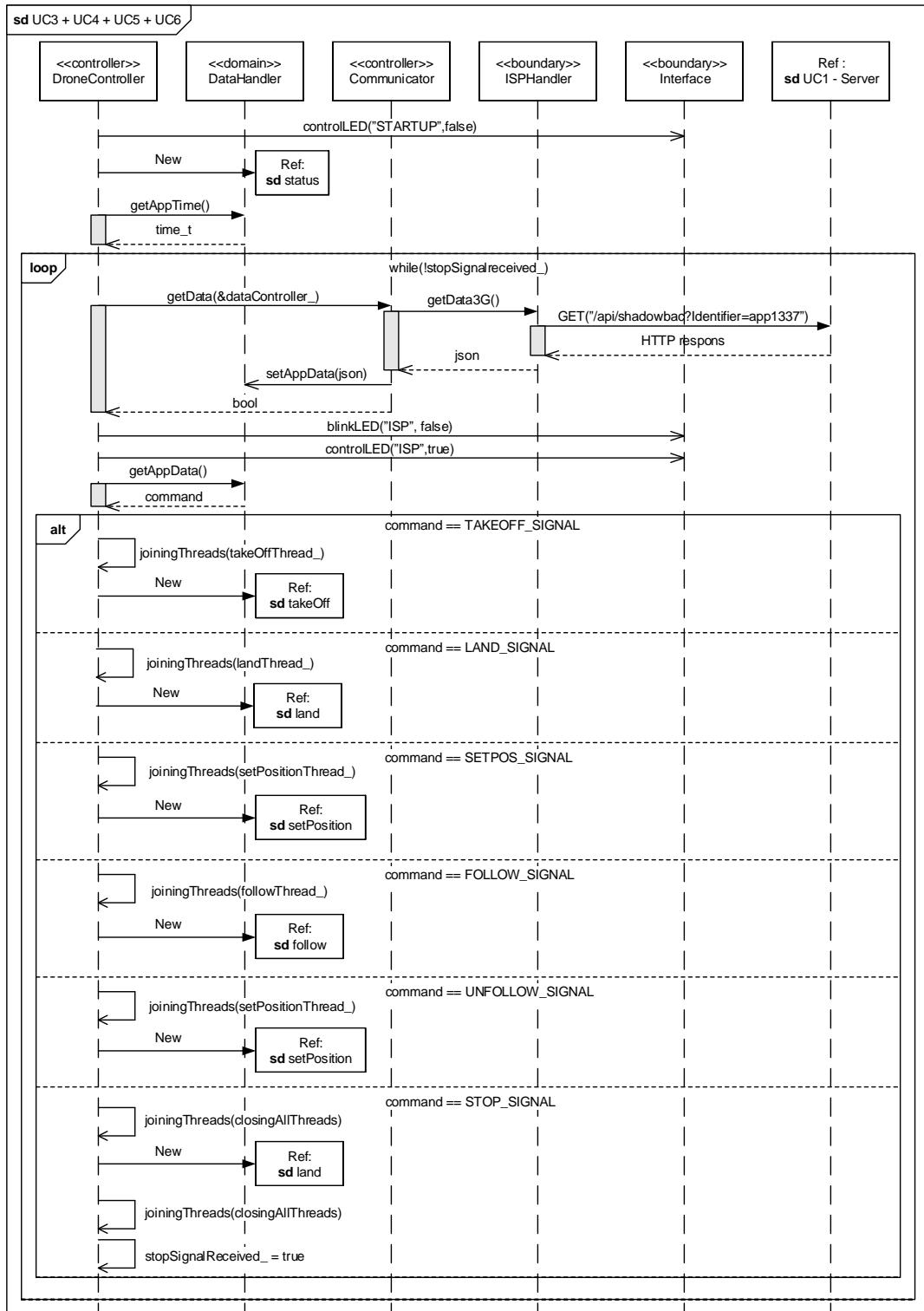
Using3G er den funktion, som styrer dronen når 3G er valgt. Denne funktion er lavet på baggrund af Use Case 3, 4, 5 og 6. De konceptuelle klasser kan ses på figur 3.19.



Figur 3.19: CPU Drone - Using3G - Konceptuelle klasser

De konceptuelle klasse er allerede beskrevet tidligere i tabel 3.2.

Ud fra de konceptuelle klasser kan der laves et sekvensdiagram. Dette sekvens diagram viser funktionen *using3G()* som danner grundlag for Use Case 3, 4, 5 og 6. Sekvensdiagrammet ses på figur 3.20.



Figur 3.20: CPU Drone - using3G - Sekvensdiagram

Sekvensdiagrammet er lavet ud fra, at der er 3G forbindelse. Først startes tråden *status*. Denne sender en status besked til serveren hvert 5. sekund. Herefter hentes tiden, da seneste besked blev modtaget. Denne bruges til at sikre, at der er modtaget en ny besked og der

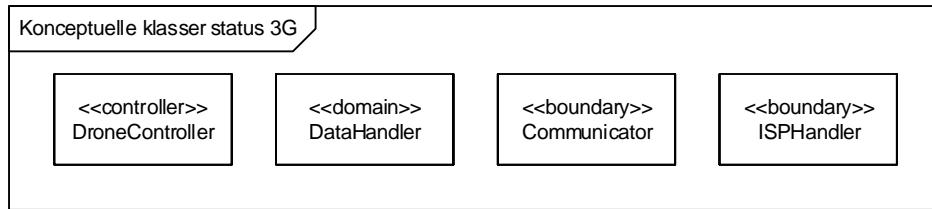
ikke blot er læst en gammel besked fra serveren.

Herefter startes et loop, som henter en ny besked fra serveren. Dette kan lede til seks tilfælde i disse Use Case's. Når dronen modtager et signal fra serveren starter den en tilhørende tråd. Hvis en anden tråd allerede kører, bliver denne lukket ned først og herefter startes den nye tråd.

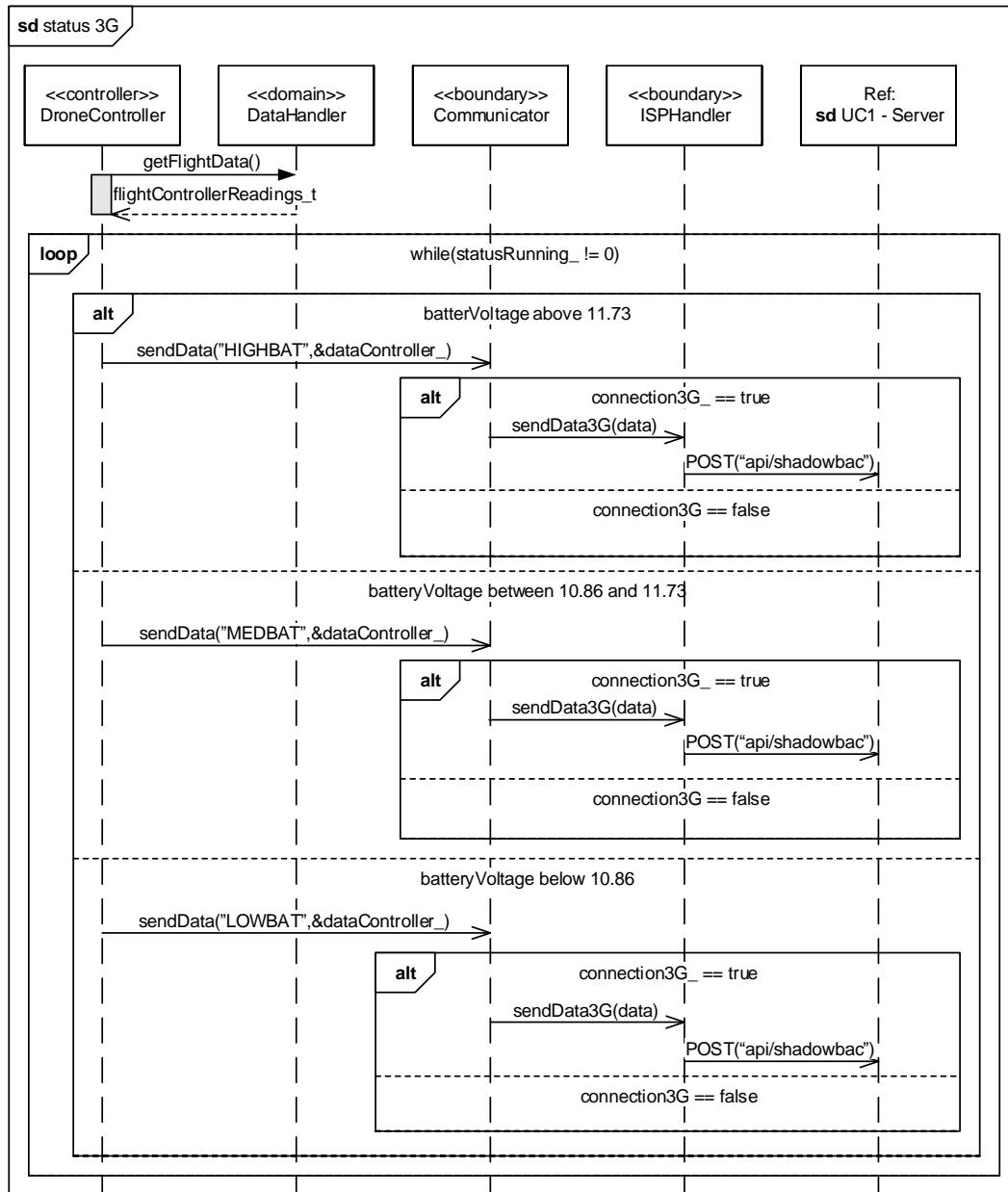
- TakeOff-signal starter drones *takeOff* for at lette dronen.
- Land-signal starter dronens *land* for at lande dronen.
- SetPosition-signal starter dronens *setPosition* for at holde positionen modtaget fra applikationen.
- Follow-signal starter dronens *follow* for at følge applikationen.
- Unfollow-signal starter dronens *setPosition*, for at holde seneste position fra applikationen.
- Stop-signal starter dronens *land* og venter på denne er færdig. Stop signalet er det eneste signal der kan stoppe while løkken i *using3G()*.

Using3G()-funktionen dækker over en stor del af Use Case 3-6. Disse Use Case's vil blive forklaret enkeltvis gennem den tilhørende tråd funktion. Et samlet statisk klassediagram vil blive udarbejdet for *using3G()* samt alle tilhørende tråde.

StatusThread



Figur 3.21: CPU Drone - Konceptuelle klasser status

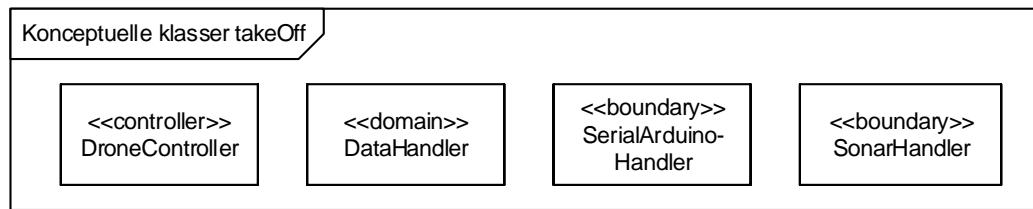


Figur 3.22: CPU Drone - Sekvens diagram status

status-tråden har ansvaret for at sende en status besked til applikationen. Denne henter først dronens data fra *DataHandler*'en for dernæst at sende en besked. Baseret på batterispændingen sendes et af 3 signaler; HIGHBAT, MEDBAT eller LOWBAT. I dette tilfælde sendes beskeden over 3G

3.2.3.3 Use Case 3

Gennem Use Case 3 og domænemodellen for dronen er der opstillet konceptuelle klasser for Use Case 3. Disse konceptuelle klasser kan ses på figur 3.23.



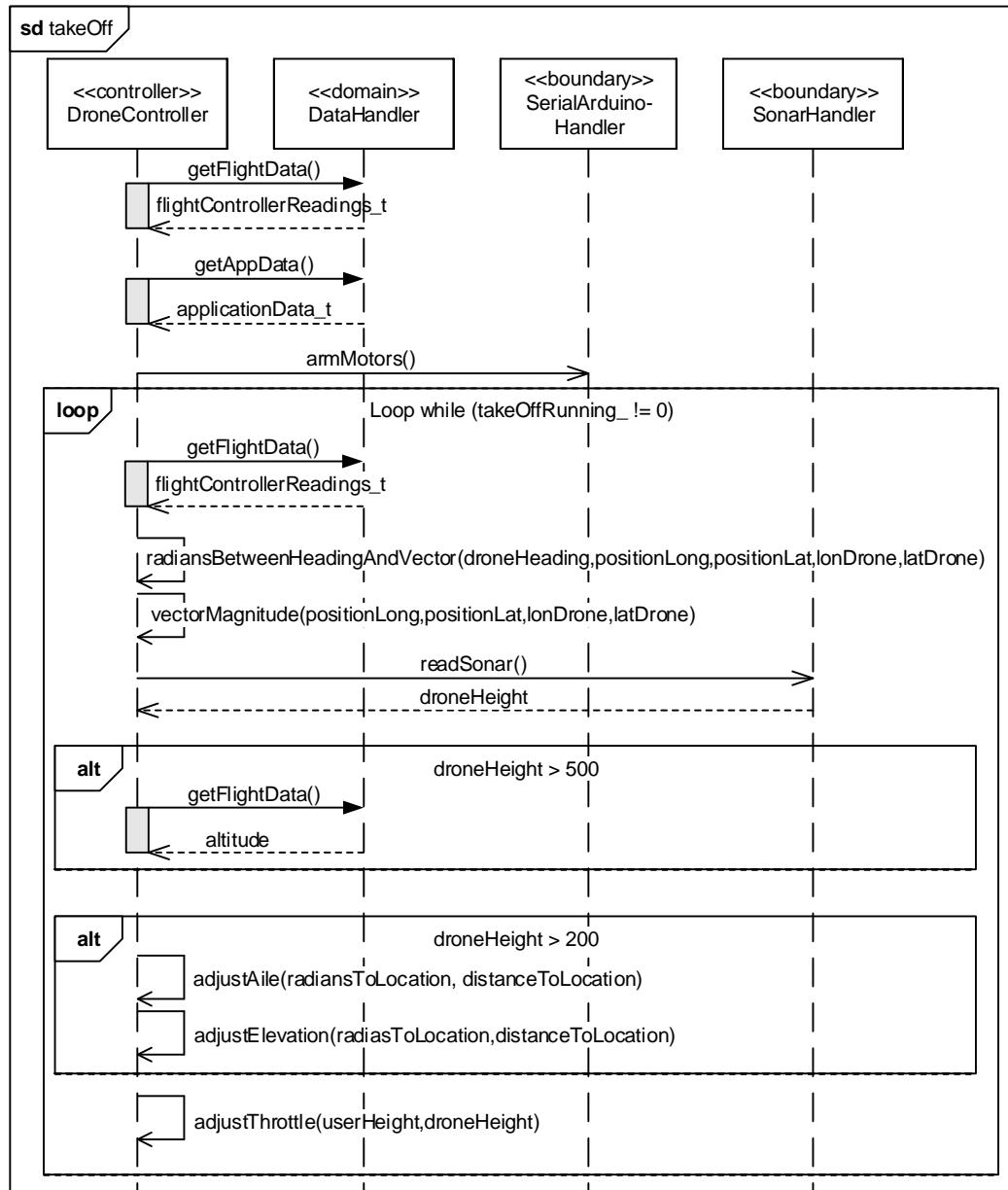
Figur 3.23: CPU Drone - Konceptuelle klasser takeOff

Den nye konceptuelle klasse vil blive forklaret i tabellen 3.3

Navn	Beskrivelsen
SonarHandler	Denne klasse står for at håndtere udlæsningen af data fra sonarsensoren samt konvertering.

Tabel 3.3: Beskrivelser af klasserne i figur 3.23

Baseret på de konceptuelle klasser kan der laves et sekvensdiagram. Sekvensdiagrammet for *using3G()* og sekvensdiagrammet for Use Case 3 på figur 3.24 dækker tilsammen Use Case 3.

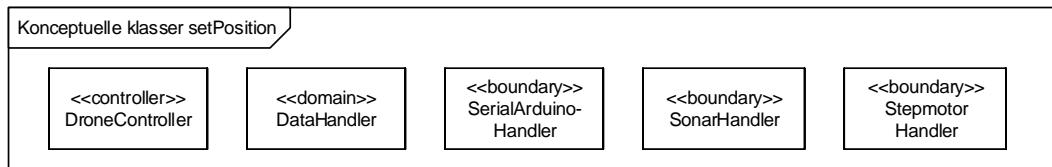


Figur 3.24: CPU Drone - Sekvens diagram takeOff

I *takeOff* tråden hentes først relevant data fra både dronen selv men også applikations seneste besked. Herefter aktiveres dronens motorer. I et loop henter dronen løbende sine data og udregner på baggrund af disse, værdier denne regulere flyvningen med. Hvis højden er lavere end 2 meter benyttes kun *adjustThrottle* til at justere højden. Over 2 meter vil dronen forsøge at holde sig på start koordinatet.

3.2.3.4 Use Case 4

Gennem Use Case 4 og domænemodellen for dronen er der opstillet konceptuelle klasser for Use Case 4. Disse konceptuelle klasser kan ses på figur 3.25.



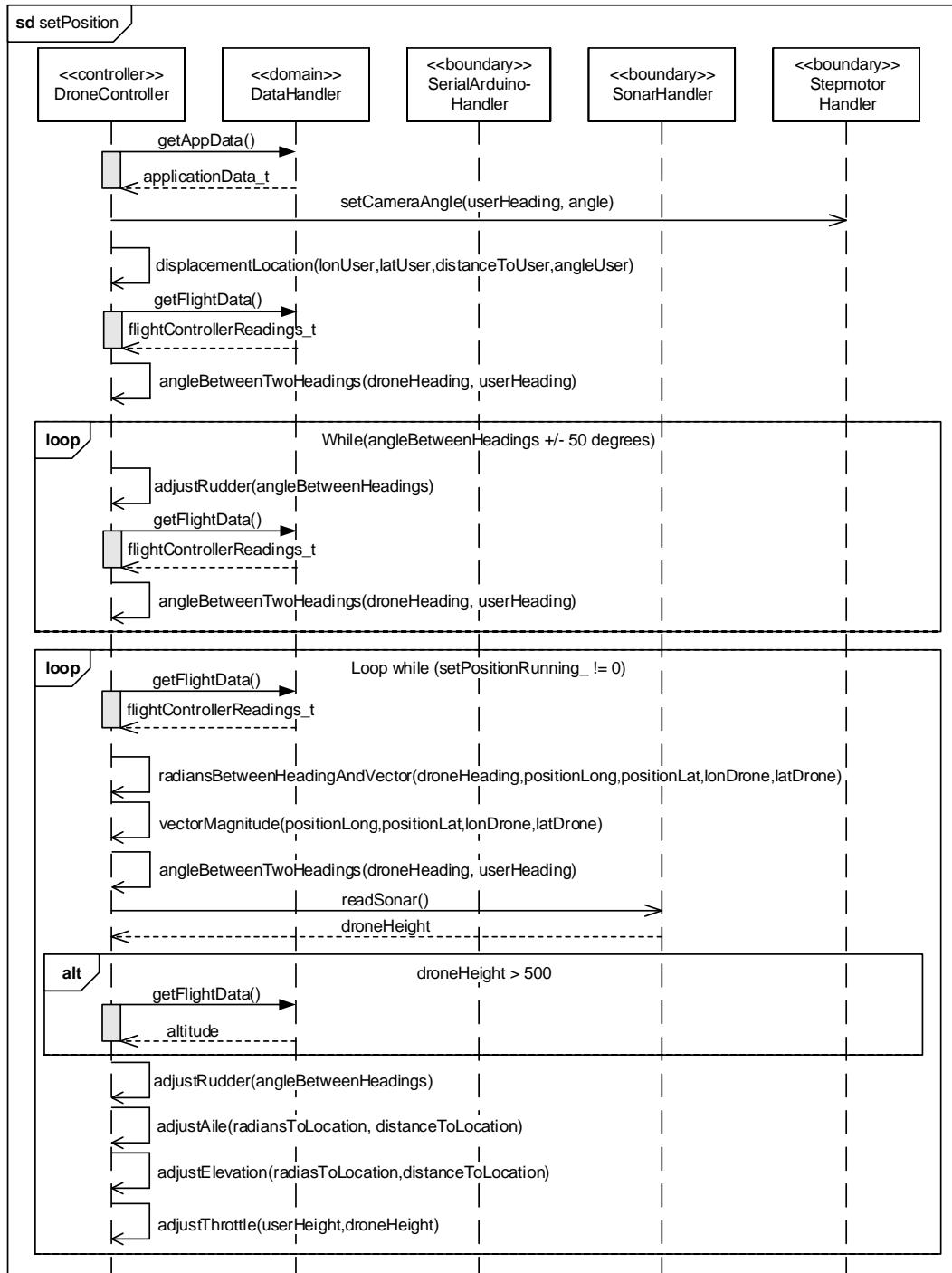
Figur 3.25: CPU Drone - Konceptuelle klasser setPosition

Den nye konceptuelle klasse vil blive forklaret i tabellen 3.4

Navn	Beskrivelsen
StepMotorHandler	Denne klasse står for at håndtere stepmotoren således kameraet kan drejes.

Tabel 3.4: Beskrivelser af klasserne i figur 3.25

Baseret på de konceptuelle klasser kan der laves et sekvensdiagram. Sekvensdiagrammet for *using3G()* og sekvensdiagrammet for Use Case 4 på figur 3.26 dækker tilsammen Use Case 4.

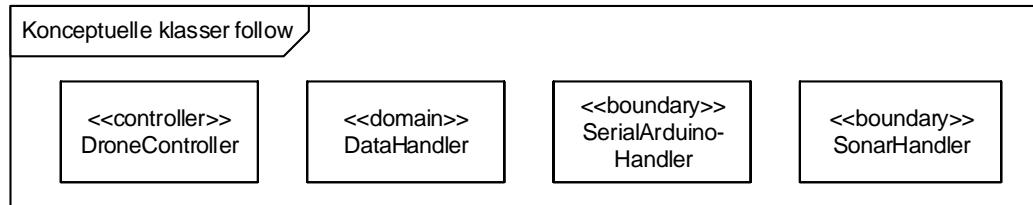
Figur 3.26: CPU Drone - Sekvens diagramm `setPosition`

I `setPosition` hentes der først data fra applikationen. Herefter indstilles kameraet i en given vinkel. Dronen udregner positionen i forhold til applikationen og forskellen i retningen de peger. Dronen begynder derefter at dreje sig selv ind, således at denne har samme retning som applikationen. Når dette er inden for $+/ - 50^\circ$ vil dronen begynde et nyt loop, hvor den også begynder at navigere til punktet. I dette loop udregner dronen løbende forskellige

værdier denne benytter til reguleringen under flyvningen.

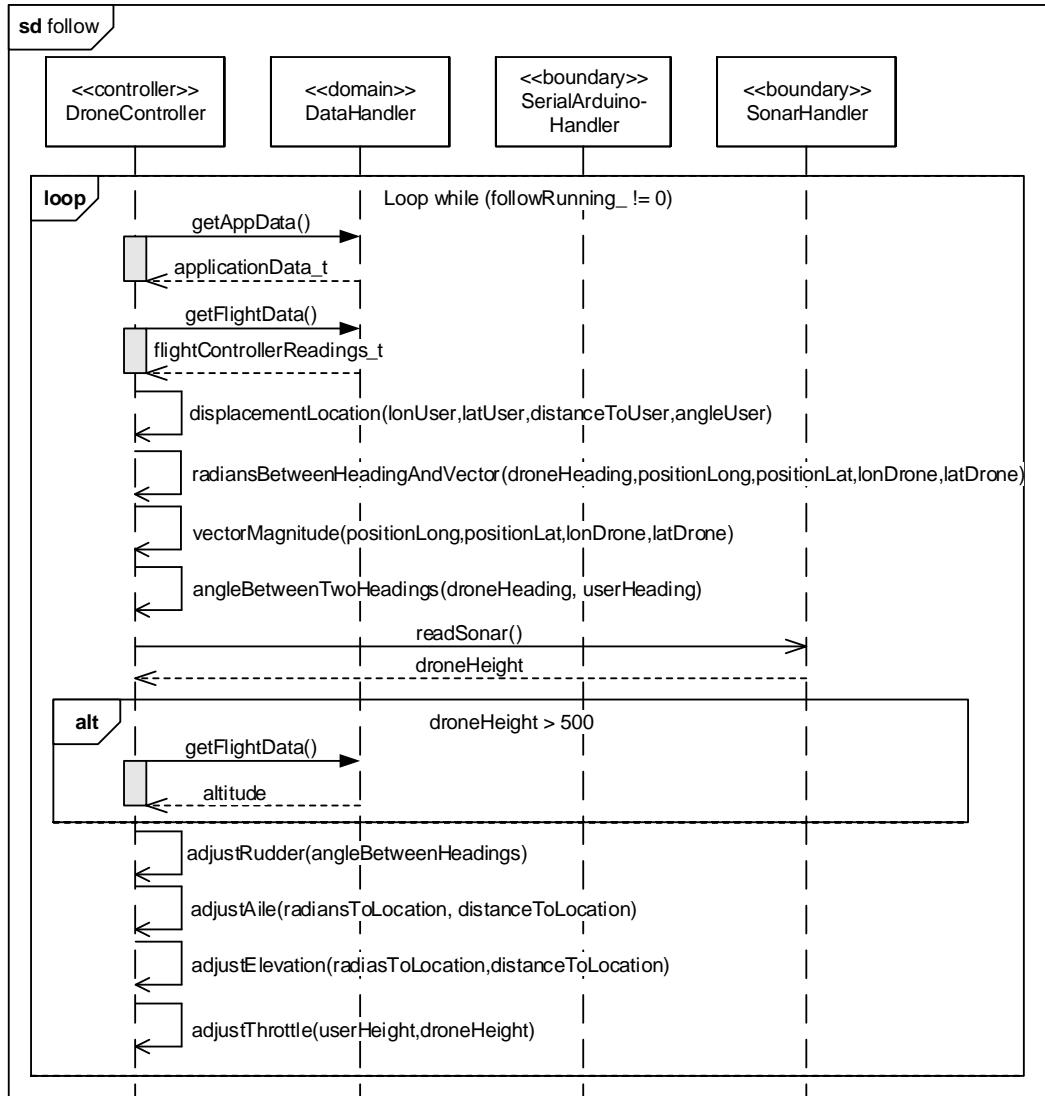
3.2.3.5 Use Case 5

Gennem Use Case 5 og domænemodellen for dronen er der opstillet konceptuelle klasser for Use Case 5. Disse konceptuelle klasser kan ses på figur 3.27.



Figur 3.27: CPU Drone - Konceptuelle klasser follow

Baseret på de konceptuelle klasser kan der laves et sekvensdiagram. Sekvensdiagrammet for *using3G()* og sekvensdiagrammet for Use Case 5 på figur 3.28 dækker tilsammen Use Case 5.

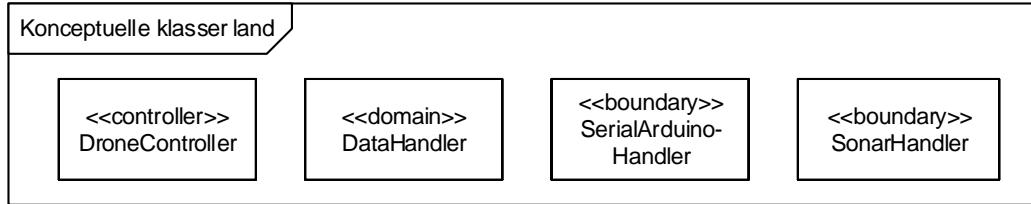


Figur 3.28: CPU Drone - Sekvens diagram follow

I *follow* startes et loop med det samme. Dronen henter derefter dataet modtaget fra applikationen og herefter sin egen data. Fire funktioner, til at udregne forskellige parametre, benyttes således at dronen kan regulere sin flyvning, så den følger applikationen. Skulle højden af dronen være over 5 meter benyttes højden fra barometeret. De fire funktioner der benyttes til at regulere flyvningen er: *adjustRudder*, *adjustAile*, *adjustElevation* og *adjustThrottle*.

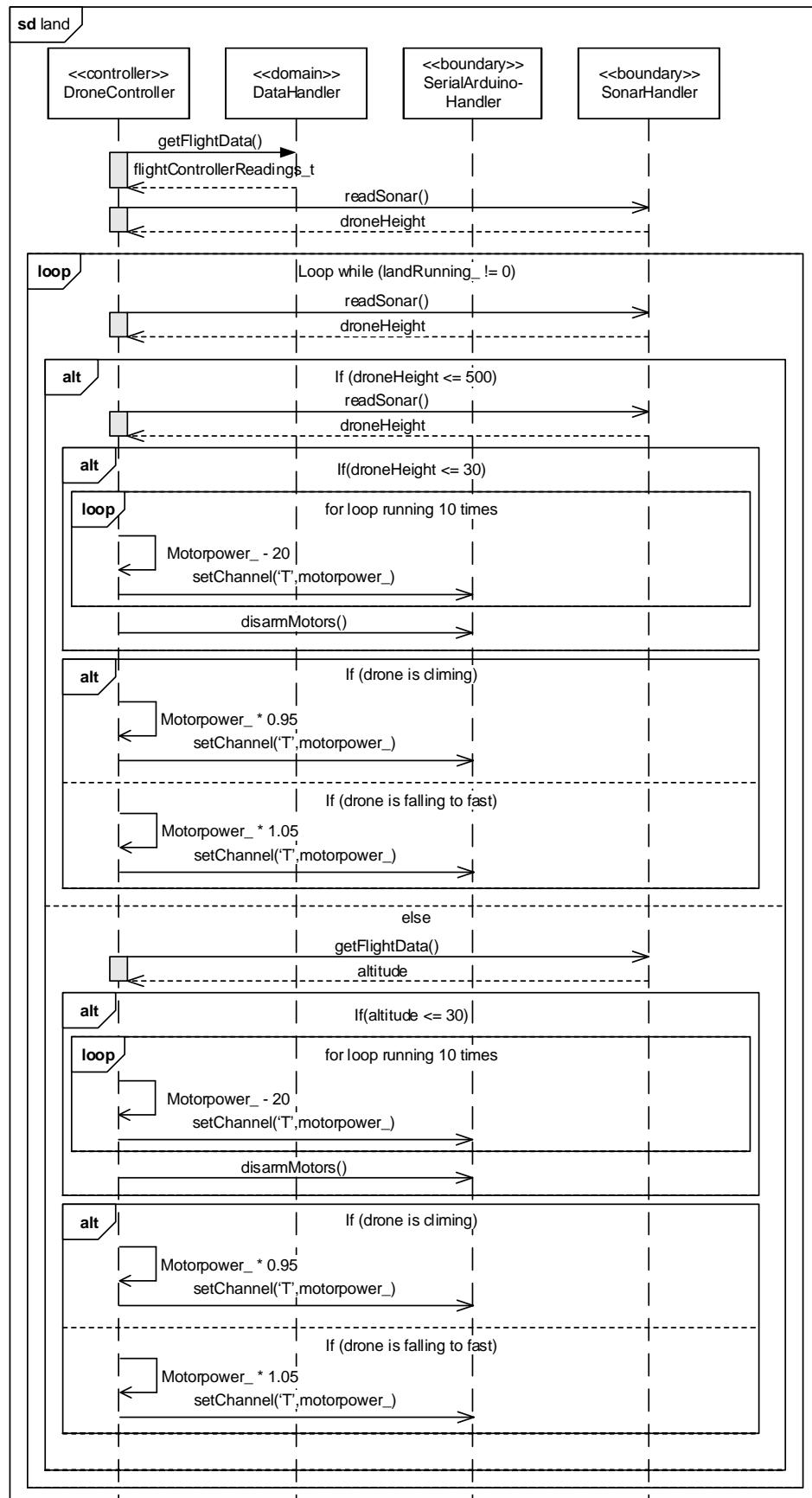
3.2.3.6 Use Case 6

Gennem Use Case 6 og domænemodellen for dronen er der opstillet konceptuelle klasser for Use Case 6. Disse konceptuelle klasser kan ses på figur 3.29.



Figur 3.29: CPU Drone - Konceptuelle klasser land

Baseret på de konceptuelle klasser kan der laves et sekvensdiagram. Sekvensdiagrammet for *using3G()* og sekvensdiagrammet for Use Case 6 på figur 3.30 dækker tilsammen Use Case 6.

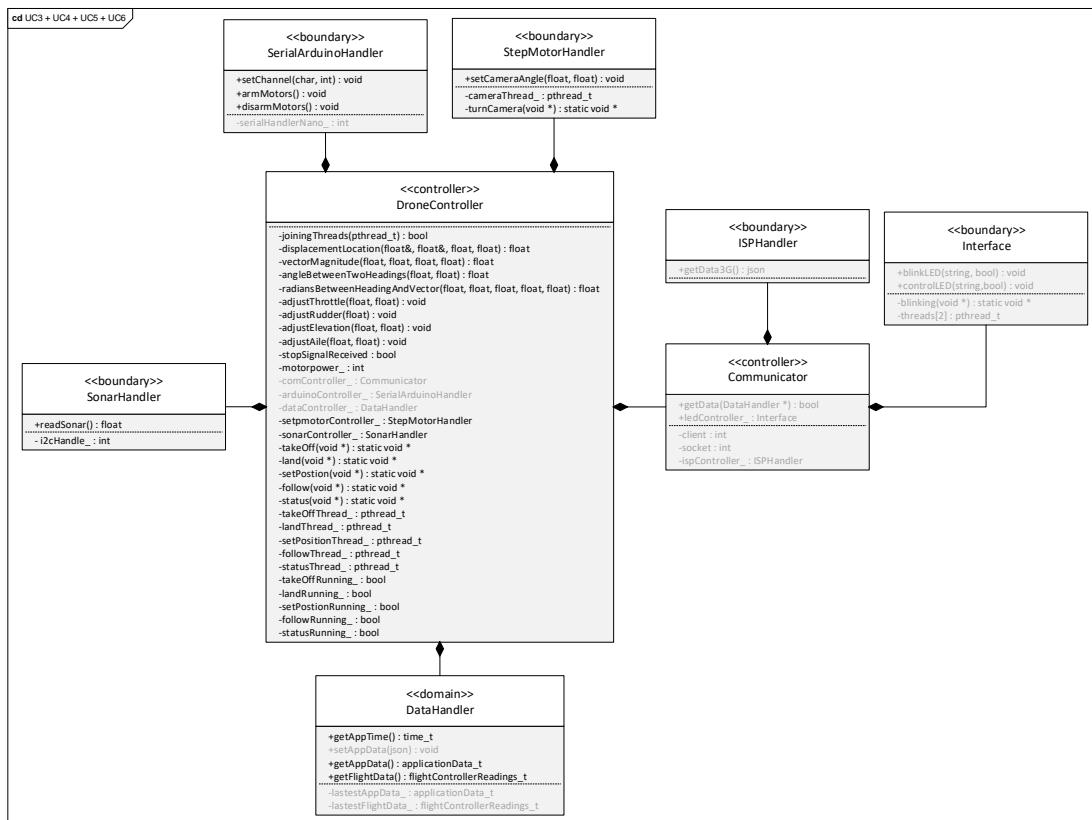


Figur 3.30: CPU Drone - Sekvens diagram land

I land hentes der først relevant data fra dronen selv. Herefter læses højden fra sonarsenoren. Et loop startes hvor sonaren igen aflæses, baseret på sonarhøjden vurderes det hvorvidt barometeret eller sonaren skal benyttes. I begge tilfælde kigger dronen på hvorvidt denne er i et fald eller om denne stiger. Såfremt dronen er i et fald mod jorden gøres ingenting. Hvis dronen falder for hurtigt baseret på højdemålingerne vil motorkraften blive justeret op. Stiger dronen derimod nedjusteres motorkraften. Hvis afstanden til jorden når under 30 centimeter nedjusteres motorkraften betydeligt gennem et for-loop, herefter slukkes motorerne.

3.2.3.7 Statisk klassediagram - using3G(), Use Case 3, 4, 5 og 6

Baseret på *using3G()*, Use Case 3, 4, 5 og 6 er følgende statiske klassediagram udarbejdet.

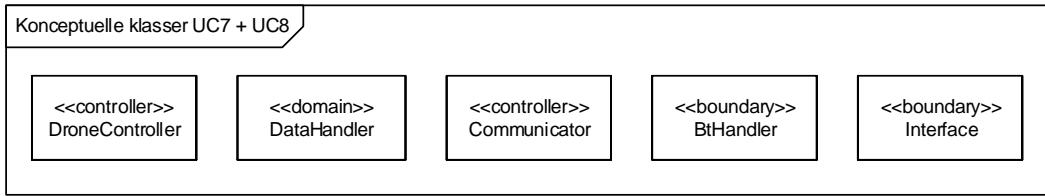


Figur 3.31: CPU Drone - Use Case 3 og 6 - Klassediagram

Det statiske klassediagram viser alle funktionerne der benyttes i systemet på dronen i *using3G()*, Use Case 3, 4, 5 og 6. Funktionerne vist tidligere er skrevet med grå skrift.

3.2.3.8 Use Case 7 og 8

Gennem Use Case 7, Use Case 8 og domænemodellen for dronen er der opstillet konceptuelle klasser for Use Case 7 og 8. Disse konceptuelle klasser kan ses på figur 3.32. Initialiseringen af Bluetooth foregår i funktionen *usingBT()*. Use Case 7 og 8 er derfor samlet med funktionen *usingBT()*.



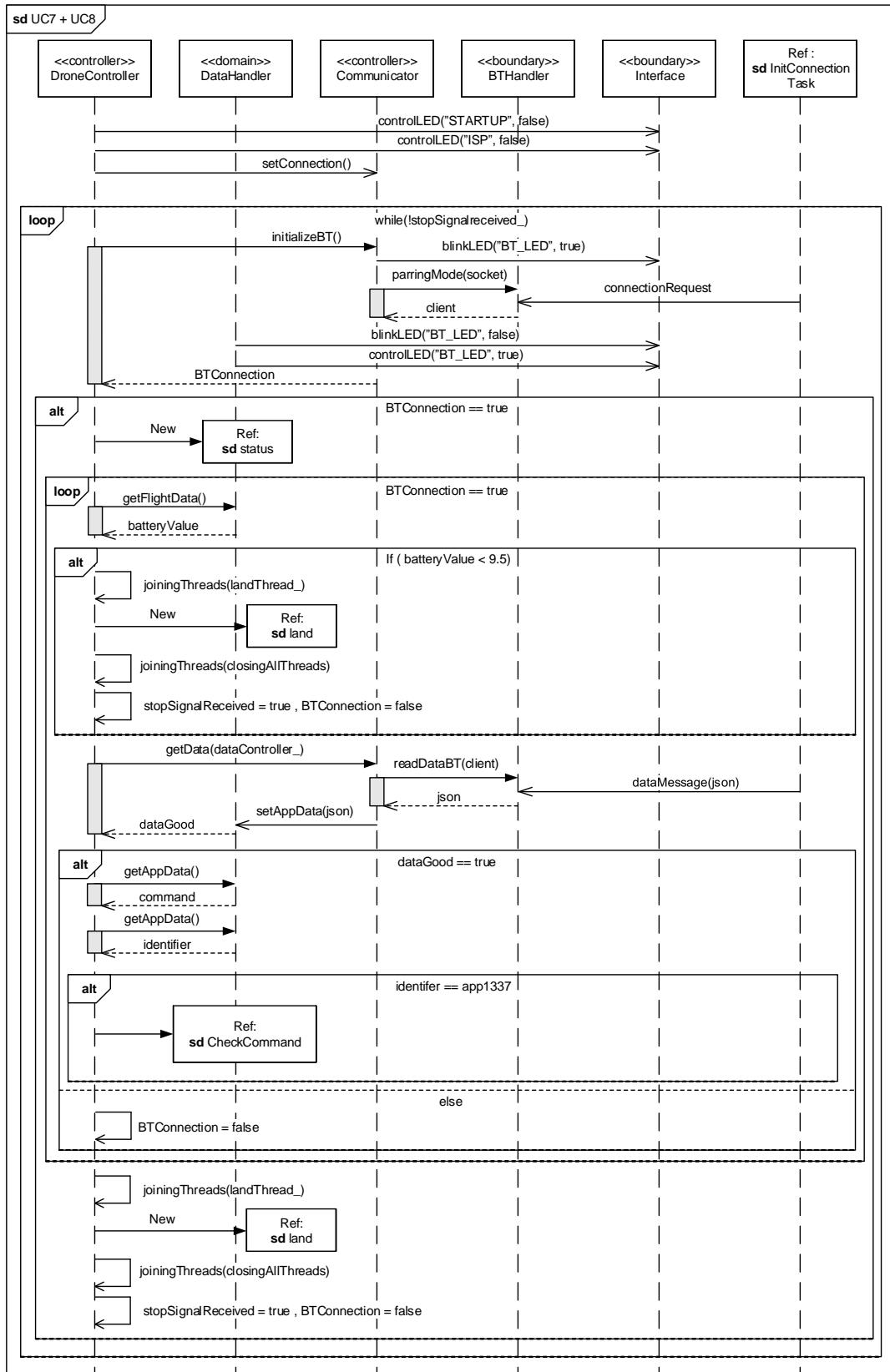
Figur 3.32: CPU Drone - UC7 og UC8 konceptuelle klasser

Den nye konceptuelle klasse bliver forklaret i tabel 3.5

Navn	Beskrivelsen
BtHandler	Denne klasse står for kommunikationen over Bluetooth samt etableringen af denne forbindelse.

Tabel 3.5: Beskrivelser af klasserne i figur 3.32

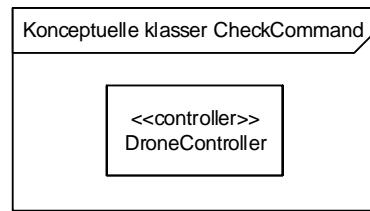
Ud fra de konceptuelle klasser kan der laves et sekvensdiagram. Sekvensdiagrammet dækker både Use Case 7 og Use Case 8 på dronen samlet i funktionen *usingBT()*. Sekvensdiagrammet ses på figur 3.33.



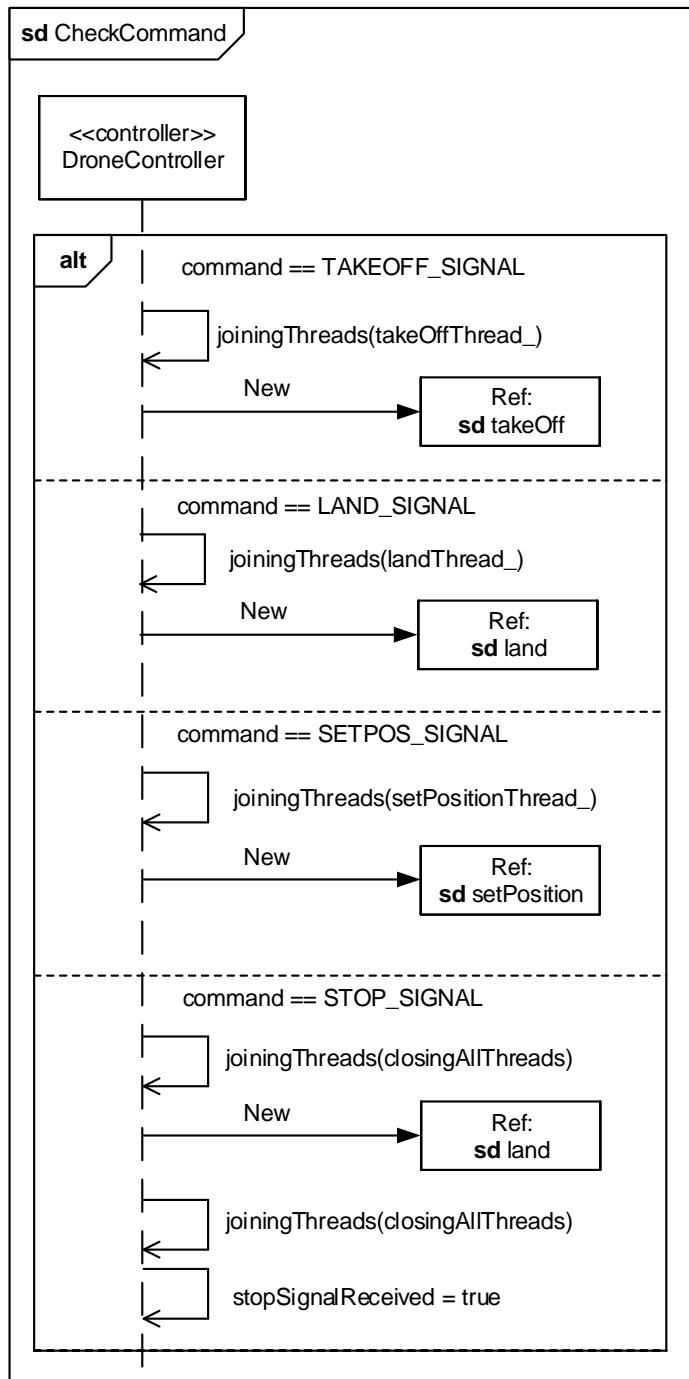
Figur 3.33: CPU Drone - UC7 og UC8 Sekvensdiagram

Denne funktion nåes ved at 3G enten er nede eller at applikationen ikke har sendt et Start-signal inden for 60 sekunder efter opstart. Først sættes kommunikationsformen i *Communicator*'en således at denne skifter til Bluetooth. Herefter initialiseres Bluetooth, dette er et blokerende kald, som venter på en forbindelse fra applikationen. Når denne er oprettet starter *usingBT()*-funktionen *status* således at applikationen begynder at modtage statusbeskeder. Herefter tjekkes batterispændingen. Er denne for lav kaldes *land* og koden lukker ned. Ellers venter Bluetooth på en besked fra applikationen, med den korrekte *identifier*. Skulle dronen miste sin Bluetooth forbindelse kaldes *land* med det samme.

Hvis en korrekt *identifier* er modtaget fra applikationen fortsætter funktionen i sekvensdiagrammet CheckCommand. Den konceptuelle klasse er vidst i figur 3.34. Sekvensdiagrammet ses i figur 3.35

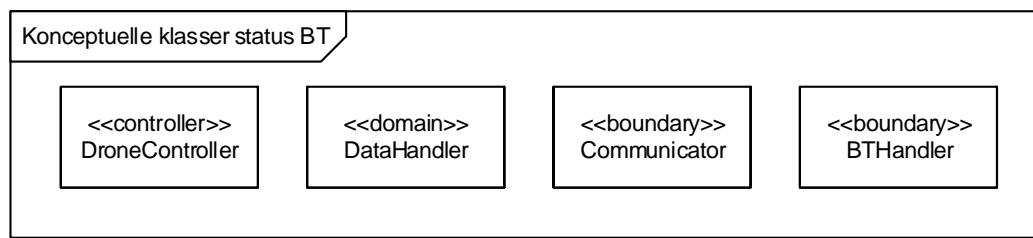


Figur 3.34: CPU Drone - CheckCommand Konceptuelle Klasser

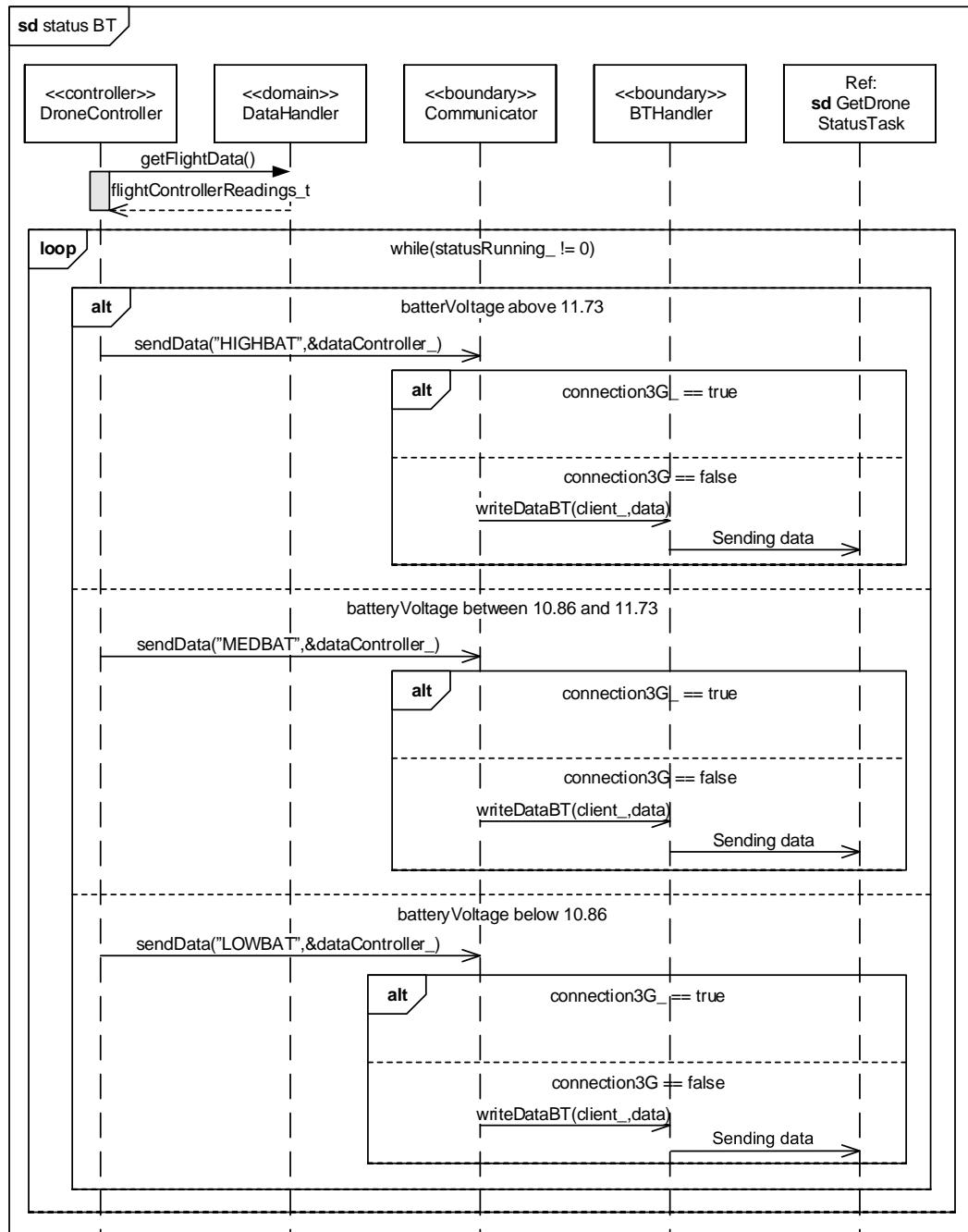


Figur 3.35: CPU Drone - CheckCommand Sekvensdiagram

I dette sekvens diagram, som er et fortsættelse på `usingBT()` validere dronen den modtagne kommando mod dronens muligheder. Såfremt en given kommando er modtaget kaldes den tilhørende tråd.

StatusThread

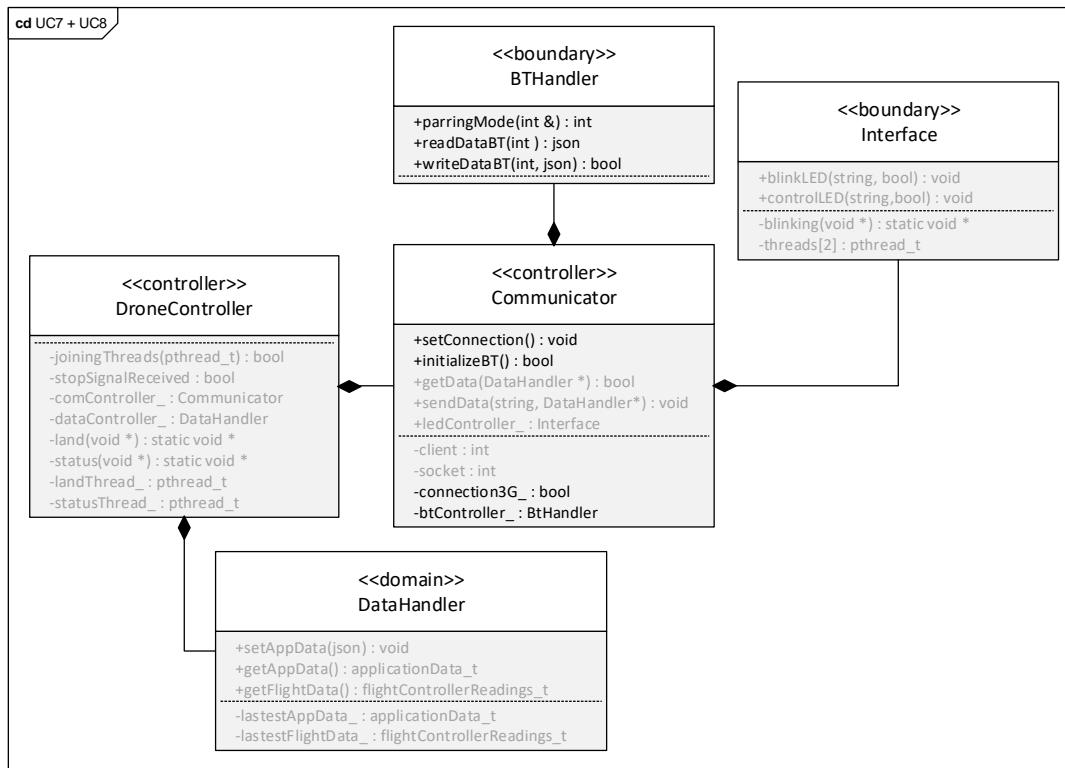
Figur 3.36: CPU Drone - Konzeptuelle klasser status



Figur 3.37: CPU Drone - Sekvensdiagram status

status har ansvaret for at sende en status besked til applikationen. Denne henter først dronens data fra *DataHandler*'en for dernæst at sende en besked. Baseret på batterispændingen sendes et af 3 signaler; HIGHBAT, MEDBAT eller LOWBAT. I dette tilfælde sendes beskederne over Bluetooth.

Baseret på sekvensdiagrammerne kan der opstilles et statisk klasse-diagram for Use Case 7, 8 og *usingBT()*. Dette ses på figur 3.38



Figur 3.38: CPU Drone - UC7 og UC8 Klassediagram

Det statiske klassediagram viser alle funktionerne der benyttes i systemet på dronen i Use Case 7, 8 og *usingBT()*.

3.2.3.9 Use Case 9, 10 og 11

Disse 3 Use Case's dækker over præcis det samme som Use Case 3, 4 og 6. Forskellen i disse Use Case's er kommunikationsformen. Her benyttes Bluetooth hvor der tidligere er benyttet 3G. Håndteringen af Bluetooth ligger i funktionen *usingBT()* se figur 3.33, dette betyder også at trådene benyttet i Use Case 3, 4 og 6 benyttes igen.

- Use Case 9 kalder samme tråd som Use Case 3, se figur 3.24.
- Use Case 10 kalder samme tråd som Use Case 4, se figur 3.26.
- Use Case 11 kalder samme tråd som Use Case 6, se figur 3.30.

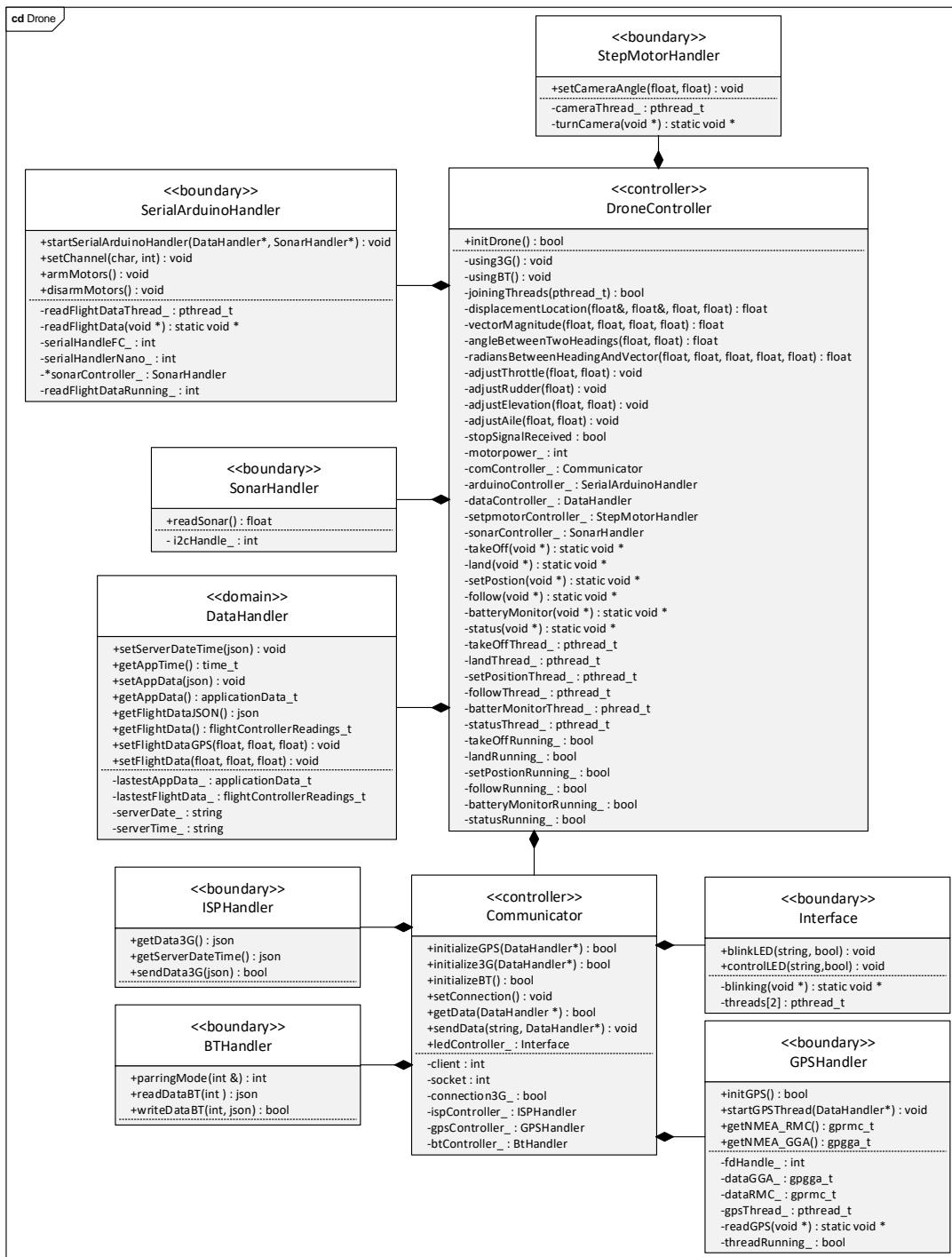
På baggrund af dette vil der ikke blive opstillet et statisk klassediagram for disse Use Case's da funktionaliteten allerede er nævnt tidligere og indgår i det statiske klassediagram på figur 3.31.

3.2.3.10 Arduino Nano

Arduino Nano'en bliver i systemet brugt som mellemmanden mellem dronens Raspberry Pi og FlightController'en. Denne modtager serielle kommandoer fra Raspberry Pi'en og baseret på kommandoer ændrer denne et givent PWM output til FlightController'en. Da denne CPU er en lille del af dronen vil denne ikke blive udarbejdet gennem Use Case's. Arduino Nano's funktionalitet vil blive beskrevet yderligere i afsnit 3.7.3.11 i Implementation View.

3.2.3.11 Samlet klassediagram for CPU'en på dronen

Baseret på de statiske klassediagrammer udarbejdet gennem dronens Logical View kan et samlet klassediagram opstillet. Dette er vist på figur 3.39.



Figur 3.39: CPU Drone - Samlet klassediagram for dronen

3.2.4 Applikation

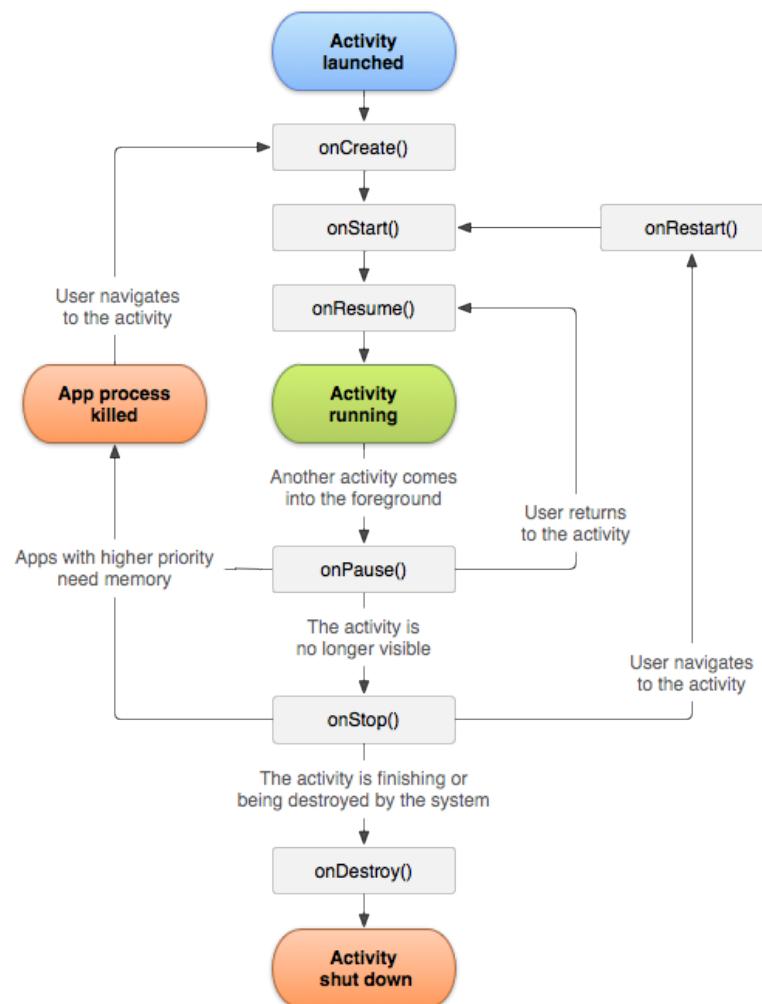
I dette afsnit vil Logical View blive udarbejdet for den CPU, der styrer applikationen. For at give en bedre forståelse for hvordan en applikation er opbygget og de dermed følgende konsekvenser for designet, vil en applikations livscyklus og generelle opbygning først blive gennemgået.

3.2.4.1 Opbygning af en applikation

De to vigtigste komponenter i en applikation er activities og services.

Activity[7]

En activity er ansvarlig for at oprette et skærmbillede til brugeren (typisk ud fra en tilhørende xml fil) og reagere på brugerens input. En vigtig egenskab af activities er deres livscyklus-metoder, som bliver kaldt alt efter i hvilket stadie aktiviteten befinder sig:



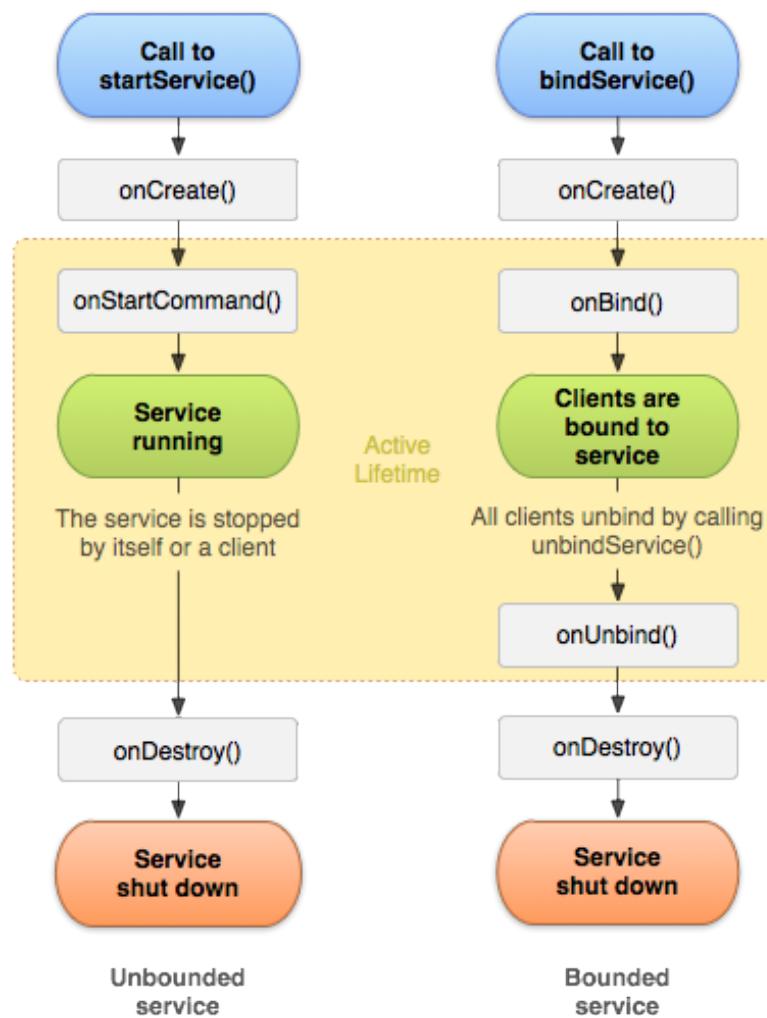
Figur 3.40: Livscyklus-metoder for en activity

Under udviklingen af applikationen skal der altså tages hensyn til at brugeren til enhver tidspunkt kan lukke applikationen. Der skal ryddes op efter brugeren, så der ikke er noget hukommelse der leaker. Derudover skal det vigtigste data gemmes, da dette skal bibeholdes

efter applikationen er lukket ned. En meget vigtig detalje er at en activity bliver nedlagt og oprettet igen, når brugeren drejer sin telefon og orienteringen på skærmen bliver ændret. Alle variabler der ikke aktivt bliver gemt, bliver altså sat tilbage til deres oprindelige tilstand.

Service[8]

En service kører i applikationens baggrund og er ikke nødvendigvis direkte tilknyttet til en activity. Som udgangspunkt kører den dog på samme tråd som aktiviteten. Det er muligt at starte en service på forskellige måder. I denne applikation bliver servicen både startet og der laves en *bind* til servicen. Når servicen er startet, bliver den først afsluttet, når der kaldes *stopService()*. Servicen bliver i denne applikation stoppet, når applikationen lukkes helt ned og dronen er på jorden. Udeover at servicen bliver startet, laves der også et *bind* til servicen. Fordelen ved dette er at aktiviteten kan kalde servicens public metoder. På figur 3.41 ses de forskellige livscykler for en *bound* og en startet service.



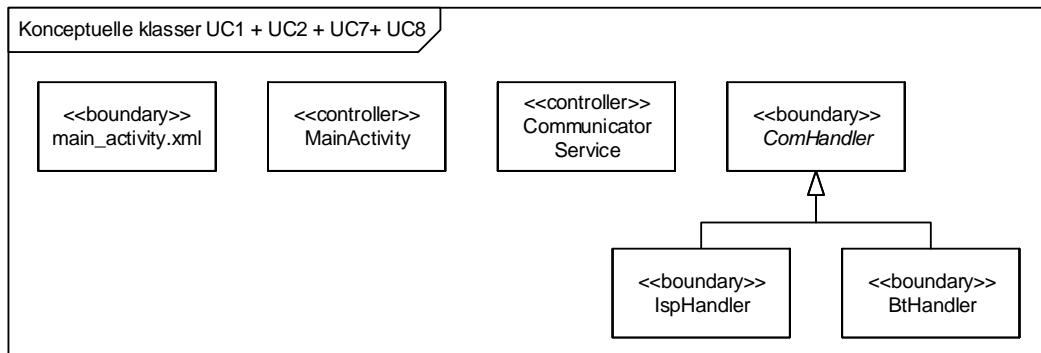
Figur 3.41: Livscyklus-metoder for en service

Da brugeren til et hvert tidspunkt kan lukke applikationen, vil det i de efterfølgende sekvensdiagrammer ikke blive vidst da dette vil føre til uoverskuelige diagrammerne. Der vil dog blive gjort rede for den funktionalitet, der håndterer en nedlukning af applikationen

i de forskellige Use Case's.

3.2.4.2 Use Case 1, 2, 7 og 8

Use Case 1, 2, 7 og 8 er alle Use Case's, hvor der oprettes en forbindelse til dronen. Da disse Use Case's hænger sammen vil der blive udarbejdet et samlet klassediagram for disse. Ud fra Use Case 1, 2, 7, 8 og domænemodellerne blev der fundet følgende konceptuelle klasse.



Figur 3.42: CPU Applikation - UC1, UC2, UC7 og UC8 konceptuelle klasser

Der ses at *main_activity.xml* i virkeligheden ikke er en klasse, men en XML fil. Den indeholder designet af brugerinterfacet. Når brugeren trykker på en knap, bliver et callback kaldt fra Android systemet, som bliver fanget af en *OnClickListener* [9].

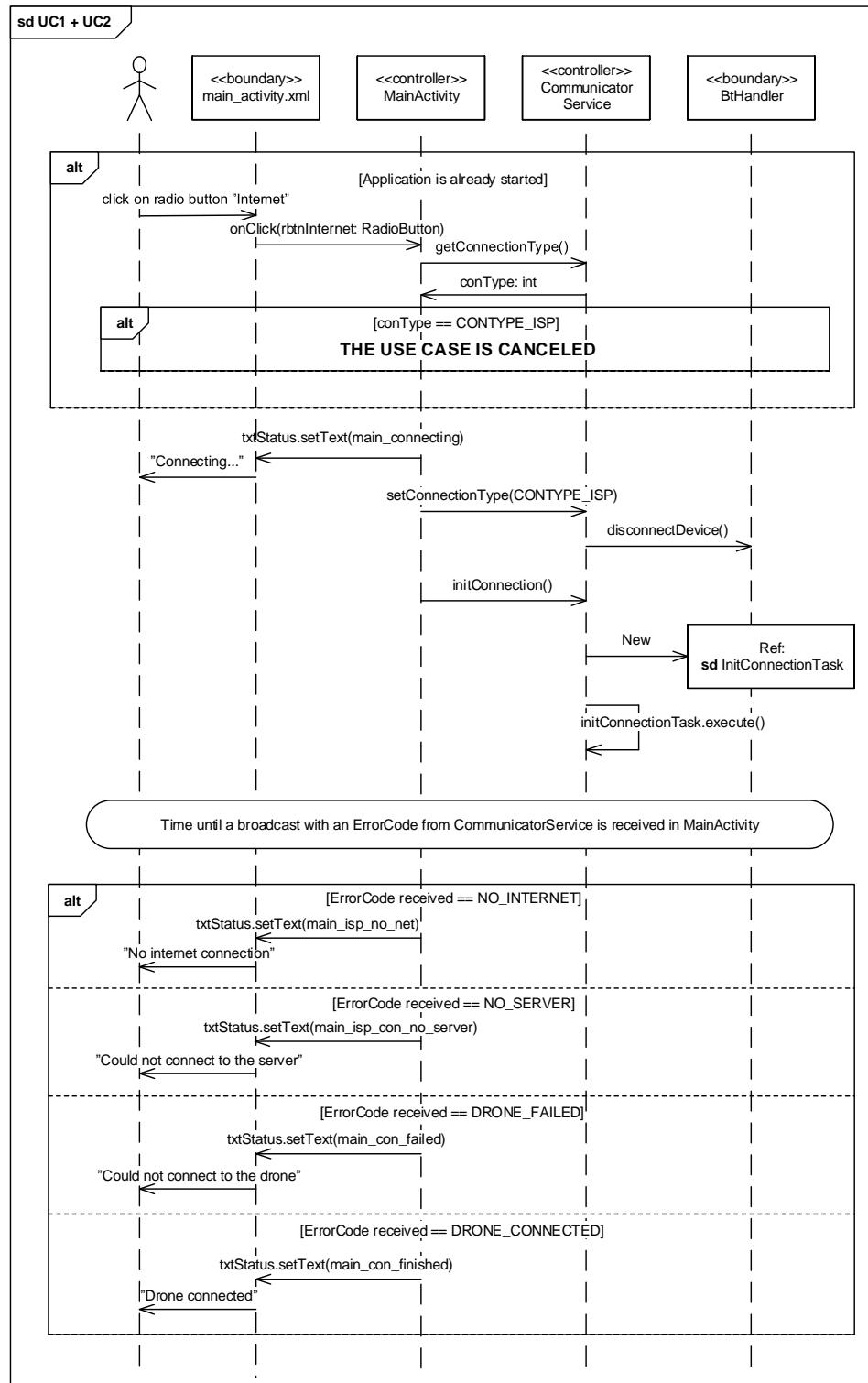
Navn	Beskrivelsen
main_activity.xml	MainActivities interface beskrevet i en xml fil.
MainActivity	Controller klassen, der håndterer input fra brugeren og kommunikerer med servicen.
CommunicatorService	Styrer alt kommunikation med dronen i applikationens baggrund og sender beskeder tilbage til MainActivity.
ComHandler	En abstrakt klasse, de andre kommunikations-handler arver fra. Dette fører til et fælles interface og mere genbrugelig funktionalitet.
BtHandler	Ansvar for at kommunikere over Bluetooth.
IspHandler	Ansvar for at kommunikere over internettet.

Tabel 3.6: Beskrivelser af klasserne i figur 3.42

Ud fra de konceptuelle klasser laves et sekvensdiagram over Use Case 1 og 2 og et sekvensdiagram over Use Case 7 og 8. Det blev valgt at vise to Use Case's i samme sekvensdiagram, da Use Case 1 og 7 hovedsagelig beskriver dronens side af

kommunikationen, hvorimod Use Case 2 og 8 hovedsagelig behandler applikationens side. Ikke alle konceptuelle klasser er i disse sekvensdiagrammer, da noget af funktionaliteten ligger i en baggrunds-tasks, der bliver beskrevet i diagrammerne 3.45 og 3.46. Alle systemts baggrunds-tasks arver fra Android klassen *AsyncTask* [10].

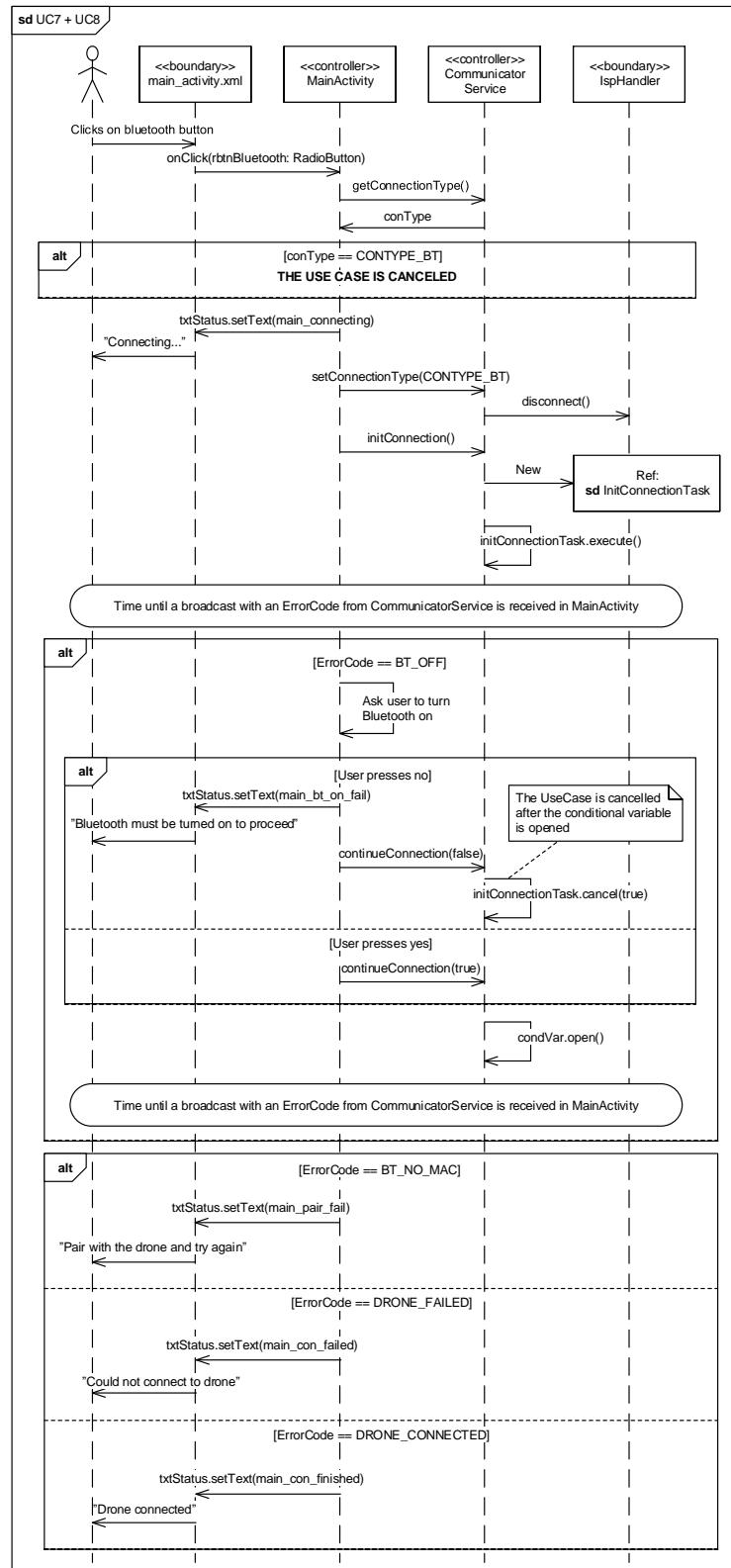
Sekvensdiagram 3.43 dækker over Use Case 1 og 2.



Figur 3.43: CPU Applikation - UC1 og UC2 sekvensdiagram

Øverst på sekvensdiagrammet, ses det at Use Case'en bliver afbrudt, når *CommunicatorService* returnerer, at forbindelsestypen er ISP. Dette bliver gjort, da Internet knappen ikke skal kunne trykkes flere gange, da dette ikke er en normal knap, men en RadioButton.

Denne fungerer som en switch mellem internet og Bluetooth. Derudover ses at Bluetooth forbindelsen bliver afbrudt, hvis der ønskes at forbinde over internettet. Understående er sekvensdiagram 3.44, der beskriver Use Case'ene 7 og 8.



Figur 3.44: CPU Applikation - UC7 og UC8 sekvensdiagram

På sekvensdiagrammet ses at brugeren bliver spurgt om han vil tænde for Bluetooth. Dette sker ved hjælp af et dialog vindue, som efter man sender et *startActivityForResult*-meddelelse.

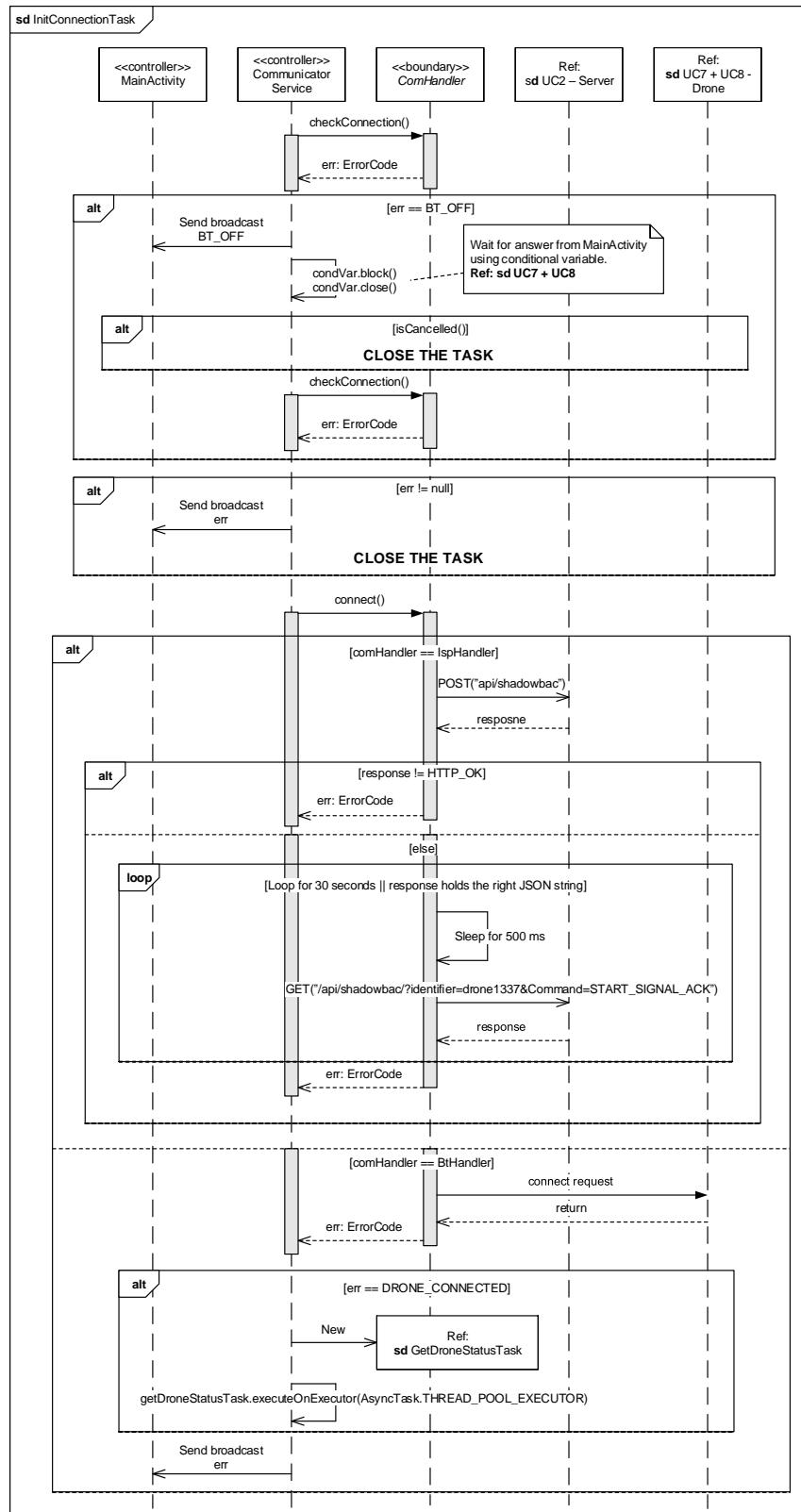
sult(...)-request [11] bliver genereret af Android. *MainActivity* får svar fra dialog vinduet ved hjælp af en callback funktion.

For at håndtere orientation skift, bruges der en enum, som gemmer dronen connection status. Denne enum bliver gemt ved hjælp af *DataHandler*-klassen i *SharedPreferences* [12], som gemmer værdierne når mobilen drejes. Der kan læses mere om *DataHandler*-klassen i afsnit 3.4.3.

Udover funktionaliteten vist i sekvensdiagrammet, får brugeren mulighed for at genstarte Use Case'en, når den ikke bliver afsluttet succesfuld, ved at trykke på en knap der bliver vist når Use Case'en fejler.

For at sende og modtage broadcast beskederne fra *CommunicatorService* til *MainActivity*, bruges der en *LocalBroadcastManager* [13]. *MainActivity* registrerer og afregistrerer denne broadcast i henholdsvis dens *onCreate* og *onDestroy* metoder.

Som beskrevet før er noget af funktionaliteten i baggrund-tasks. Understående er sekvensdiagram 3.45 for funktionaliteten i *InitConnectionTask*.

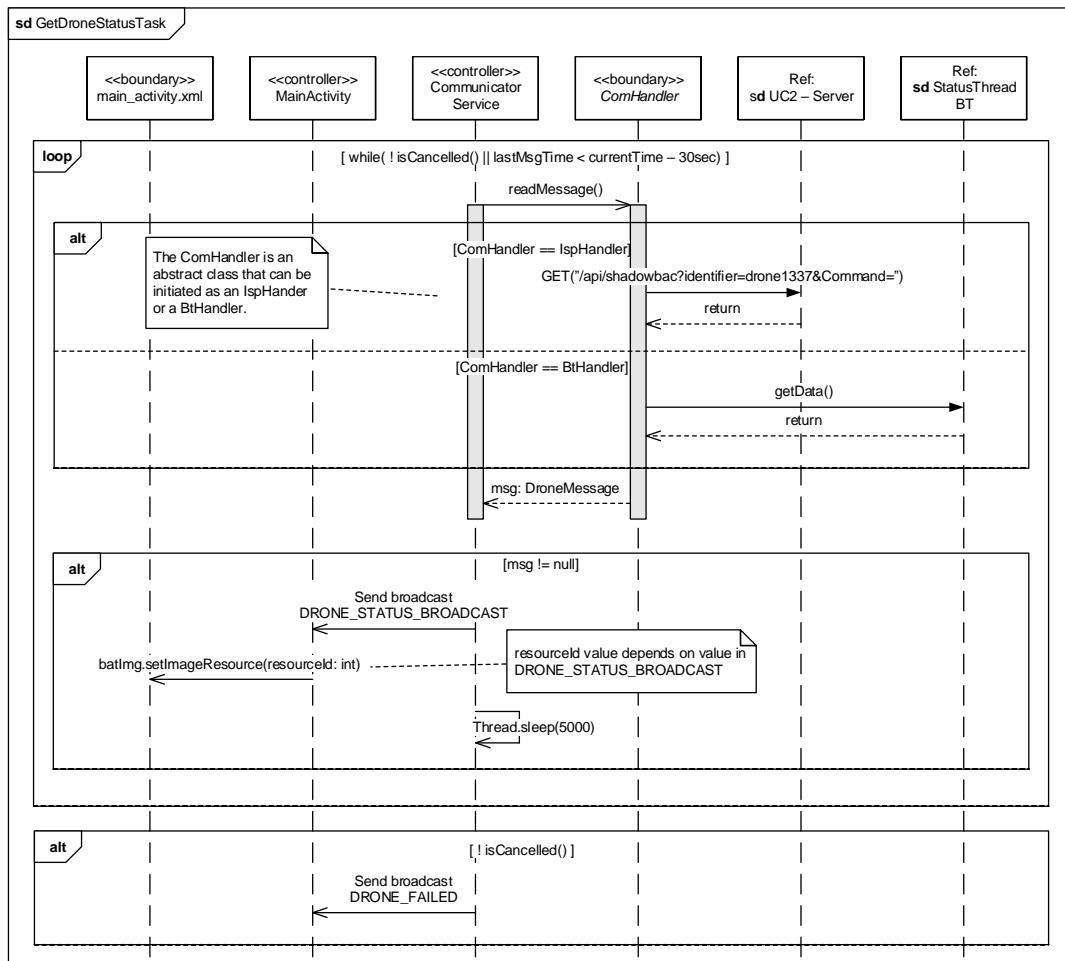


Figur 3.45: CPU Applikation - sekvensdiagram over InitConnectionTask

Det ses at der i slutningen af tasket at *GetDroneStatusTask* bliver startet. Det ses at

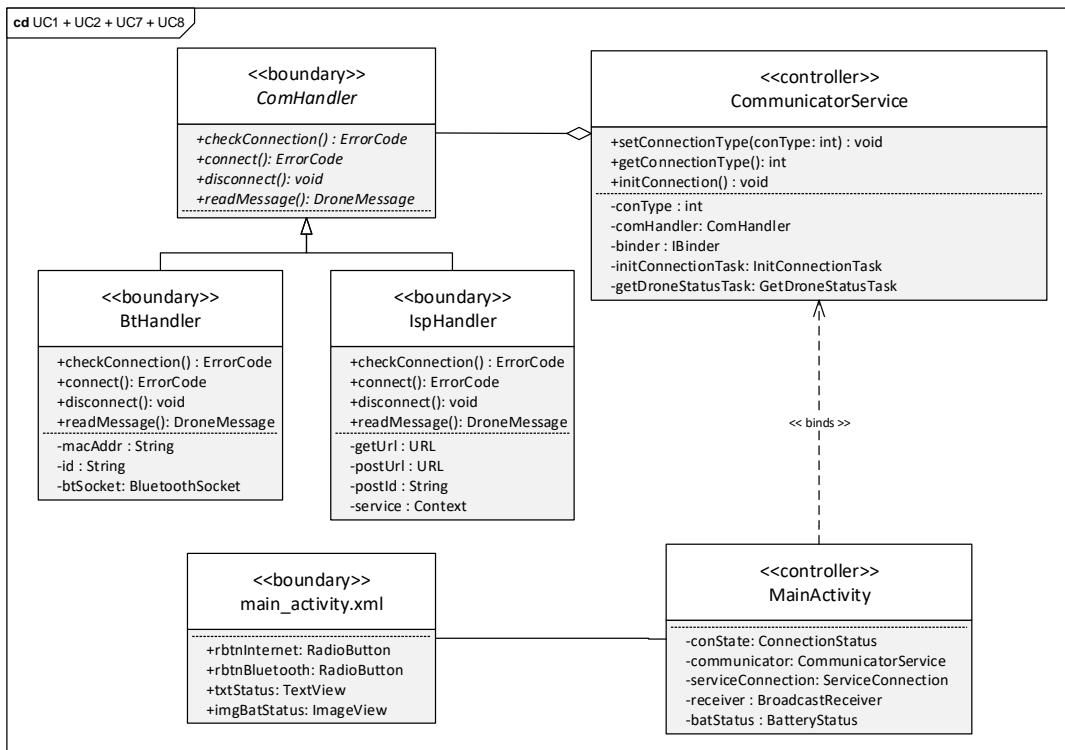
denne task ikke bliver startet med samme metode som *InitConnectionTask*. Dette skyldes at *execute()* funktionen ville starte tasket i samme tråd som alle andre baggrundstask. Da denne task dog skal køre permanent og parallelt med de andre baggrundstasks, startes den ved hjælp af en *Executor*, som fører til at den kører på sin egen tråd.

Underst  ende p   figur 3.46 er sekvensdiagrammet for funktionaliteten i GetDroneStatusT-task.



Figur 3.46: CPU Applikation - sekvensdiagram over GetDroneStatusTask

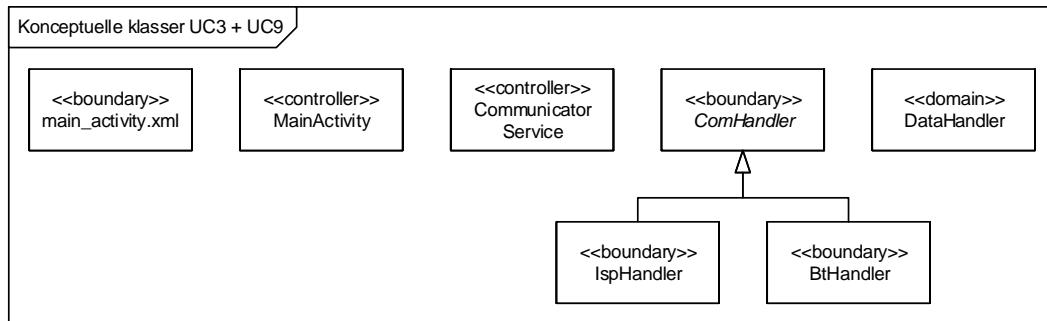
Ud fra sekvensdiagrammerne blev der opstilt f  lgende klassediagram der ses p   figur 3.47.



Figur 3.47: CPU Applikation - UC1, UC2, UC7 og UC8 Klassediagram

3.2.4.3 Use Case 3 og 9

Ud fra Use Case 3, 9 og domænemodellen for systemet blev der fundet følgende konceptuelle klasser.

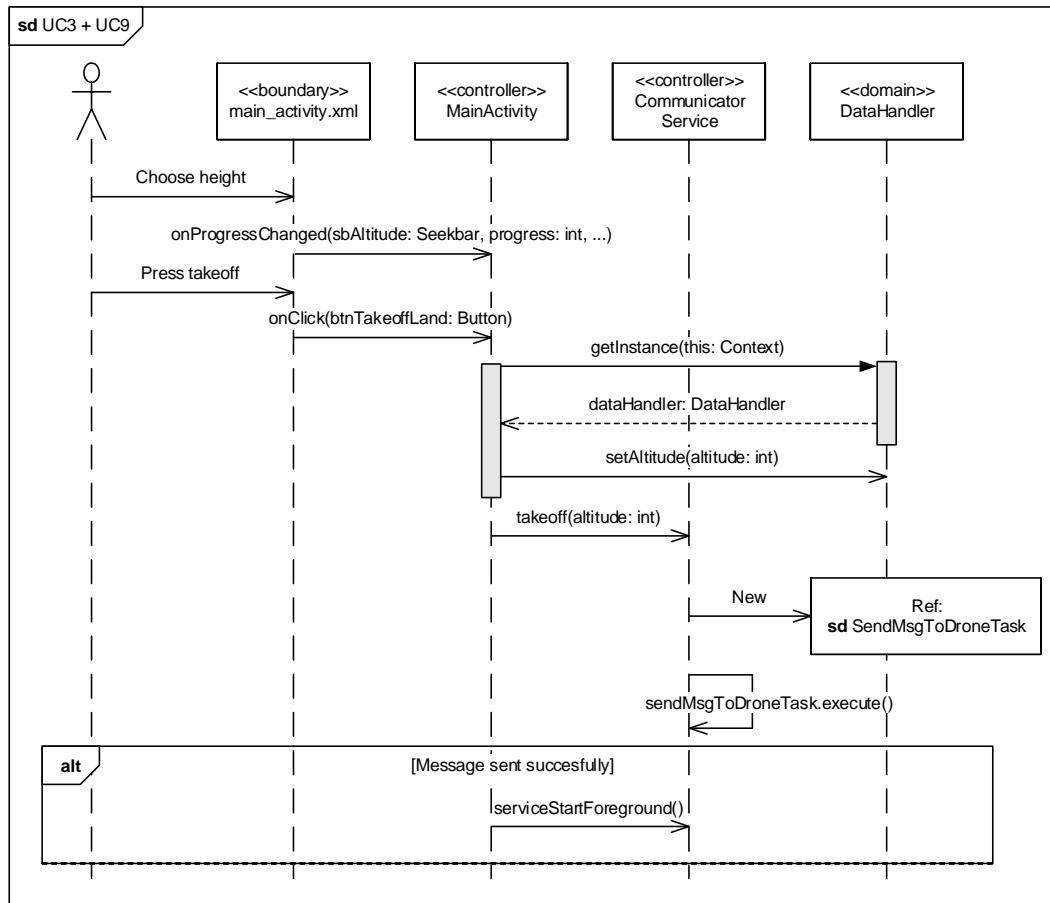


Figur 3.48: CPU Applikation - UC3 og UC9 konceptuelle klasser

Navn	Beskrivelsen
DataHandler	DataHandler er en singleton klasse, der står for at skrive og læse fra mobiltelefons interne memory.

Tabel 3.7: Beskrivelser af klasserne i figur 3.48

Grundet *ComHandler*'en, kan de to Use Case's behandles i det samme sekvensdiagram, da den eneste forskel på de to Use Case's er, hvilken kommunikations handler der skal bruges. Ikke alle konceptuelle klasser er inkluderet i det efterfølgende klassediagram, da en del af Use Case'ens funktionalitet bliver beskrevet i *SendMsgToDroneTask*'s sekvensdiagram, der vises på figur 3.50.



Figur 3.49: CPU Applikation - UC3 sekvensdiagram

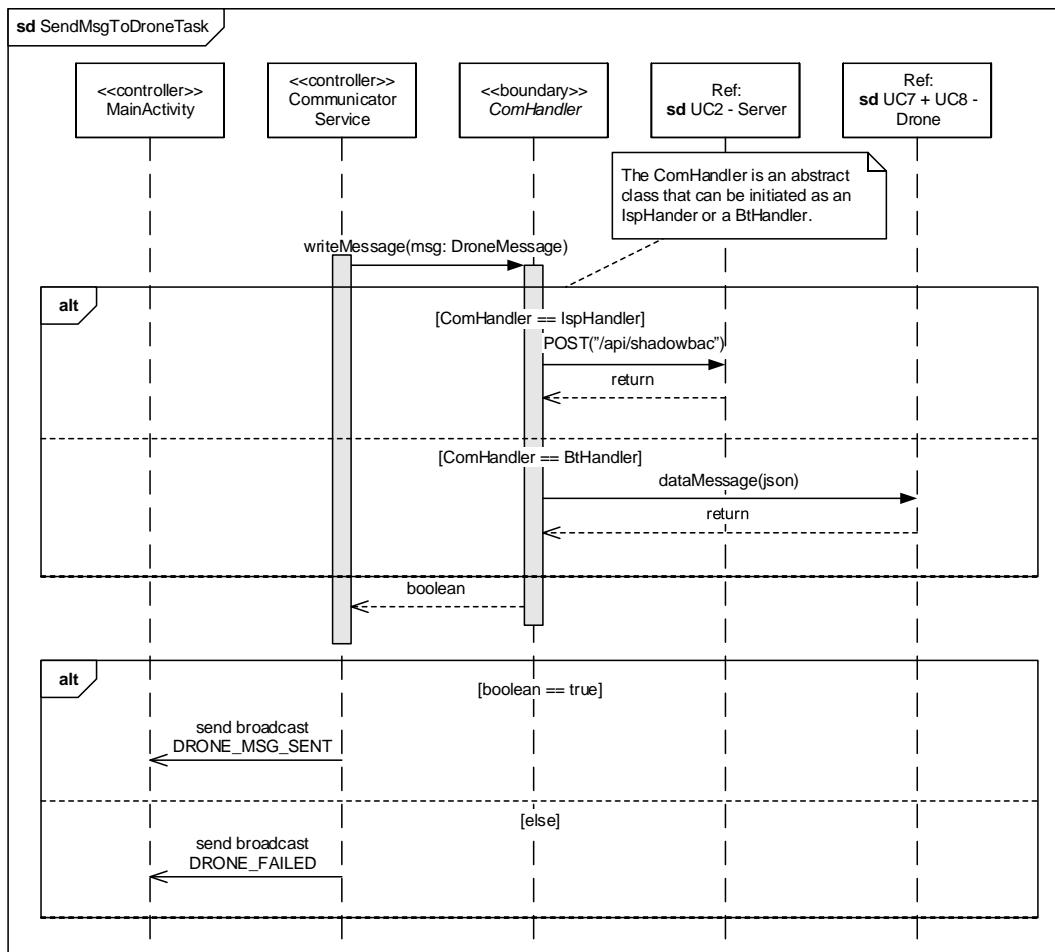
Det ses at når brugeren vælger en højde, at funktionen *onProgressChanged(...)* [14] bliver kaldt. Dette er en callback funktion i Android systemet, som bliver kaldt når værdien af en *SeekBar* [15] ændres.

Det ses på sekvensdiagrammet at navnet på knappen, som trykkes ved TakeOff er *btnTakeoffLand*. Knappen er den samme, der bruges når dronen skal lande. Der bliver dog løbende skiftet udseende, alt efter i hvilket stadie dronen er og om knappen er aktiv.

For at huske dronens status bruges variablen *droneStatus*, som er af typen *DroneStatus*. Dette er en enum, som bliver brugt til at holde styr på dronens tilstand og dermed hvordan user interfacet skal se ud. Denne variabel bliver gemt i mobiltelefonens interne hukommelse ved hjælp af *DataHandler*-klassen. På denne måde bliver dronens status gemt når applikationen lukkes ned.

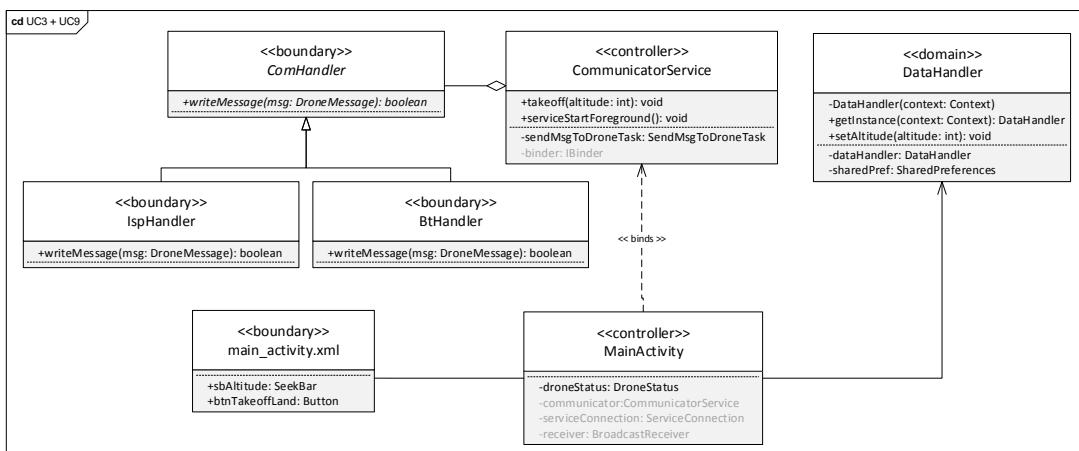
Det ses at servicen startes i forgrunden, når dronen letter. Services der kører i forgrunden bliver ikke lukket ned af styresystemet, når der er mangel på hukommelse. Ud over dette, vil der være en notifikation i mobilens status-bar, som fortæller brugeren at dronen stadig er i luften.

Det ses at selve kommunikationen med dronen sker i en separat baggrunds task. Dette blev valgt, da mange Use Case's skal bruge samme funktionalitet, og tasket dermed kan genbruges. På figur 3.50 ses sekvensdiagrammet over funktionaliteten i *SendMsgToDroneTask*.



Figur 3.50: CPU Applikation - *SendMsgToDroneTask* sekvensdiagram

Ud fra sekvensdiagrammerne blev følgende klassediagram opstillet, der ses på figur 3.51.

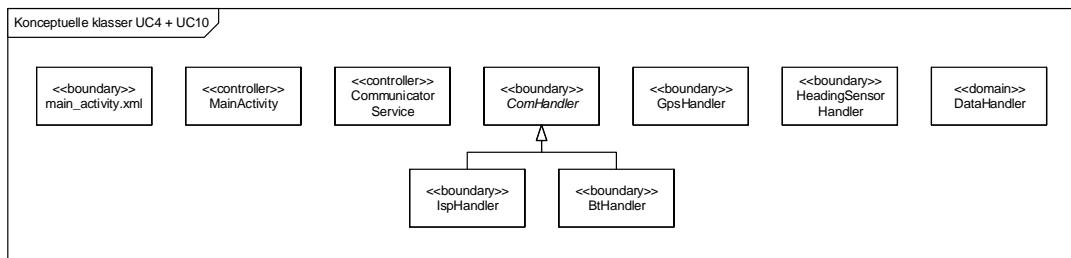


Figur 3.51: CPU Applikation - UC3 Klassediagram

På klassediagrammet ses den nye funktionalitet, der blev tilføjet til systemet i Use Case 3 og 9 samt funktionaliteten for *SendMsgToDroneTask*. I *DataHandler* er det valgt at vise constructoren, som er private, for at understrege at dette er en singleton klasse.

3.2.4.4 Use Case 4 og 10

Ud fra Use Case'ene og systemets domænemodel, blev der fundet følgende konceptuelle klasser.

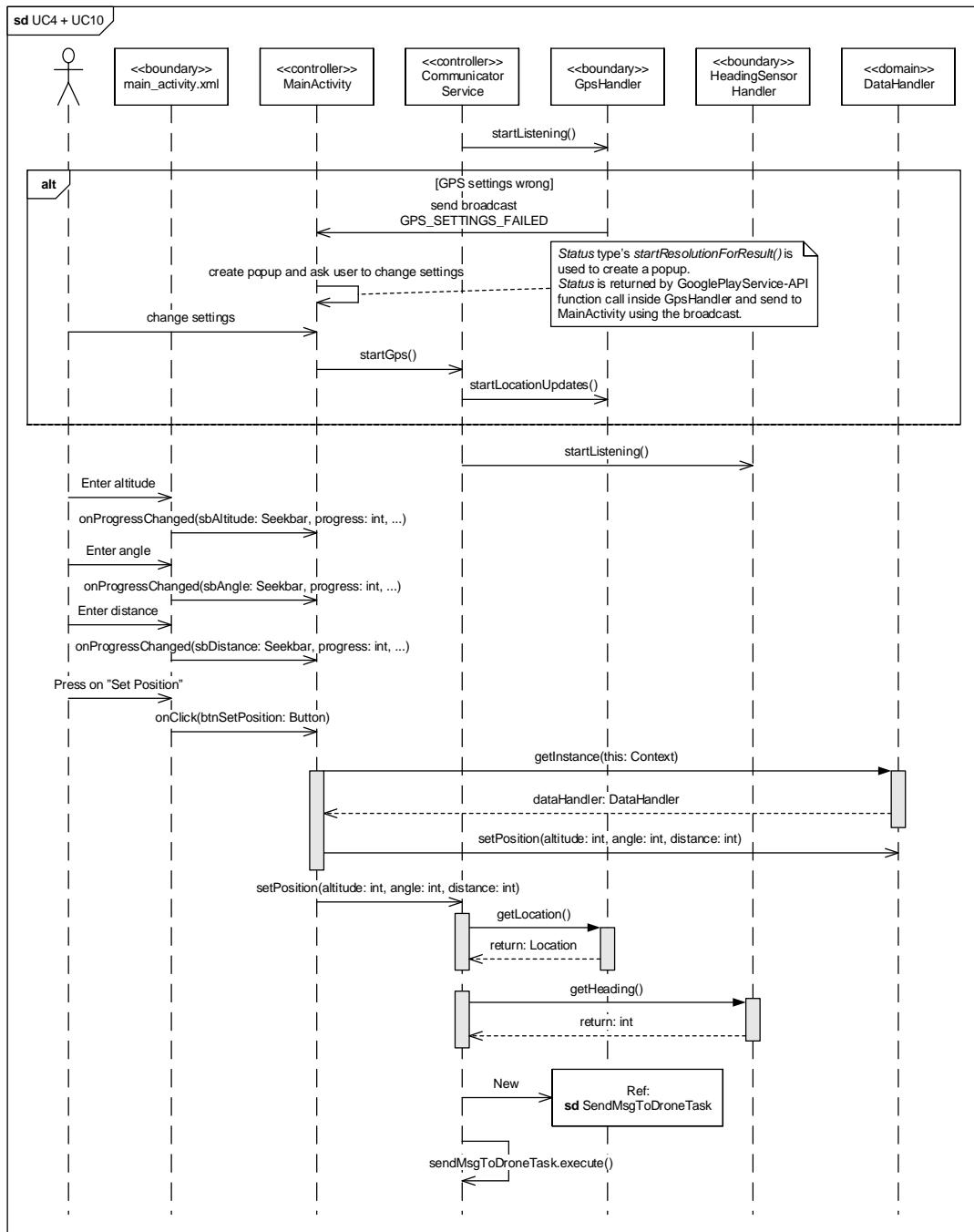


Figur 3.52: CPU Applikation - UC4 og UC10 konceptuelle klasser

Navn	Beskrivelsen
GpsHandler	Modtager GPS koordinater ved hjælp af Google Play services location API [16].
HeadingSensorHandler	Beregner mobiltelefonens retning i forhold til magnetisk nord ud fra accelerometer og magnetometer.

Tabel 3.8: Beskrivelser af klasserne i figur 3.52

Da der i Use Case'ene skal sendes en enkelt besked til dronen, bliver *SendMsgToDroneTask*'et benyttet. Grundet dette indgår *ComHandler*'en ikke i det efterfølgende sekvensdiagram, men i sekvensdiagrammet for *SendMsgToDroneTask* i figur 3.50.



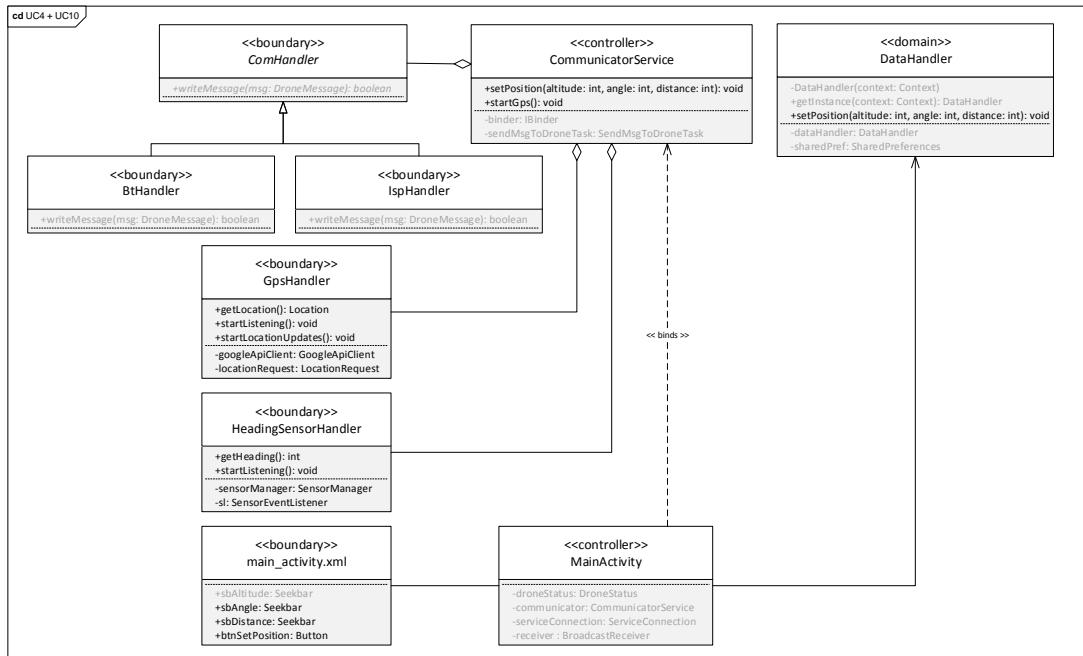
Figur 3.53: CPU Applikation - UC4 og UC10 sekvensdiagram

På diagrammet ses det at både *GpsHandler* og *HeadingSensorHandler* har en funktion med navn *startListening()*. Dette skyldes at sensorer på en Android mobiltelefon ikke tilgås direkte. Der implementeres et callback, som bliver kaldt, når der er nye sensor data tilgængeligt fra Android styresystemet. Denne funktionalitet beskrives mere detaljeret i afsnit 3.7.4.6.

Det ses at den valgte position bliver gemt i mobiltelefonens memory ved hjælp af *DataHandler*'en. Dette bliver gjort, da *FollowMeTask*, hvis funktionalitet er beskrevet

i diagram 3.57, henter disse data fra *DataHandler*'en. Derudover bliver brugerens sidst valgte position gemt og de tre *SeekBar*'s indstilles til dette når applikationen åbnes igen. Ud over den viste funktionalitet, blev der tilføjet en *ProgressBarHelper*, der wrapper Android-klassen *ProgressBar* [17], som åbner en loading screen, der bliver vist, hvis brugeren ønsker at indstille dronens position uden at GPS data er klar.

De funktioner, som Use Case 4 og 10 bidrager med til systemet, kan ses i det følgende klassediagram vist i figur 3.54.

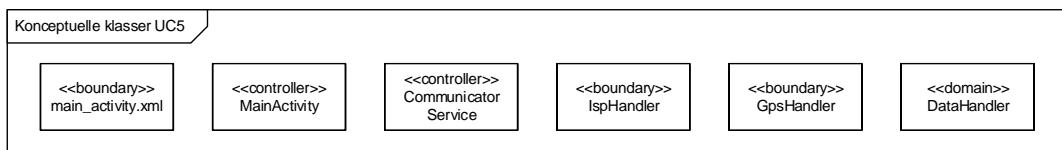


Figur 3.54: CPU Applikation - UC4 og UC10 Klassediagram

Figur 3.54 viser at der benyttes en *GoogleApiClient* [18] og *LocationRequest* [19], til at konfigurer præcisionen og hyppigheden af GPS-sigtalet. *HeadingSensorHandler* benytter en *SensorManager* [20], for at subscribe på accelerometer og magnetometer sensorene.

3.2.4.5 Use Case 5

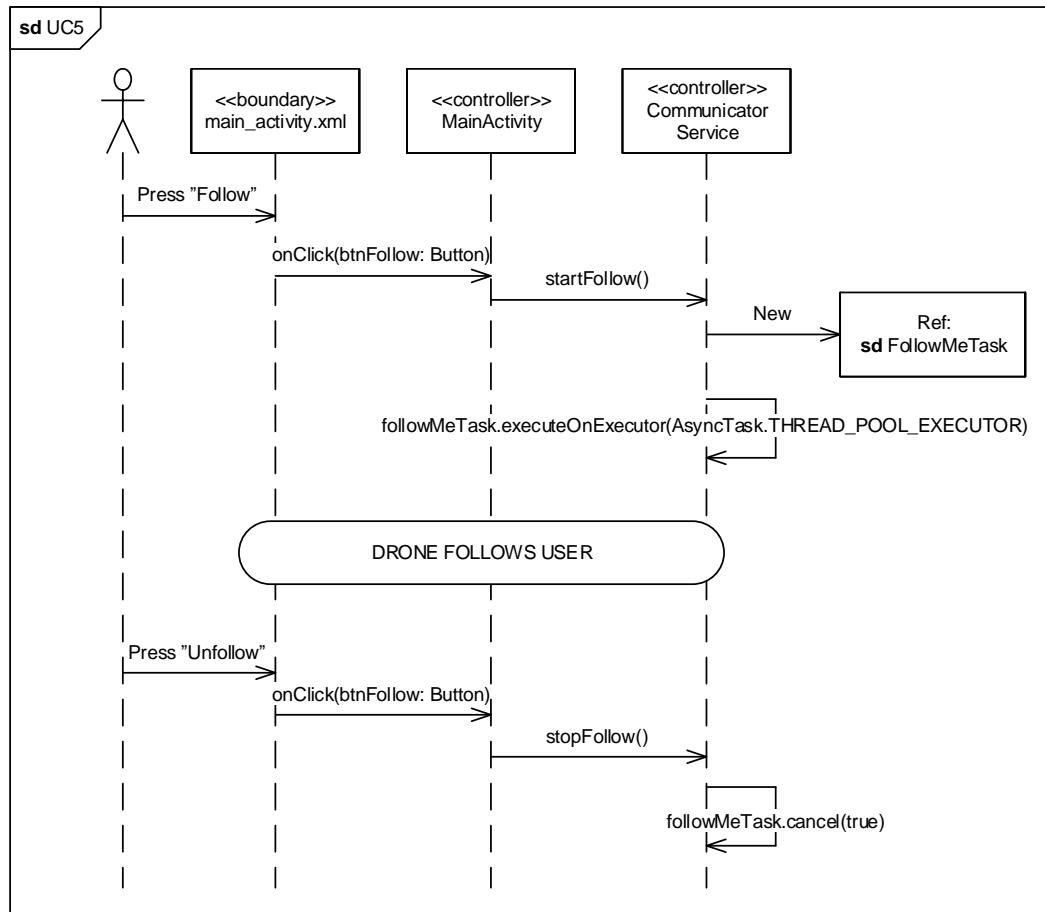
Ud fra Use Case'en og systemets domænemodel blev der fundet følgende konceptuelle klasser.



Figur 3.55: CPU Applikation - UC5 konceptuelle klasser

HeadingSensorHandler bliver ikke længere brugt, da brugerens retning bliver beregnet

ud fra GPS-koordinaterne. Underst  ende sekvensdiagram i figur 3.56 viser Use Case'ens funktionalitet. En del af funktionaliteten blev flyttet i en baggrunds task, da denne funktionalitet udf  res indtil brugeren ikke l  ngere   nsker det. Denne funktionalitet bliver vist i *FollowMeTask*'ets sekvensdiagram i figur 3.57.



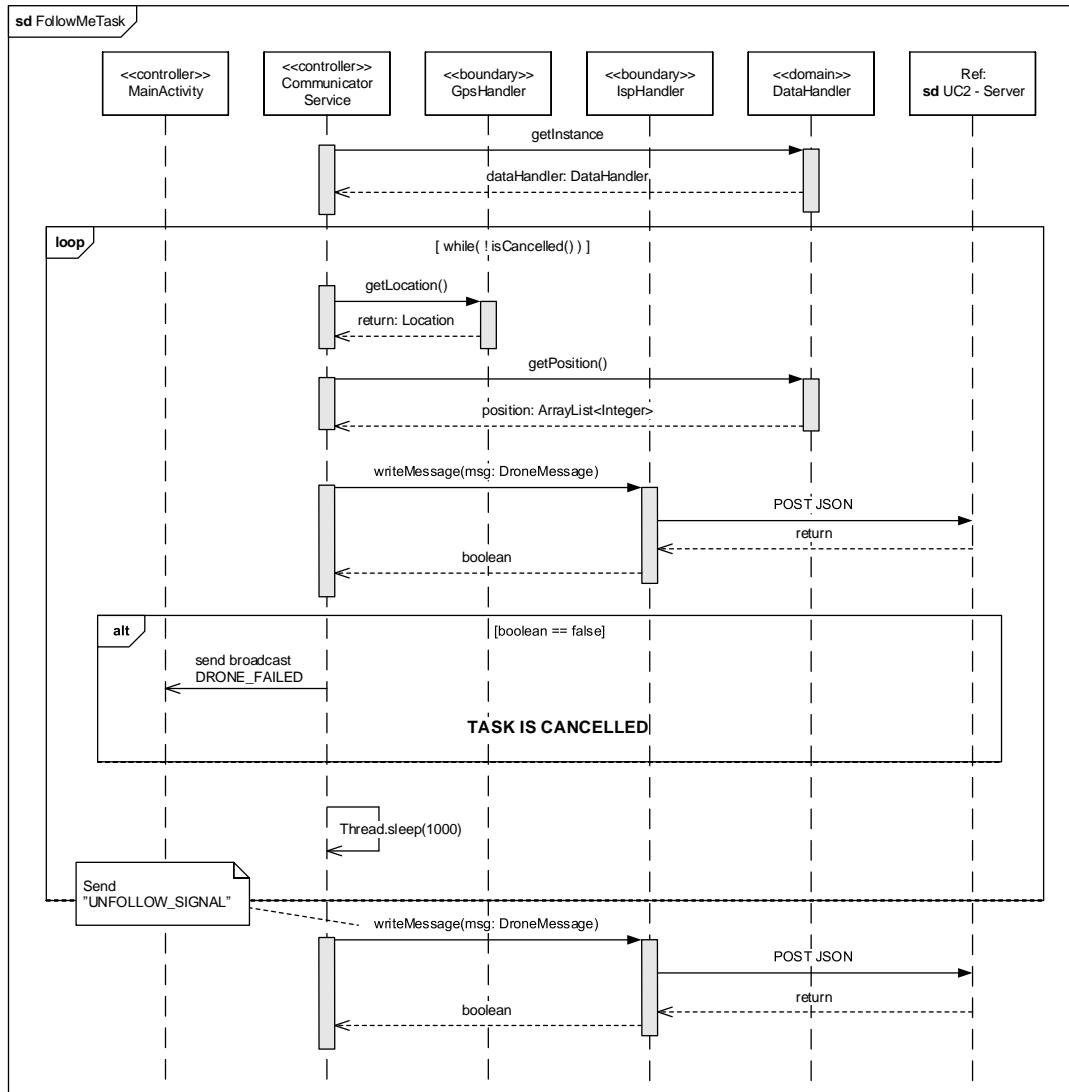
Figur 3.56: CPU Applikation - UC5 sekvensdiagram

P  r sekvensdiagrammet ses det at *FollowMeTask* bliver, lige som *GetDroneStatusTask*, eksekveret ved hj  lp af en *Executor*. Dette blev valgt, da denne skal k  re over l  ngere tid i applikations baggrund og skal have sin egen tr  d.

Tiden, som dronen folger brugeren i, varierer alt efter brugerens   nske og er derfor ikke defineret.

Det ses at *followMeTask.cancel(true)* bliver kaldt n  r tasken skal stoppes. Dette funktionskald f  rer ikke til at tasken afbrydes med det samme. Funktionen *isCancelled()*, som regelm  sigt skal kaldes i taskfunktionen, vil dog returnere true. Tasken skal dog manuelt afsluttes.

I sekvensdiagrammet i figur 3.57 ses *FollowMeTask*'ets funktionalitet.



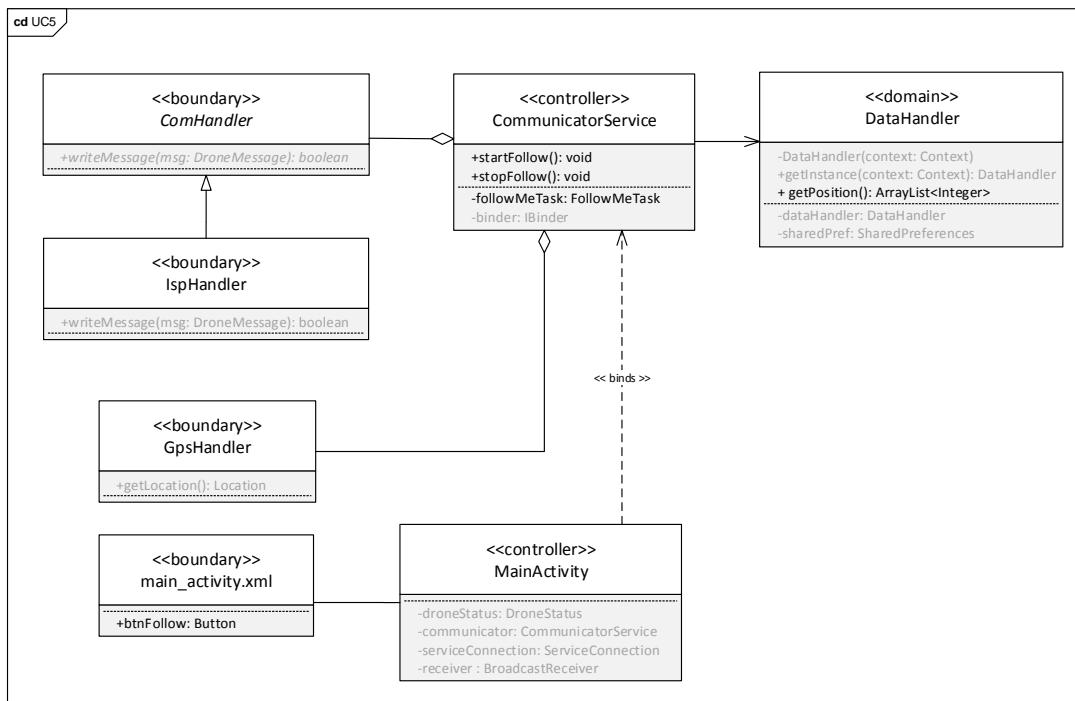
Figur 3.57: CPU Applikation - FollowMeTask sekvensdiagram

Det ses at efter tasken bliver afbrudt, bliver der stadig sendt en besked til dronen. Dette bliver gjort, da dronen er nødt til at vide, at det er efter hensigten, at der ikke bliver sendt flere opdateringer med brugerens lokation.

Det ses, at værdierne sat i Set Position Use Case'en løbende hentes fra `DataHandler`-klassen. Dette bliver gjort, da værdierne løbende kan opdateres af brugeren.

Det blev valgt at vise `IspHandler` i stedet for `ComHandler` på sekvensdiagrammet, for at understrege, at `FollowMeTask` kun kan aktiveres, når applikationen er forbundet til dronen via internettet.

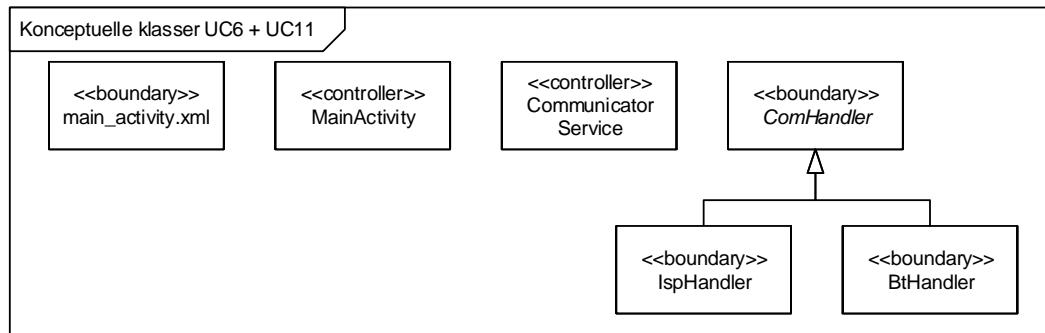
På figur 3.58 ses klassediagram med den nye funktionalitet, der er blevet tilføjet gennem Use Case 5.



Figur 3.58: CPU Applikation - UC5 Klassediagram

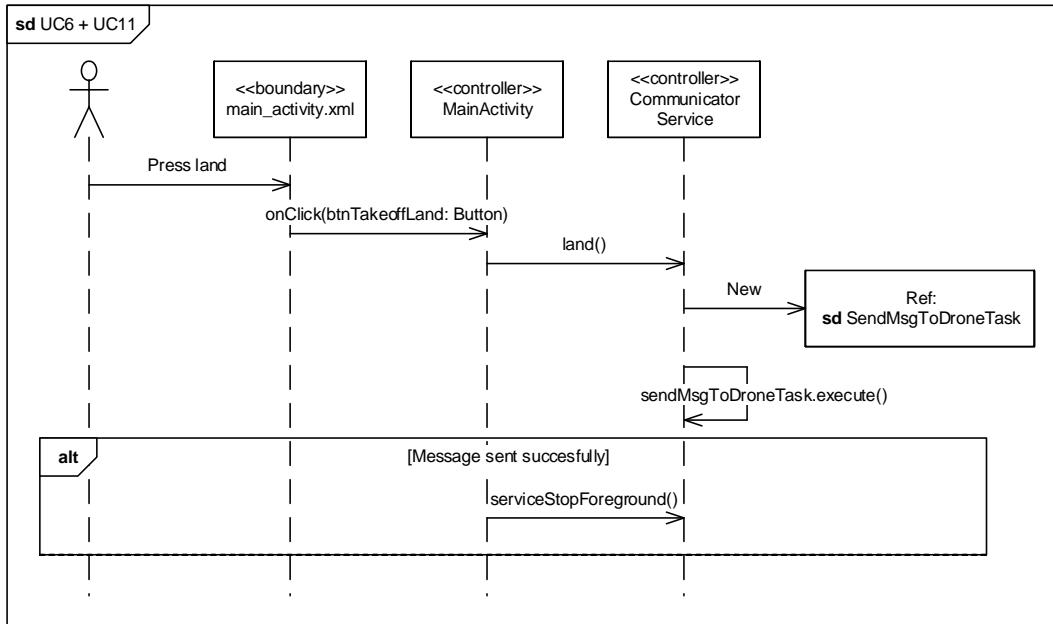
3.2.4.6 Use Case 6 og 11

Ud fra Use Case'ene og domænemodellen for systemet blev der fundet følgende konceptuelle klasser:



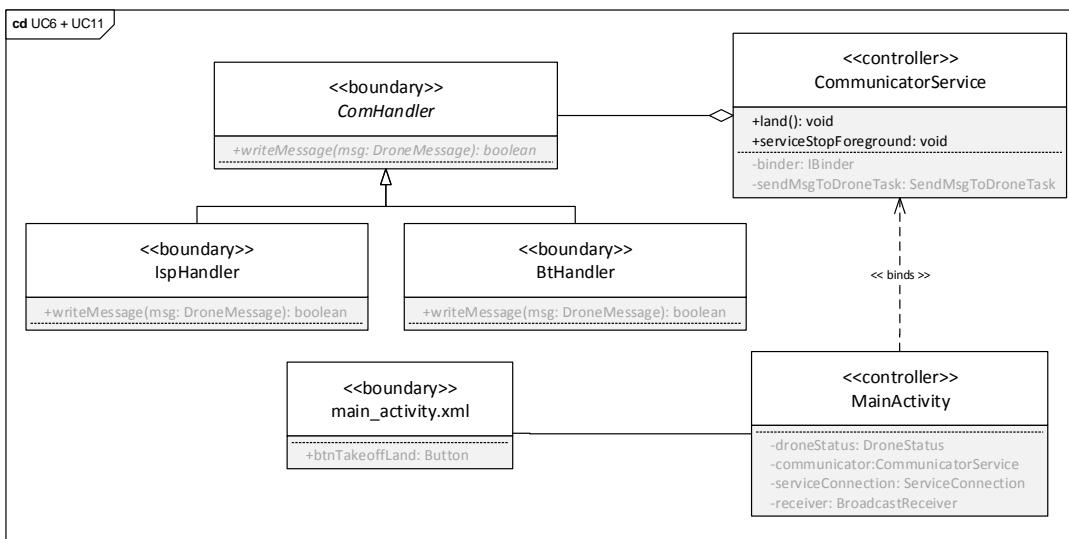
Figur 3.59: CPU Applikation - UC6 og UC11 konceptuelle klasser

Ud fra de konceptuelle klasser er følgende sekvensdiagram opstillet.



Figur 3.60: CPU Applikation - UC6 og UC11 sekvensdiagram

Der ses på sekvensdiagrammet, at *SendMsgToDroneTask* kunne genbruges fra Use Case 3 og 9. *CommunicatorService* har derimod fået to nye funktioner. *serviceStopForeground* sørger for at notifikationen i brugerens status bar forsvinder og applikationen dermed bliver sat tilbage til tilstanden før dronen er lettet. På figur 3.61 ses klassediagrammet for Use Case 6 og 11.

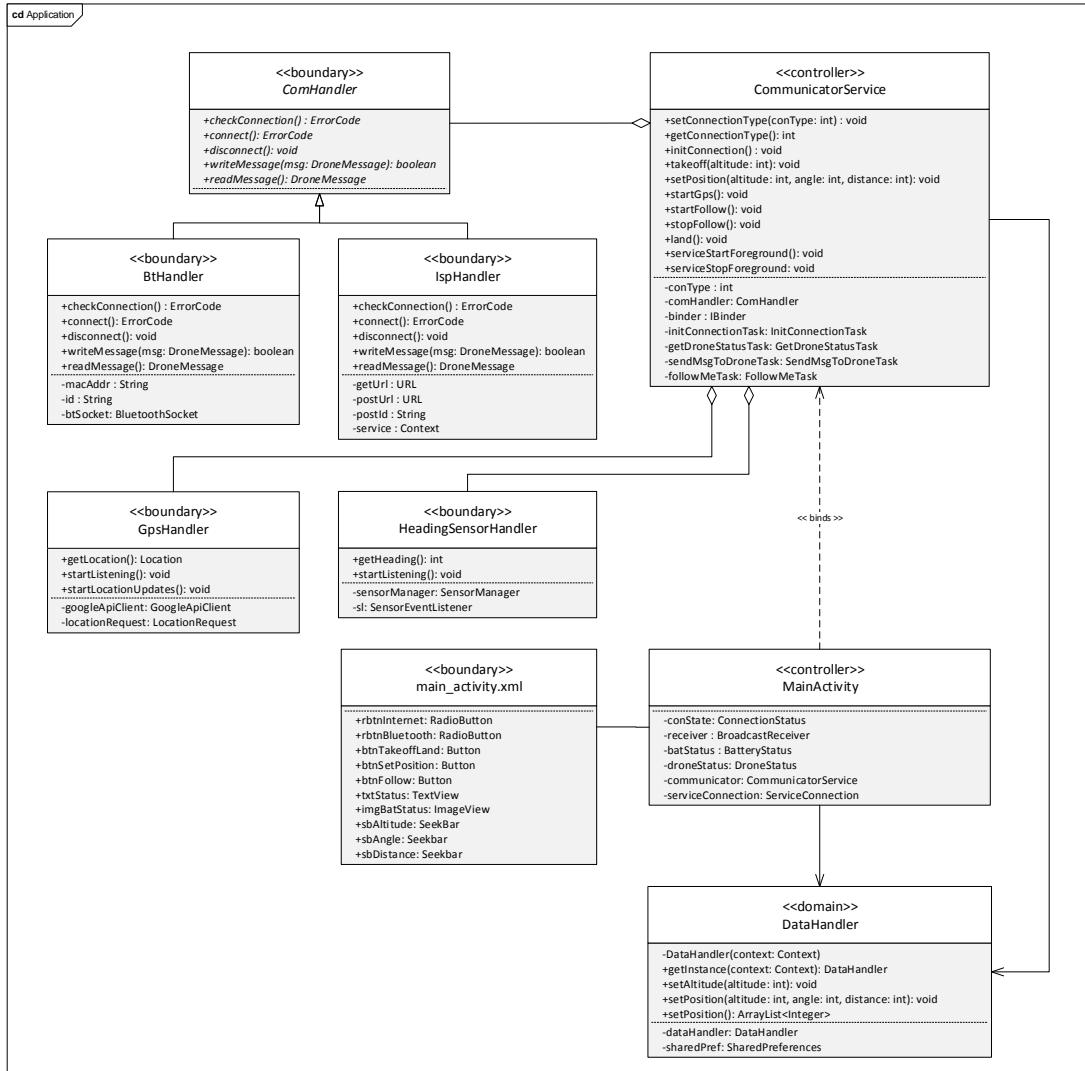


Figur 3.61: CPU Applikation - UC6 og UC11 Klassediagram

På klassediagrammet ses den nye funktionalitet, der blev tilføjet til systemet i Use Case 6 og 11.

3.2.4.7 Samlet klassediagram for Applikationen

Ud fra de statiske klassediagrammer for de enkelte Use Case's der er udarbejdet i Applikationens Logical View, er et samlet klassediagram for applikationen blevet opstillet. Dette ses på figur 3.62.



Figur 3.62: CPU Drone - Samlet klassediagram for Applikationen

3.3 Process View

I dette afsnit beskrives de forskellige processer eller tråde, som kører på de enkelte CPU'er og deres indbyrdes kommunikation og ansvarsområder.

3.3.1 Server

Serveren har primært kun en proces som håndterer de indkommende HTTP beskeder. Ansvaret for denne tråd er, at udfører de korrekte handlinger baseret på HTTP beskeden, denne har modtaget. Ingen tråde er implementeret på serveren fra projektets side.

3.3.2 Drone

På dronen benyttes en Raspberry Pi 3, som har en quad-core processor. Dette gør, det derfor oplagt at benytte et multitrådet system. I dette afsnit vil de forskellige tråde i dronen blive forklaret og inddelt efter hvilken klasse de tilhøre.

3.3.2.1 DataHandler

Dronen indeholder en *DataHandler* til at samle alt data på dronen når den er tændt. *DataHandler*'en samler alt data på dronen et sted således at alle klasser kan tilgå data'et og har mulighed for at opdatere data'en. Flere af dronens tråde benytter *DataHandler*'en, da disse læser informationer fra f.eks. FlightController'en og GPS satellitterne.

Da flere tråde har mulighed for at tilgå det fælles data, skal det beskyttes af mutex'es, således at der ikke kan opstå problemer med at to tråde tilgår samme data på samme tid. *DataHandler*'en indeholder to structs til at gemme hhv. informationer fra dronen selv og applikationen. Disse structs er beskyttet af en mutex hver. Dette gør, at når dataet skal sættes eller læses fra disse structs, skal den tilhørende mutex først låses.

Da dronens *DataHandler* på denne måde er tråd sikker, behøves de enkelte tråde ikke bekymre sig om dette problem. Det gør at alle tråde, som har en pointer til *DataHandler*'en blot kan benytte funktionerne velvidende at disse er sikret således at en korrekt dataoverførelse er garanteret.

3.3.2.2 DroneController

DroneController-klassen har primært tråde til at styre dronens bevægelse i x, y og z retningen. Dette betyder, at der er tråde, der styrer hvorvidt dronen skal flyve op/ned, frem/tilbage, eller dreje. Derudover har klassen to tråde, en som står for at sende en batteristatus-besked til applikationen hvert 5. sekund, efter at applikationen har forbundet til dronen, og en der står for at styre batteriets LED.

DroneController'en agere main-tråd, hvori denne står for at starte alle andre tråde. main-tråden vil starte *batteryMonitor*-tråden, som den første tråd. Denne tråd vil fortsætte indtil programmet termineres. *status*-tråden bliver startet lige så snart brugeren etablerer forbindelse til dronen. Er forbindelsen over bluetooth, vil *status*-tråden blive joinet lige så snart bluetooth-forbindelsen lukkes. Er forbindelsen over 3G, vil *status*-tråden fortsætte indtil programmet termineres. Denne funktionalitet er implementeret i *using3G()* og *usingBT()*.

takeOff-tråden, *land*-tråden, *setPosition*-tråden og *follow*-tråden, kan ikke kører på samme tid. Dette betyder, at hver gang main-tråden forsøger at starte en af disse tråde, skal den sørge for at checke om en af de andre tråde kører. Hvis dette er tilfældet skal den tråd joines inden den nye tråd kan startes. Dette gøres, da der ellers vil opstå situationer hvor trådene modarbejder hinanden. Eksempelvis hvis *land*-tråden og *takeOff*-tråden kører på samme tid.

using3G

Denne funktion ligger i main-tråden. Funktionen kaldes når det første valide Start-signal fra serveren hentes ned. Er denne funktion først startet vil den fortsætte indtil et Stop-signal er modtaget, forbindelsen til serveren har været nede i 30 sekunder eller at batterispændingen er under 9.5V. Inden main-tråden går ud af funktionen vil *land*-tråden blive kaldt og joined således at det sikres at dronen er landet.

usingBT

Denne funktion ligger i main-tråden. Funktionen kaldes hvis der ikke kan oprettes forbindelse til serveren. Funktionen vil derefter initialisere bluetooth og vente på en indgående forbindelse. Er denne funktion startet vil den fortsætte indtil batterispændingen når ned på 9.5V, et Stop-signal er modtaget eller at bluetooth forbindelsen termineres. Inden main-tråden går ud af funktionen vil *land*-tråden blive kaldt og joined således at det sikres at dronen er landet.

status

Input:

Reference til klassen selv i form af en "this" pointer.

Beskrivelsen:

Tråden skal hvert 5. sekund sende en besked til applikationen enten via 3G eller Bluetooth. Denne besked skal bestå af en JSON pakke som indeholder relevante data. De relevante data er hentet fra *DataHandler*'en. Beskeden bliver sendt vha. dronens *Communicator*.

batteryMonitor

Input:

Reference til klassen selv i form af en "this" pointer.

Beskrivelsen:

Denne tråd står for at opdatere dronens RGB diode på interfacet til at afspejle dronens batterispænding. Tråden henter batterispændingen fra *DataHandler*'en og baseret på værdien, sætter den farven på RGB dioden.

takeOff

Input:

Reference til klassen selv i form af en "this" pointer.

Beskrivelsen:

Opgaven for denne tråd er at styre reguleringen, når dronen skal lette. Dronen vil løbende udlæse højden fra sonarsensoren indtil højden overstiger 5 meter. Derefter vil dronen benytte barometeret til at måle højden. Baseret på dronens nuværende højde og brugerens ønskede højde for dronen, styrer dronen motorernes kraft. Dette betyder, at motorerne får

højere kraft jo større forskellen er imellem dronens højde og brugerens ønskede højde for dronen. Motorerne vil på den måde nå en vis kraft, når den ønskede højde er nået. Dronen vil blive ved med at regulere motorerne, således at højden blive holdt mere eller mindre konstant. Når *takeOff*-tråden startes vil den gemme dronens GPS-koordinat. Når dronen når en højde på over 2 meter, vil dronen begynde at holde sig på det gemte GPS-koordinat. Dette gør, at dronen ikke vil drive afsted under letningen.

land

Input:

Reference til klassen selv i form af en "this" pointer.

Beskrivelsen:

land-tråden har den modsatte opgave i forhold til *takeOff* tråden. Denne skal bringe dronen sikkert ned på jorden igen. Tråden starter med at nedsætte motorkraften med 5%. Alt efter hvilken højde dronen befinder sig i benyttes hhv. enten sonarsensoren eller barometeret. Tråden udregner hvor hurtigt dronen falder baseret på målte højde værdier og justerer hhv. motorkraften op eller ned for derved at opretholde et konstant kontrolleret fald. Når dronen når ned i en højde af 30 centimeter nedjusteres motorkraften betydeligt.

setPosition

Input:

Reference til klassen selv i form af en "this" pointer.

Beskrivelsen:

Denne tråd vil bringe dronen hen i brugerens ønskede position af dronen. Tråden starter med at hente alle nødvendige data fra *DataHandler*'en. Disse indeholder GPS-koordinater på brugeren, vinklen og afstanden dronen skal have til brugeren, den ønskede højde af dronen og brugerens retning. Derudover indeholder dataene også dronens GPS-koordinat og retning. Tråden udregner på baggrund af disse data et GPS-koordinat som er den position, brugeren ønsker dronen i. Tråden vil nu styre dronen til det ønskede punkt. Det er muligt for dronen at kalde denne funktionen igen. Dette vil blot resultere i at den nuværende *setPosition*-tråd vil blive joined og en ny vil starte op.

follow

Input:

Reference til klassen selv i form af en "this" pointer.

Beskrivelsen:

Denne tråd vil få dronen til at følge efter brugeren. Tråden vil konstant udlæse data fra brugeren og beregne brugerens ønskede droneposition på baggrund af disse. Tråden vil derfor hele tiden forsøge at få dronen flyttet hen til den nyberegnede droneposition. Dette vil resultere i at dronen følger efter brugeren.

3.3.2.3 GPSHandler

GPSHandler-klassen benytter en tråd til løbende at læse GPS-beskederne fra dronens GPS-modul. Dette ønskes opdateret kontinuerligt og er derfor placeret i en tråd.

readGPS

Input:

Reference til klassen selv i form af en "this" pointer og en pointer til *DroneController*-klassens *DataHandler*.

Beskrivelsen:

Denne tråd læser på den serielle forbindelse til GPS-modulet og læser hhv. en GPGGA-besked og en GPRMC-besked. Når beskederne er læst fra den serielle port gemmes de i dronens *DataHandler* via en set funktion. Når tråden er startet lukkes denne ikke før dronen slukkes.

3.3.2.4 Interface

Interface-klassen benytter tråde til at blinke med LED'erne på dronens interface.

blinking

Input:

Denne tråd modtager et pin-nummer som integer og en pointer til en integer.

Beskrivelsen:

Denne tråd kan blive oprettet op til 3 gange på samme tid, da dronen har 3 LED'er på sit interface, som skal kunne blinke. Pin-nummeret, som er givet med som input parameter, bestemmer hvilken pin på Raspberry Pi'en der tændes og slukkes hvert halve sekund. Tråden kører i et loop så længe integeren, som pointeren peger på, er forskellig fra 0. Når denne ændre værdi til 0 brydes while-loopet og tråden kan lukkes sikkert. Disse tråde benytter en mutex, da alle bruger den samme funktion til at ændre stadiet på en given pin udgang (High/Low). Det vides ikke om denne funktion er trådsikker og derfor er det valgt, at beskytte denne funktion med en mutex. Det vil sige, at kun én pin udgang kan ændres ad gangen.

3.3.2.5 SerialArduinoHandler

SerialArduinoHandler-klassen har en tråd til kontinuerligt at læse data fra dronens FlightController. Herfra læses højde fra barometeret, spændingen på batteriet og retningen fra magnetometeret og accelerometeret.

readFlightData

Input:

Reference til klassen selv i form af en "this" pointer og en pointer til *DroneController*-klassens *DataHandler*.

Beskrivelsen:

Tråden står for at hente data fra sensorerne på FlightController'en. Dette indebærer følgende sensorer; barometer, magnetometer, accelerometer, og batterimåleren. Baseret på magnetometeret og accelerometeret kan denne tråd udregne en retning på dronen i forhold til magnetisk nord.

3.3.2.6 StepMotorHandler

StepMotorHandler-klassen styrer dronens stepmotor og dermed kameraet. Denne klasse bruger en tråd til at indstille kameraets position.

turnCamera

Input:

Reference til klassen selv i form af en "this" pointer.

Beskrivelsen:

Denne tråd benytter to hjælpefunktioner til henholdsvis at dreje kameraet imod og med urets retning. Når main-tråden kalder funktionen til at styre kamerat ind i en given vinkel, omregnes denne vinkel til et antal steps. Antallet af steps benyttes af denne tråd, til at vide hvor langt kameraet skal drejes.

3.3.3 Applikation

Som udgangspunkt kører en Android applikation på en tråd, som kaldes main-tråden. Selvom der i dette system oprettes en service til at håndterer opgaver der tager længere tid, vil disse som udgangspunkt kører i main tråden. Dette fører til, at brugerinterfacet ikke kan benyttes imens der f.eks. sendes data over 3G. En overbelastet main-tråd, fører til en uresponsivt brugergrænseflade. Dette detekteres af Android styresystemet. Brugeren vil blive spurgt om, vedkommende vil lukke applikationen og eventuel deinstallere den.

Derfor er det vigtigt at køre alle opgaver, der tager længere tid, i en anden tråd. I dette system blev det valgt at bruge klassen *AsyncTask* [10]. En *AsyncTask* har en række callback funktioner, der kan overskrives. Den vigtigste er *doInBackground*. Her skrives koden, der ønskes eksekveret i baggrunden. Når opgaven bliver afsluttet kaldes callback funktionen *onPostExecute*. Der er mulighed for at afbryde en baggrundstask. Dette afslutter dog ikke tasken med det samme, men skal checkes løbende med funktionen *isCancelled*. Når opgaven afsluttes efter den blev afbrudt, bliver funktionen *onCancelled* kaldt i stedet for *onPostExecute*.

Tasks oprettes som klasser i *ConnectionService*. Disse klasser skal arve fra klassen *AsyncTask*. En task kan defineres med forskellige parametre og returnværdier, der bruges i callback funktionerne. Det er således muligt, at sende data med til en task, som denne skal bruge.

Som udgangspunkt kører alle *AsyncTask* der bliver startet i samme tråd. For at køre de forskellige tasks parallelt i forskellige tråde, skal de startes med en executor:

```
backgroundTask.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, msg);
```

Her ses, at tasken *backgroundTask* startes med en executor *THREAD_POOL_EXECUTOR* og at der bliver sendt parametren *msg* med til den asynkrone task. Grundet de asynkrone tasks, returnerer servicen resultater til *MainActivity* ved hjælp af en *LocalBroadcastReceiver* [13].

3.3.3.1 DataHandler

Da *DataHandler*-klassen tilgås fra forskellige tråde, skal denne klasse gøres trådsikker. For at gøre dette blev klassen implementeret som Singleton klassen. Klassen har altså en privat constructor og skal tilgås ved hjælp af en *getInstance()* funktion. Dermed bliver der kun oprettet en instans af klassen. Derudover blev der ved hjælp af Java's *synchronize* [21] funktionalitet sikret, at der ikke kan kaldes to af klassens funktioner på samme tid af forskellige tråde.

3.3.3.2 CommunicatorService

Klassens ansvar er alt kommunikation med dronen og serveren. Derfor bliver alle asynkrone tasks erklæret i denne klasse.

InitConnectionTask

Input:

Ingen.

Output:

En enum af typen *ErrorCode* der bliver sendt til *MainActivity* i *onPostExecute()*.

Beskrivelsen:

Denne tråd er ansvarlig for, at forbinde til den valgte kommunikationstype. Som det første testes der, om der er forbindelse til henholdsvis internettet eller Bluetooth.

Hvis Bluetooth er slukket, bedes brugeren om at tænde for det. Dette bliver gjort ved at sende en broadcast til aktivitien, hvorefter der bruges en *ConditionVariable* [22]. Den kaldes *block()* på dette objekt, som fører til at tråden venter, indtil der bliver kaldt *open()* fra en anden tråd på objektet. Dette bliver gjort i main-tråden, når brugeren har svaret på, om Bluetooth må aktiveres.

Dernæst oprettes selve forbindelsen. Når der skiftes forbindelsestype, imens tråden bliver eksekveret, ventes der på, at den er blevet afsluttet, før den startes igen.

Når det lykkes, at oprette forbindelsen bliver *GetDroneStatusTask* startet i *onPostExecute*.

SendMsgToDroneTask

Input:

Objekt af typen *DroneMessage*.

Output:

En enum af typen *ErrorCode* der bliver sendt til *MainActivity* i *onPostExecute()*.

Beskrivelsen:

Denne tråd sender objektet der sendes med som parameter over 3G eller Bluetooth til henholdsvis serveren eller dronen. Tråden bruges i alle Use Case's hvor der skal sendes en besked en enkelt gang, som f.eks. TakeOff-beskeden.

GetDroneStatusTask

Input:

Ingen.

Output:

En enum af typen *ErrorCode* der bliver sendt til *MainActivity* i *onPostExecute()*.

Beskrivelsen:

Denne tråd læser via 3G eller Bluetooth fra henholdsvis serveren eller dronen. Der modtages løbende status beskeder fra dronen, som indeholder blandt andet informationer omkring dronens batteri status. Batteri status'en sendes videre til *MainActivity*, som en broadcast.

Tråden kører altid, når der er blevet oprettet forbindelse til dronen. Den bliver læst en ny besked hvert 5. sekund. Hvis der ikke bliver modtaget en ny besked i 30 sekunder, afsluttes tasken og der sendes en besked til *MainActivity*, om at applikationen har mistet forbindelse til dronen.

FollowMeTask

Input:

Ingen.

Output:

En enum af typen *ErrorCode* der bliver sendt til *MainActivity* i *onPostExecute()*.

Beskrivelsen:

Tråden sender løbende opdateringer til dronen. Ud over Follow-signal kommandoen, indlæser tråden den mest aktuelle lokation fra *GpsHandler*-klassen. afstand, højde og vinkel som dronen skal have i forhold til brugeren indlæses fra *DataHandler*'en. Denne information sendes til dronen. Der bliver sendt en ny kommando hvert sekund, indtil brugeren ikke længere ønsker at dronen følger ham.

3.4 Data View

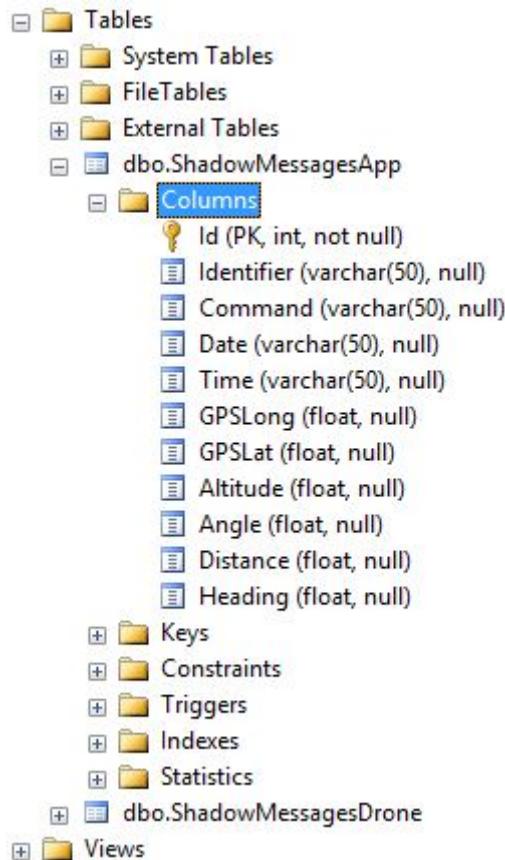
Dette afsnit omhandler data, som bliver gemt og håndteret i systemet. Serverens databasens opsætning og hvordan denne tilgås og modificeres er et eksempel på dette.

3.4.1 Server

3.4.1.1 Serverens Database

Databasen i systemet indeholder kun data, som ønskes persistent. Denne skal gemme alle tidlige beskeder i sine tabeller og skal kunne tilgå disse på senere tidspunkter.

Databasen indeholder to tabeller. En tabel til at holde alle beskeder sendt fra applikationer og en til alle beskeder sendt fra droner. Tabellerne er ens, hvad angår kolonnerne og datatyperne, da de skal indeholde samme type af beskeder jf. tabel 3.10. I figur 3.63 vises den ene af tabellerne, som den ses i programmet Microsoft SQL Server Management Studio [23]. Id er den primary key i databasen og bliver automatisk incrementeret for hver nye besked.



Figur 3.63: Kolonnerne i databasens tabel

De forskellige kolonner i tabellerne består af forskellige typer. Disse typer er kort forklaret herunder:

- varchar(50) er en variable string på maksimalt 50 karakterer. En header på 2 bytes

angiver længden af den aktuelle værdi.

- int er et 32bit heltal.
- float er et 32bit kommatal.

For at kunne benytte disse tabeller bruges flere forskellige Stored Procedures. En procedure er en måde, at skrive SQL scripts på. Dette giver muligheden for at lave en form for funktioner til databasen. I projektet er der oprettet 5 procedurer, 2 til applikations tabel og 3 til dronens tabel.

- En procedure til at returnere det nyeste element i tabellen.
- En procedure til at returnere det nyeste element i tabellen med den specifikke kommando. - (Kun dronens tabel)
- En procedure til at tilføje et nyt element i tabellen.

Disse procedure kan benyttes direkte fra serverens Web Api. På denne måde kan Web Api'et udnytte databasen.

3.4.1.2 Serverens Web Api

Serverens Web Api holder ikke nogle persistente data men kun midlertidige instanser af beskederne, som skal benyttes i de forskellige HTTP requests. Serverens Web Api indeholder en model, som igen indeholder variabler tilsvarende de, databasen indeholder.

```
namespace ShadowBAC_Sever.Models
{
    /// <summary>
    /// Message based on the tables in the database being used.
    /// </summary>
    public class ShadowBACMessage
    {
        public int Id { get; set; }
        public string Identifier { get; set; }
        public string Command { get; set; }
        public string Date { get; set; }
        public string Time { get; set; }
        public float GPSLong { get; set; }
        public float GPSLat { get; set; }
        public float Altitude { get; set; }
        public float Angle { get; set; }
        public float Distance { get; set; }
        public float Heading { get; set; }
    }
}
```

Listing 3.1: ShadowBACMessage

Når data hentes eller skal bruges i Web Api'et fra databasen håndteres det i instanser af denne klasse, som ses i Listing 3.1.

3.4.2 Drone

På dronen er der ingen data, som gemmes i koden ved nedlukning af dronen. Dette betyder, at dronen ikke indeholder nogen persistent data, dog gemmes en debug log på Raspberry Pi'ens styresystem. Denne indeholder debug log's fra koden. Den bruges primært under udviklingen, da det er svært at debugge koden på Raspberry Pi'en. Loggen genereres gennem Easylogging++, som forklares i yderligere detaljer i afsnittet 3.7.3.4.

Selvom dronen ingen persistent data indeholder, gemmer dronen en del data, når den er tændt. Dette gøres gennem dronens *DataHandler*. Denne klasse er implementeret primært for at have alt data samlet et sted. Da Raspberry Pi'en benytter et multitrådet system, er *DataHandler*'en implementeret således, at den er trådsikker. Dette sikrer dronen mod problemer med tråde, der tilgår hukommelse på samme tid.

3.4.3 Applikation

I Applikationen er der en række vigtige variabler, der skal gemmes, selvom applikationen lukkes ned. Dette får især betydning, når applikationen lukkes ned, og dronen er i luften. Det ønskes her, at applikationen husker sin tilstand, når den åbnes op igen. Da datamængden er meget lille, blev det valgt, at gemme dem i en fil i intern memory ved hjælp af *SharedPreference* API'et [12]. Denne fil bliver oprettet privat, så den ikke kan tilgås fra andre applikationer.

De følgende variabler bliver gemt i den lokale memory:

Drone ID	Dronens unikke ID
Drone Status	Den tilstand dronen er i (på jorden, i luften eller følger)
Connection Status	Forbindelsens tilstand (ikke forbundet, forbundet, mislykket...)
Altitude	Den sidste højde, brugeren har indstillet
Angle	Den sidste vinkel, brugeren har indstillet
Distance	Den sidste afstand, brugeren har indstillet
Position Set	Et flag, der bliver sat, når brugeren har valgt en position

Tabel 3.9: Data gemt på smartphonen

For at gøre det nemt i de andre klasser at gemme data, blev der valgt at lave en wrapper omkring *SharedPreference* som hedder *DataHandler*. Da der gemmes og tilgås data fra forskellige tråde, blev der valgt at lave klassen trådsikker.

3.5 Deployment View

I dette afsnit beskrives de forskellige protokoller, som forbinder systemets blokke. Kommunikationen mellem applikationen og dronen heriblandt deres samspil med serveren.

3.5.1 Besked struktur

Strukturen af beskederne er opbygget således, at kun det mest nødvendige bliver sendt mellem applikationen og dronen. På baggrund af dette, er følgende informationer valgt til besked strukturen.

Beskeds struktur
Identifier
Command
Date
Time
GPSLong
GPSLat
Altitude
Angle
Distance
Heading

Tabel 3.10: Systemets besked struktur

Identifier, er hvem der har sendt beskeden, dette kunne f.eks. være app33 eller drone57. *Command*, er den kommando, som beskeden indeholder. Denne kan være en af flere muligheder se tabel 3.11. *Date*, er dato'en beskeden er modtaget på serveren i formatet "dd:mm:yy". *Time*, er tiden beskeden er modtaget på serveren i formatet "hh:mm:ss". *GPSLong* og *GPSLat* er hhv. longitude og latitude fra et GPS koordinat. *Altitude*, *Angle*, *Distance* er højden, vinklen og afstanden som dronen skal holde i forhold til brugeren. *Heading*, er retningen som beskedens afsender havde på det givne tidspunkt. Inden beskederne sendes pakkes disse i et JSON format.

Den maksimale pakkestørrelse i bytes kan udregnes på følgende måde baseret på typerne nævnt i afsnittet 3.4.1.1:

Varchar(50) svarer til maksimalt 50 bytes, da en karakter eller char er 8bit, dvs. at en varchar(50) kan maksimalt indeholde 50 karakterer med 2 bytes overhead. Disse er der 4 af altså 208bytes. Herefter indeholder pakken 6 float's, som er 32bit eller 4 bytes. Derfor bliver den maksimale pakke størrelse 232bytes.

Inden beskederne sendes via 3G eller Bluetooth pakkes disse ind i et JSON format. Dette er en åben standard som benyttes, når dataet indeholder attributter med tilhørende værdier. Når disse beskeder pakkes på denne måde, er det nemt at lave et interface på begge sider af systemet. En typisk JSON pakke vil se ud som den er vist i listing 3.2.

```
{"Identifier ":"App1337","Command":"START_SIGNAL","Date":"08-11-16","Time":"12:48:49",
"GPSLong":10.1914854,"GPSLat":56.1714973,"Altitude":15,"Angle":90,"Distance":0.0,"Heading":0.0}
```

Listing 3.2: Besked pakke i JSON format.

3.5.1.1 Kommandoer

Feltet *Command* i besked strukturen kan indeholde følgende typer af kommandoer:

Kommando	Kommandobeskrivelse
START_SIGNAL	Kommando der sendes fra applikationen til dronen, for at validere forbindelsen.
START_SIGNAL_ACK	En ACK på START_SIGNAL-kommandoen fra applikationen når applikationen modtager denne, er forbindelsen valideret.
TAKEOFF_SIGNAL	Kommando der sendes fra applikationen til dronen, når brugeren ønsker dronen skal lette.
LAND_SIGNAL	Kommando der sendes fra applikationen til dronen, når brugeren ønsker dronen skal lande.
FOLLOW_SIGNAL	Kommando der sendes fra applikationen til dronen, når brugeren ønsker dronen skal følge brugeren. Kommandoen sendes kontinuerligt så længe Follow-funktionaliteten er aktiv.
UNFOLLOW_SIGNAL	Kommando der sendes fra applikationen til dronen, når dronen skal stoppe med at følge applikationen.
SETPOS_SIGNAL	Kommando der sendes fra applikationen til dronen, når brugeren ønsker dronen i en specifik position.
LOWBAT_SIGNAL	Kommando der sendes fra dronen til applikationen, såfremt dronens batterispænding er i det lave område.
MEDBAT_SIGNAL	Kommando der sendes fra dronen til applikationen, såfremt dronens batterispænding er i det midterste område.
HIGHBAT_SIGNAL	Kommando der sendes fra dronen til applikationen, såfremt dronens batterispænding er i det høje område.
STOP_SIGNAL	Kommando der sendes fra applikationen til dronen, når brugeren ønsker dronen skal slukkes.

Tabel 3.11: Kommandoer

3.5.2 Kommunikation mellem systemets blokke

Da serveren er bindeleddet mellem dronen og applikationen, er kommunikationen herimellem vigtig. Både dronen og applikationen har tilgang til 3G og kan på denne måde nå serveren alle steder med dækning. Dronen og applikationen kan gennem Bluetooth oprette direkte forbindelsen til hinanden. Begge har ligeledes adgang til GPS netværket gennem hhv. smartphone's GPS og dronens GPS modul.

3.5.2.1 TCP

TCP er en transportlagsprotokol, som er en af de vigtigste protokoller i nutidens internet. Denne protokol gør det muligt for enheder på internettet, at kommunikere med hinanden og sende data til hinanden. TCP arbejder sammen med IP, som definere hvordan pakker med data skal sendes til hinanden. Dette betegnes ofte som TCP/IP.

TCP er en forbindelses-orienteret protokol. Det betyder, at TCP, gennem et handshake, kan oprette en forbindelse mellem de to enheder, før derefter at opretholde denne til kommunikationen er færdig. Transportlagsprotokollen bestemmer også hvordan pakker skal

fragmenteres, og leverer disse til netværkslaget. TCP håndterer tab af pakker og ødelagte pakker gennem retransmission. Hvis en pakke aldrig kommer frem, vil senderen aldrig modtage en acknowledgement på denne pakke og derfor gensemdes pakken. Skulle pakker være blevet ødelagt under transporten opdager TCP dette gennem checksummen og kan derved vente på en retransmission.

TCP sikrer pålitelig kommunikation. Dette betyder, at alle pakker skal nå frem, dog ikke nødvendigvis i den korrekte rækkefølge. TCP kan håndtere, at pakkerne kommer i forskellig rækkefølge og stadig samle disse til den oprindelige datapakke [24].

3.5.2.2 HTTP Protokol

HTTP er en applikations protokol til udveksling af data mellem en server og en klient. HTTP bruges hver gang en hjemmeside hentes ned på en browser. Dette er klienten, som sender et HTTP-request til en given server og modtager et svar. Der eksisterer forskellige typer af HTTP-beskeder. I dette projekt er GET og POST dog de eneste implementeret. HTTP benytter en header til de mest nødvendige informationer og en body til selve dataet. I et GET-request skal f.eks. den ønskede URL specificeres og hvilken udgave af HTTP, der benyttes.

En vigtig ting omkring HTTP-protokollen er, at denne bygger på konceptet request-response. Dette betyder, at det altid er klienten, som skal tage initiativet og aldrig serveren. Hvis klienten mangler informationer, må denne bede om disse hos serveren og ligeledes sende data til serveren, når dette er nødvendigt [25].

HTTP-protokollen passer godt ind i projektet, som bindeleddet mellem dronen og applikationen. Dette giver begge muligheden for, at skrive til serveren med en ny besked. Denne besked kan således requestes fra modparten og derved hentes fra serveren. Ulempen ved dette system er, at dronen og applikationen selv skal requeste informationerne og ikke ved hvornår disse er tilgængelige. Derved skal en del af CPU kraften gå med at hente de måske rigtige data fra serveren. GET bruges altså til at hente en eller flere beskeder fra serveren, mens POST bruges til at ligge beskeder op på serveren.

3.5.2.3 Bluetooth

Bluetooth bruges i alle former for applikationer idag, alt fra høretelefoner, tastaturer, mus og andet. Bluetooth fungerer ved at en master kan have op til 7 slaver tilknyttet i et piconet. Piconets kan igen hænge sammen, da en slave kan være master i det næste piconet. Bluetooth opererer ligesom wifi på det frie 2.4GHz bånd og er derved også utsat for en masse støj. For at være mere resistent overfor støj benytter Bluetooth frekvens hopping, hvor denne hopper 1600 gange i sekundet mellem forskellige frekvenser. Hvis støj skulle opstå vil Bluetooth kun miste få pakker, da denne er hurtigt videre til næste frekvens.

Bluetooth 4.0 kan sende med en hastighed på op til 1Mbps. I Bluetooth 4.0 findes også BLE. Dette er lavet for at forblive på IoT markedet. Det skal fungere ved lavere strømme (<15uA) men med samme eller højere rækkevidde (>100m). [26]

Bluetooth fungerer som systemets failsafe, såfremt 3G-dækningen skulle være ikke eksisterende. Da projektet er lavet til udendørs brug forventes der en LoS til brugeren.

Dette gør, at der i projektet forventes en rækkevidde på 50m, hvilket vil være tilstrækkeligt.

3.5.2.4 GPS

GPS er et netværk oprindeligt af 24 satellitter i omløb omkring jorden. Oprindeligt blev systemet opsat af det amerikanske Department of Defense til brug i militærer aktiviter. I 1980's blev systemet gjort tilgængeligt for civile. GPS virker overalt i verden døgnet rundt. Idag fungere systemet gennem 31 satellitter i en højde på 20.200km over jordens overflade.[27]

En GPS satellit når omkring jorden cirka 1 gange i døgnet og sender løbende et signal ned mod jorden. Når en GPS modtager, modtager signaler fra 3 eller flere satellitter kan denne udregne sin position baseret på signalerne. Ud fra dette kan position altså bestemmes med en given usikkerhed. Jo flere satellitter, jo bedre bliver præcisionen af positionen. [28]

I projektet benyttes GPS til at bestemme positionen af både dronen og brugerne. Når dronen skal følge brugerne eller holde en given position, er det netop her GPS benyttes. GPS moduler kommer i mange forskellige størrelse og prisklasser. Til dette projekt er den lille størrelse af GPS modulet en fordel på baggrund af vægten. I projektet har vi ikke behov for den bedste præcision, da dronen filmer brugerne fra en given højde og derfor har en hvis vidvinkel mod brugerne.

3.6 Security View

I dette projekt har sikkerheden ikke første prioritet og derfor er der også foretaget valg som har haft en negativ indflydelse på sikkerheden.

3.6.1 Website sikkerhed

Da systemet benytter en website, som kommunikationsformatet mellem applikationen og dronen er kommunikationen til og fra websitet blottet.

3.6.1.1 HTTP vs HTTPS

Forskellen på HTTP og HTTPS er at sidstnævnte først opretter en forbindelse og derefter sker der en udveksling af nøgler til kryptering af data. På denne måde er forbindelsen krypteret og derved også mere sikker end almindelig HTTP. Da HTTP ikke er krypteret er denne sårbar overfor "man-in-the-middle attack" og andre former for "eavesdropping". Skulle dette projekt lanceres skulle denne forbindelse krypteres, da dronernes information sendes gennem en hjemmeside. Hvis andre kunne begynde at sende beskeder til dronerne ville dette være en sikkerhedsbrist. [29]

3.6.2 Login

En yderligere sikkerhed kunne bestå af et login system. Alle brugere af systemet skal registreres gennem applikationen med et brugernavn og en kode. Disse informationer kunne gemmes i en tabel i systemets database. På denne måde kunne disse informationer være med til at gøre systemet mere sikkert, således at en hver med en smartphone ikke kan "stjæle" en drone.

3.6.3 Dronen

Dronen har en risiko faktor, da denne har 4 motorer med propeller, som spinder med en høj hastighed. Når disse er aktive er dronen farligt at være i nærheden af.

3.6.3.1 Stop signal

På dronen er et stop signal implementeret til at stoppe dronens kode. Dette skal give brugeren en sikkerhed i, at dronen ikke kan starte sine motorer, når brugeren er i nærheden af dronen. Bruger kan sende stop signalet fra applikationen på alle tidspunkter. Når dronen modtager et stop signal, starter denne tråden til at lande dronen. Såfremt dronen allerede er på jorden, afsluttes tråden med det samme igen, da afstanden til jorden er under 30 centimeter. Når dronen er landet når programmet på dronen slutningen og terminerer. Når koden er termineret, har dronen ingen mulighed for, at aktivere motorene, da den serielle kommunikation til Arduino Nano'en er termineret.

Den eneste måde at aktivere dronen på igen, er at frakoble batteriet og forbinde dette igen. Dette genstarter drones Raspberry Pi 3, som igen starter programmet under opstart.

3.6.3.2 Manglende GPS-signal under flyvning

I dronens kode er der implementeret et tjek på, om dronen får et valid GPS-signal. Hvis dronen under opstart ikke kan oprette et GPS-fix, vil dronen lukke ned igen. Hvis dronen er startet og er i luften og den mister sit GPS-fix, vil dronen lande på stedet og lukke ned. Dette er gjort af sikkerhedsmæssige hensyn til både brugeren og dronen, da dronen ikke vil kunne flyve sikkert uden GPS-fix.

3.6.3.3 For lav batterispænding under flyvning

Dronen monitorerer sin egen batterispænding kontinuerligt. Hvis batterispændingen kommer under 9.5 volt, vil dronen lande og lukke ned. Dette er gjort for, at dronen ikke skal aflade batteriet helt og pludselig styrte ned. Batterispændingen bliver kontinuerligt sendt i statusbeskederne til brugeren. Derved kan brugeren følge med i hvad spændingen på batteriet er.

3.7 Implementation View

I dette View vises det hvordan systemet skal opsættes, såfremt projektet er tilgængeligt. Systemets souce kode er vedlagt i bilag. I dette afsnit vil vigtige dele af projektet blive fremhævet og blive beskrevet i større detaljer.

3.7.1 Server

For at kunne arbejde videre på projektets server, skal man først have følgende programmer installeret.

- Visual Studio 2010 Update 3 [30]
- Microsoft .NET Core 1.0.1 VS 2015 Tooling Preview 2 [31]
- Microsoft SQL Server Management Studio [23]

Når disse programmer er installeret kan koden åbnes. Inden koden kan testes skal der opsættes en database, hvis ikke projektets database stadig eksisterer. Såfremt denne eksisterer kan koden afprøves med det samme i en lokal browser ved at trykker på F5. Dette åbner i en browser på den lokale pc. Her kan der navigeres til følgende stier:

1. `http://localhost:XXXXX/api/shadowbac?time=gettime`
2. `http://localhost:XXXXX/api/shadowbac?identifier=appXXXX&command=KOMMANDO`
3. `http://localhost:XXXXX/api/shadowbac?identifier=droneXXXX&command=KOMMANDO`
4. `http://localhost:XXXXX/api/shadowbac`

Sti 1, vil returnere serverens tid og dato indsat i en ellers tom besked. Sti 2, vil kun returnere den seneste besked modtaget fra applikationen, som benytter identifier = appXXXX. Parameteren KOMMANDO vil blive ignoreret. Det samme gælder for sti 3. Denne vil blot returnere seneste besked fra dronen som benytter identifier = droneXXXX, men her returneres den seneste besked med den specifikke KOMMANDO. Hvis en tom string sendes med i KOMMANDO returneres den nyeste besked blot.

Sti 4 skal bruges hvis man ønsker at gemme en ny besked i serverens database. Der skal opsættes en korrekt HTTP POST besked med et JSON element i HTTP body'en. Dette kan eventuelt gøres gennem programmet Postman [32].

Hvis ikke projektets database eksisterer mere skal der oprettes en database eller benyttes en lokal database. I dette projekt er en SQL Server blevet benyttet. Denne type af databaser kan modificeres gennem Microsoft SQL Server Management Studio eller gennem en service udbyder. I denne oprettede database skal der laves følgende ting:

- Oprettes 2 tables
- Oprettes 5 procedures

Tables

Der skal oprette to identiske tables, som begge skal indeholde følgende:

Variable navn	Type
Id (Primary Key)	int
Identifier	VARCHAR(50)
Command	VARCHAR(50)
Date	VARCHAR(50)
Time	VARCHAR(50)
GPSLong	float
GPSLat	float
Altitude	float
Angle	float
Distance	float
Heading	float

Tabel 3.12: Indhold i databasens tables

De skal have navnene: ShadowMessagesApp og ShadowMessagesDrone.

Procedures

Der skal nu laves 5 procedurer til at hente fra og indsætte i de to tables.

GetLastAppMessage og GetLastDroneMessage:

Disse to er ens foruden at de læser fra hver sin table. Denne procedure returnere den seneste besked modtaget fra applikationen eller dronen med den Identifier som er givet med som parameter. SQL koden til at oprette disse procedurer ses i Listing 3.3.

```
CREATE PROCEDURE usp_ShadowBACMessages_GetLastAppMessage
(
    @Identifier VARCHAR(50)
)
AS
BEGIN
    SELECT TOP 1* FROM ShadowMessagesApp WHERE Identifier = @Identifier ORDER BY Id
    DESC;
END
```

Listing 3.3: GetLastAppMessage

GetLastDroneMessage læser blot fra ShadowMessagesDrone istedet og navnet på proceduren ændres til usp_ShadowBACMessages_GetLastDroneMessages.

GetLastDroneMessageSpecific:

Denne procedure returnerer den seneste besked modtaget fra dronen med den specifikke *Identifier* og *Command* angivet. Denne bruges primært af applikationen til at requeste den seneste besked fra dronen mellem alle statusbeskederne. SQL koden til at oprette denne procedure ses i Listing 3.4.

```
CREATE PROCEDURE usp_ShadowBACMessages_GetLastDroneMessageSpecific
(
    @Identifier VARCHAR(50),
    @Command VARCHAR(50)
)
AS
```

```

BEGIN
    SELECT TOP 1 * FROM ShadowMessagesDrone WHERE Identifier = @Identifier AND
        Command = @Command ORDER BY Id DESC;
END

```

Listing 3.4: GetLastDroneMessageSpecific

AddNewAppMessage og AddNewDroneMessage:

Denne procedure modtager parametrene til en ny besked som skal gemmes og gemmer denne besked i ShadowMessagesApp. SQL koden til at oprette disse procedurer ses i Listing 3.5.

```

CREATE PROCEDURE usp_ShadowBACMessages_AddNewAppMessage
(
    @Identifier VARCHAR(50)
    ,@Command   VARCHAR(50)
    ,@Date      VARCHAR(50)
    ,@Time      VARCHAR(50)
    ,@GPSLong   FLOAT
    ,@GPSLat    FLOAT
    ,@Altitude  FLOAT
    ,@Angle     FLOAT
    ,@Distance  FLOAT
    ,@Heading   FLOAT
)
AS
BEGIN
    INSERT INTO ShadowMessagesApp
        ([ Identifier ]
        ,[ Command]
        ,[ Date]
        ,[ Time]
        ,[ GPSLong]
        ,[ GPSLat]
        ,[ Altitude]
        ,[ Angle]
        ,[ Distance]
        ,[ Heading])
    VALUES
        (@Identifier
        ,@Command
        ,@Date
        ,@Time
        ,@GPSLong
        ,@GPSLat
        ,@Altitude
        ,@Angle
        ,@Distance
        ,@Heading)
END

```

Listing 3.5: AddNewAppMessage

AddNewDroneMessage gemmer blot beskeden i ShadowMessagesDrone istedet og navnet på proceduren ændres til usp_ShadowBACMessages_AddNewDroneMessage.

Når disse procedurer og tables er oprettet skal databasens "Connection String" sættes i Visual Studio. Der er flere måder at finde denne "Connection String" på, den nemmeste er følgende.

I Visual Studio åbnes fanen "View" og her vælges "Server Explorer". Dette vil åbne et vindue til venstre i programmet. Her højre klikkes der på "Data Connection" og "Add Connection" vælges.

Hvis en ekstern database bliver brugt skal serveren hvorpå denne ligger angives i feltet "Server name". Authentication vælges til at være "SQL Server Authentication" og login oplysningerne indtastes. I dropdown menuen "Select or enter a database name" vælges den korrekte database fra listen af muligheder. Herefter kan forbindelsen afprøves med knappen "Test Connection" og efter dette trykkes der på OK. Databasen er nu tilføjet således at denne kan tilgås gennem Visual Studio også. Marker den tilføjet database og kig i "Properties" nederst i højre hjørne af Visual Studio. Her skulle gerne være et felt med navnet "Connection String". Marker og kopier alt hvad der står i dette felt. Går til filen "Web.config" og erstat den allerede eksisterende "Connection String" med den nye.

Bruges en lokal database følges de samme trin som allerede nævnt dog skal Authentication vælges til at være "Windows Authentication" når databasen tilføjes i Visual Studio.

I dette projekt er der arbejdet med en SQL Server da dette er den Visual Studio kan arbejde sammen med direkte. Ved at installere yderligere plugins kan denne også arbejde med MySql og andre typer af databaser dog er dette ikke prøvet i projektet.

Til at hoste projektets Wep Api og database blev det valgt at benytte en Cloud Hosting service. I projektet er GearHost[33] blevet benyttet da denne har en gratis udgave, som indeholde både websites og databaser. Begrænsningen ligger primært på databaserne som maksimalt må fylde 10MB. I dette projekt er dette dog ingen begrænsning da beskeder fylder meget lidt og dette er det eneste som skal ligge i databasen. Databasen er oprettet gennem deres hjemmeside og modificeret gennem Microsoft SQL Server Management Studio[23].

Når projektet skal uploades til serveren gøres dette gennem Visual Studio. I dette projekt hvor GearHost er benyttet kan der hentes en Publish fil fra deres hjemmeside som kan benyttes direkte i Visual Studio. Ved at højre klikke på projektet i Visual Studio kan punktet "Publish" vælges i menuen. Herefter vælges "Import" og filen hentet fra GearHost importeres. Herefter er det blot at trykke "Publish". Wep Api'en bliver nu lagt op på det website som er registeret hos GearHost.

3.7.2 Drone

Efter at have konfigureret dronens mekaniske opsætning jf. AeroQuad's manual [2], er næste skridt at ligge softwaren på Arduinoen. Til at gøre dette skal et par forskellige værktøjer hentes og installeres.

Arduino IDE 1.0.6 [34]

Dette tool benyttes til at installere de nødvendige drivers til Arduinoen og hvis brugeren ønsker at lave ændringer i softwaren fra AeroQuad. Det er vigtigt at dette er version 1.0.6 da der ellers vil opstå problemer med at opfører koden til Arduinoen gennem Configuratoren.

AeroQuad Configurator v3.2 [35]

Dette er et tool udviklet af AeroQuad til at forsimple indstillingen af dronens software før

denne ligges på Arduinoen. Dette tool kan bruges til en masse test før dronen er klar til at flyve.

AeroQuad Flight Software v3.2 [35]

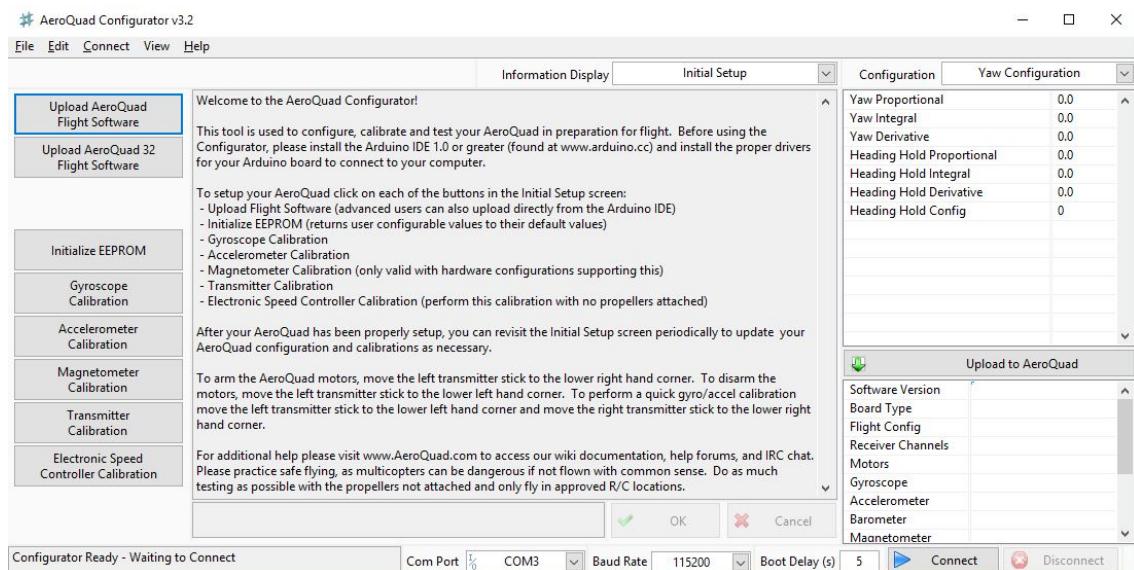
Selve koden fra AeroQuad til at styre dronen i luften. Det er vigtigt at version på AeroQuad Configuratoren og Flight Software er ens, i dette tilfælde v3.2.

AeroQuad v2.1.3 Shield

Denne skal samles såfremt denne ikke er det allerede. Her kan manualen på AeroQuad's hjemmeside [2] igen benyttes. Når denne er samlet kan Arduinoen og shield'et sættes sammen.

Upload af Flight Software

Hvorvidt man benytter Arduino IDE eller AeroQuad's Configurator til at lægge koden over skal der laves ændringer. Hvis Arduino IDE benyttes skal ændringerne laves direkte i source filerne, hvorimod hvis AeroQuad Configurator benyttes kan indstillingerne laves direkte i dette tool. I dette projekt er det valgt at benytte AeroQuad's Configurator til at lægge koden over. Først startes AeroQuad Configuratoren og følgende vindue fremkommer.

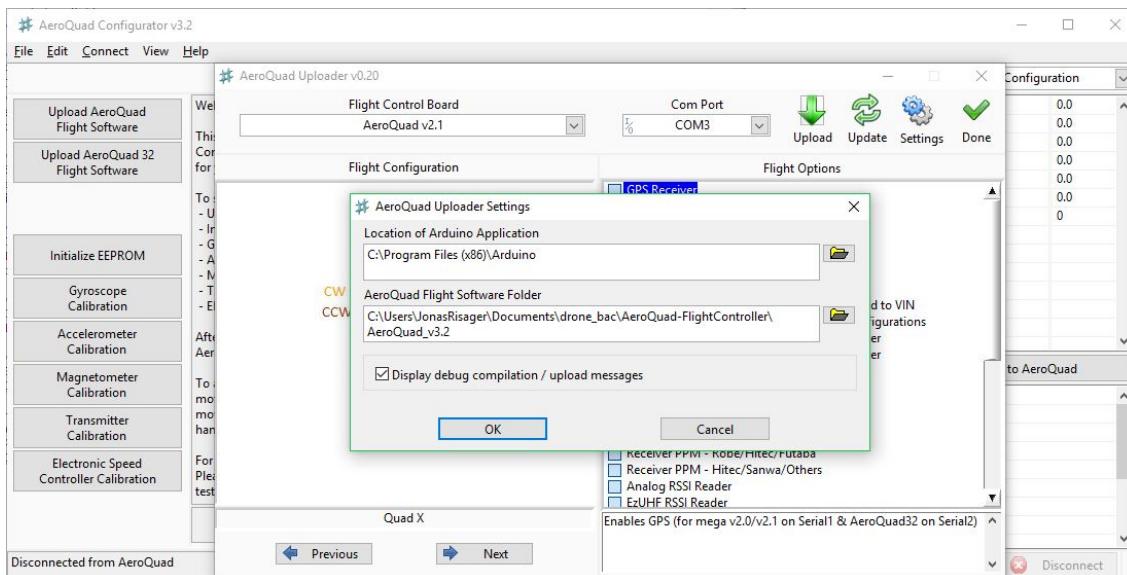


Figur 3.64: AeroQuad Configurator - Initial setup

Dette er hovedskærmen i AeroQuad's Configurator. I dropdown menuen "Information Display" kan forskellige vinduer vælges. Når der trykkes på knappen "Upload AeroQuad Flight Software" åbner følgende vindue.

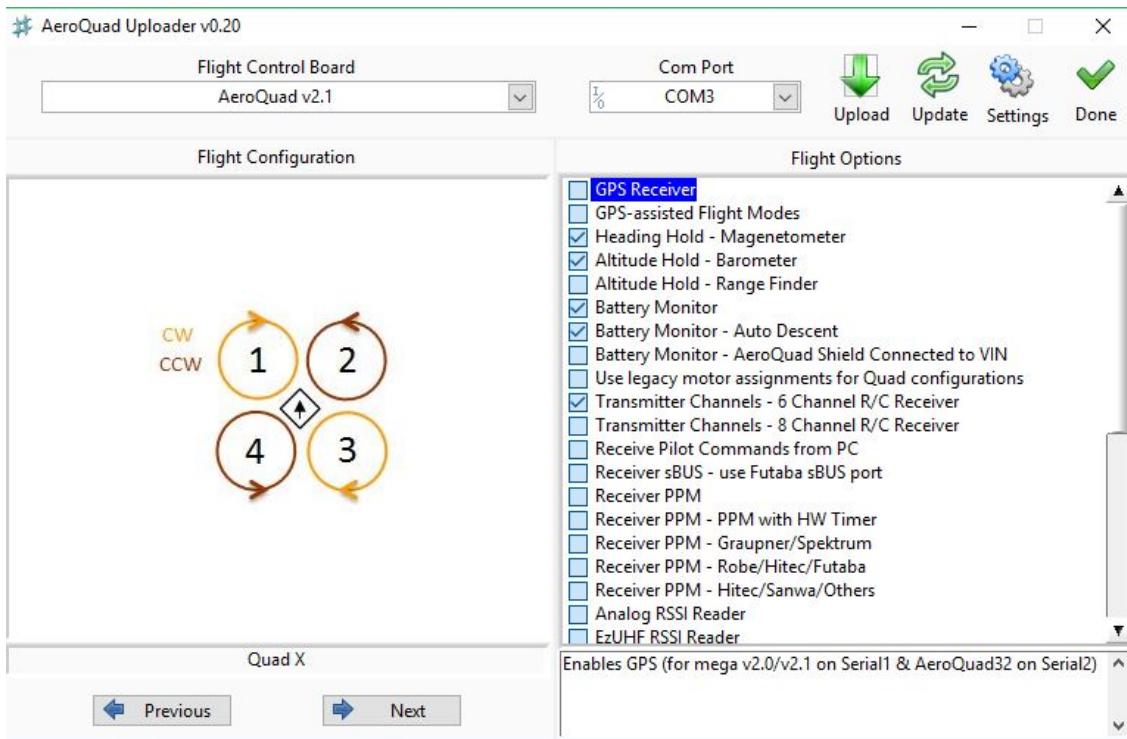
3.7. Implementation View

ASE



Figur 3.65: AeroQuad Configurator - Første upload

Hvis det er første gang man skal uploadere software kommer der et pop op vindue, som ses på figur 3.65. Her skal stien til to mappe angives. Stien til mappen, som indeholder arduino IDE applikationen, og stien til mappen med Flight Software koden. AeroQuad Configuratoren brokker sig hvis en mappe ikke er korrekt. På denne måde er man sikker på at vælge de korrekter mapper. Når dette er gjort trykkes på "OK" og vinduet lukker og følgende vindue bliver synligt.

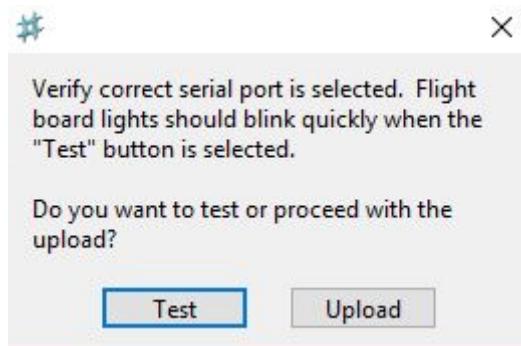


Figur 3.66: AeroQuad Configurator - Valg af funktionalitet

I figur 3.66 laves de sidste indstillinger inden koden lægges på Arduinoen. Først vælges det "Flight Control Board" som man benytter, i dette tilfælde et AeroQuad v2.1. Herefter vælges den korrekte COM port. Nu vælges den "Flight Configuration" som man ønsker. I dette projekt er der benyttet et Quad X setup. Dette skal stemme overens med opbygningen af stellet. I vinduet til højre er der en masse "Flight Options". Disse kan vælges fra eller til. Dette er afhængigt af hvad brugeren ønsker at benytte sin drone til. I dette projekt er følgende valgt:

- Battery Monitor
- Transmitter Channels - 6 Channel R/C Receiver

Når de valgte options er valgt trykkes der på knappen "Upload". Der fremkommer et vindue som vises på figur 3.67.



Figur 3.67: AeroQuad Configurator - Test

Her er det en god ide at trykke på knappen "Test" for at verificere at computeren har fået i Arduinoen. Dette ses ved at den grønne LED på shield'et blinker et par gange. Hvis denne ikke blinker er den korrekte COM port ikke valgt. Hvis LED'en blinker forsættes uploaden, der fremkommer et konsol vinduer, som gør det muligt at følge uploaden. Tilsidst fremkommer beskedet at Flight Software er uploadet til Arduinoen.

Error 129

Hvis konsol vinduet skriver at en fejl opstod og giver en fejlkode 129, skal følgende gøres:

- Download denne .dll filen. [36]
- Gå til hvor Arduino IDE er installeret på din computer.
- Eksempel:
 - C:\ProgramFiles(x86)\Arduino\arduino-1.0\hardware\tools\avr\utils\bin
- Omdøb filen msys-1.0.dll til msys-1.0.dll.bak (for at have denne som backup).
- Indsæt den downloaded .dll fil.

Dette burde løse problemet. Koden kan nu ligges på FlightControlleren.

Initialisering og kalibrering

Koden er nu lagt på Arduinoen og dronen kan kalibreres. Dette kan gøres gennem Configuratoren Initial Setup vindue, som ses på figur 3.64. I venstre side finder man følgende knapper:

- Initialize EEPROM
- Gyroscope Calibration
- Accelerometer Calibration
- Magnetometer Calibration
- Electronic Speed Controller Calibration

Disse køres i den rækkefølge, som de er listet. Anvisningerne i programmet følges gennem kaliberingen af de forskellige ting. Når disse kalibreringer er kørt, er dronen indstillet med defaultværdier i PID reguleringen.

3.7.3 DroneControllerUnit

Dette afsnit beskriver opsætningen af Raspberry Pi 3 med forklaringer af de vigtigste biblioteker samt en forklaring af de vigtigste funktioner, der bruges af DroneControllerUnit'en.

3.7.3.1 Raspberry Pi 3

Hele projektet er samlet i en zip-fil. I zip-filen ligger blandt andet et image til Raspberry Pi 3. Dette image kan flashes ned på et SD-kort, med en størrelse på minimum 16GB, vha. windowsprogrammet Win32DiskImager [37]. I dette image ligger hele styresystemet til Raspberry Pi 3 og al den opsætning der er lavet i styresystemet for at få dronen til at fungere. Der er brugt nogle specielle scripts for at få styresystemet til at starte Dronens kode og 3G-donglen fra boot-up. Dette bliver beskrevet yderligere i 3.7.3.2.

3.7.3.2 Scripts

Der benyttes et bash-script til at skifte mode på USB 3G-donglen. Dette er nødvendigt for at styresystemet på Raspberry Pi'en registerer donglen som en 3G dongle og ikke som en lagerenhed. Dette script er kaldet huawei3gstartup og køres lige efter boot-up. Dette opnåes ved at ligge scriptet i /etc/init.d og gøre det eksekverbar vha. chmod.

```
sudo chmod +x huawei3gstartup
```

Listing 3.6: Gøre bash-script eksekverbart

```
#!/bin/bash
sudo usb_modeswitch -v 12d1 -p 157d -J
```

Listing 3.7: huawei3gstartup

Et andet script der bliver brugt hedder DroneExecute.sh. Dette script bruges til at starte koden fra terminalen og at køre koden fra boot-up. Scriptet indeholder en kommando til at eksekvere koden fra den lokation som NetBeans gemmer koden i.

```
#!/bin/bash

sudo /home/pi/.netbeans/remote/192.168.2.3/desktop-re1b7e8-Windows-x86_64/C/
    Users/Mathias/Documents/drone_bac/BAC/Code/DroneController/DroneController

exit 0
```

Listing 3.8: DroneExecute.sh

For at kunne køre dette script fra boot-up tilføjes der blot en enkelt linje til et script der hedder rc.local. Dette script ligger allerede i styresystemet fra start. Linjen der tilføjes er /home/pi/DroneExecute.sh.

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

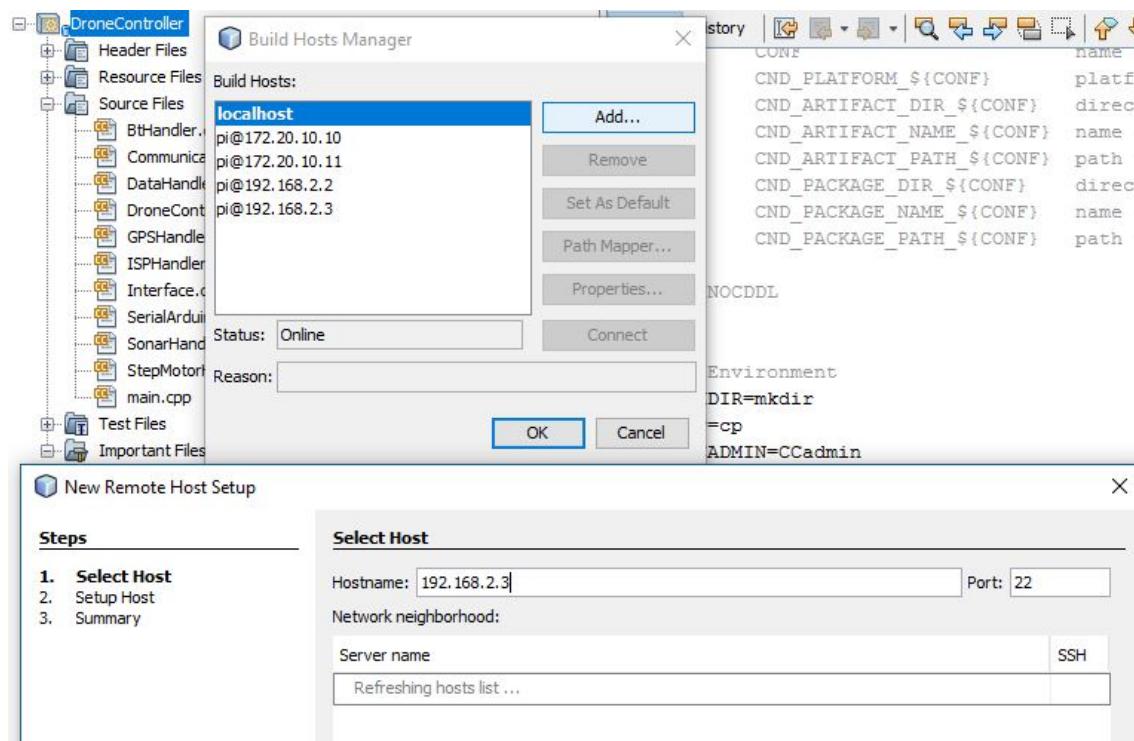
# Starting the ShadowBAC-X1 program
/home/pi/DroneExecute.sh

exit 0
```

Listing 3.9: rc.local

3.7.3.3 IDE

Som nævnt i 3.7.3 benyttes Netbeans 8.1 som software udviklingsmiljø. Dette skal sættes op til C++ samt at kunne fjerncompile på Raspberry Pi'en vha. en SSH forbindelse. Når C++ projektet er oprettet, kan der indstilles hvilken build host, der skal bruges.



Figur 3.68: Build Host

Her vælges hostname til den IP-adresse som Raspberry Pi 3 har på det lokale netværk, og dernæst skal der angives et brugernavn og adgangskode. Dette er som standard henholdsvis pi og raspberry.

I dette projektet er der benyttet g++ compileren i version 4.9.2 på RPi'en.

Makefilen, der er nødvendig for at kunne bygge projektet, skal indeholde nogle specielle flag, for at kunne benytte de nødvendige biblioteker.

Foruden at compileren skal vælges til g++ skal der bruges følgende flag i makefilen.

```
CXXFLAGS = -Wall -g -lwiringPi -pthread -std=gnu++11 -lboost_system -lbluetooth  
-DELPP_THREAD_SAFE
```

Listing 3.10: makefile flags

Det er muligt, at bygge projektet både fra NetBeans 8.1 og på selve Raspberry Pi'en. Såfremt det ønskes, at bygge projektet gennem terminalen på Raspberry Pi'en, navigeres der til mappen hvor alle sourcefilerne og makefilen ligger. Her skal make-kommandoen udføres uden nogen parametre.

```
make
```

Listing 3.11: make-kommando

3.7.3.4 Biblioteker

Der er brugt en række specielle biblioteker i koden, som ikke er standard i C++. I det følgende afsnit vil det kort blive beskrevet hvilke biblioteker, der er brugt og hvor disse biblioteker, er blevet brugt henne.

Biblioteker:

- Boost
- Bluetooth
- JSON
- wiringPi
- EasyLogging++

Boost biblioteket *boost/asio.hpp* bliver brugt til 3G-kommunikationen imellem serveren og dronen. Dette bibliotek bliver inkluderet i *ISPHandler*'en og tager udgangspunkt i eksempekkoden fra documentationen for boost/asio [38]. Boost bibliotekerne, *boost/algorithms/string.hpp* og *boost/system/error_code.hpp*, bliver også brugt. Disse bliver henholdsvis inkluderet i *DroneController*'en og *ISPHandler*'en.

Bluetooth biblioteket bliver brugt i koden, når bluetooth modulet på Raspberry Pi'en skal tilgås. Dette bibliotek bliver inkluderet i *BtHandler*'en, hvor der bliver brugt socket-programmering. Der er taget udgangspunkt i koden fra [39].

JSON biblioteket bliver brugt til at håndtere JSON-pakker. Biblioteket er blot en headerfil, som skal placeres i mappen med alle sourcefilerne. For at benytte den i koden, skal den inkluderes ligesom alle andre headerfiler. Derudover skal der tilføjes *using json = nlohmann::json;* under inkluderingen. I makefilen skal der tilføjes *-std=c++11*. Headerfilen kan hentets på github: <https://github.com/nlohmann/json>. Dette bibliotek bliver inkluderet i *BtHandler*'en, *DataHandler*'en, *ISPHandler*'en og i *main*.

WiringPi[40] er et bibliotek, der er udviklet specielt til Raspberry Pi'ens GPIO interface. Dette bibliotek indeholder en lang række funktioner, der gør det muligt at bruge GPIO'erne på RPi'en i C++. Dette bibliotek hentes fra <http://wiringpi.com/download-and-install/>. Biblioteket indeholder nogle underbiblioteker, som også bliver brugt. Disse er *wiringSerial.h* og *wiringPiI2C.h*. Bibliotekerne er inkluderet i *GPSHandler*'en, *Interface*'et, *SerialArduinoHandler*'en, *SonarHandler*'en og *StepMotorHandler*'en.

EasyLogging++ er et bibliotek til C++, der gør det muligt at lave log-beskeder og gemme disse i log's, på en nem måde. Dette bibliotek er også blot en headerfil ligesom JSON biblioteket. Dette skal blot placeres i mappen med alle sourcefilerne og inkluderes i C++ projektet, der hvor det skal bruges. Der skal ydermere oprettes en .conf fil, hvori der bestemmes hvor log'en skal gemmes og hvilket format log-beskederne skal være i. Biblioteket skal initialiseres én gang ved hjælp af følgende macro, *INITIALIZE_EASYLOGGINGPP*. I *main* kaldes en funktion der konfigurerer alle loggerne. Loggerne konfigureres og initieres i følgende funktionskald, som ligger som en hjælpefunktion i *main*.

```

void initLoggers(){
    el :: Configurations general_conf("/home/pi/EasyLogging++/GeneralLog.conf");
    el :: Configurations gps_conf("/home/pi/EasyLogging++/gpsHandler.conf");
    el :: Configurations bt_conf("/home/pi/EasyLogging++/btHandler.conf");
    el :: Configurations data_conf("/home/pi/EasyLogging++/dataHandler.conf");
    el :: Configurations drone_conf("/home/pi/EasyLogging++/droneController.conf");
    el :: Configurations isp_conf("/home/pi/EasyLogging++/ispHandler.conf");
    el :: Configurations serial_conf("/home/pi/EasyLogging++/serialArduinoHandler.conf");
    el :: Configurations step_conf("/home/pi/EasyLogging++/stepMotorHandler.conf");
    el :: Configurations com_conf("/home/pi/EasyLogging++/communicator.conf");
    el :: Configurations test_conf("/home/pi/EasyLogging++/droneClassTester.conf");
    el :: Configurations interface_conf("/home/pi/EasyLogging++/interface.conf");
    el :: Configurations sonar_conf("/home/pi/EasyLogging++/sonarHandler.conf");

    el :: Loggers::getLogger("general");
    el :: Loggers::reconfigureLogger("general",general_conf);

    el :: Loggers::getLogger("gps");
    el :: Loggers::reconfigureLogger("gps",gps_conf);

    el :: Loggers::getLogger("bt");
    el :: Loggers::reconfigureLogger("bt",bt_conf);

    el :: Loggers::getLogger("data");
    el :: Loggers::reconfigureLogger("data",data_conf);

    el :: Loggers::getLogger("drone");
    el :: Loggers::reconfigureLogger("drone",drone_conf);

    el :: Loggers::getLogger("isp");
    el :: Loggers::reconfigureLogger("isp",isp_conf);

    el :: Loggers::getLogger("serial");
    el :: Loggers::reconfigureLogger("serial",serial_conf);

    el :: Loggers::getLogger("step");
    el :: Loggers::reconfigureLogger("step",step_conf);

    el :: Loggers::getLogger("com");
    el :: Loggers::reconfigureLogger("com",com_conf);

    el :: Loggers::getLogger("test");
    el :: Loggers::reconfigureLogger("test",test_conf);

    el :: Loggers::getLogger("interface");
    el :: Loggers::reconfigureLogger("interface",interface_conf);

    el :: Loggers::getLogger("sonar");
    el :: Loggers::reconfigureLogger("sonar",sonar_conf);
}

```

Listing 3.12: InitLoggers()

3.7.3.5 Dokumentation

I koden på dronen benyttes der Doxygen[41] til at lave en dokumentation for alle funktionerne. Dette kræver, at der indsættes specielle kommentarer i koden. Disse kommentare kan Doxygen tilgå og lave en dokumentation ud fra. Der henvises til sourcefilerne for at se disse kommentarer. Doxygen installeres ved at køre følgende to kommandoer i terminalen:

```
sudo apt-get install doxygen
```

```
sudo apt-get install graphviz
```

Listing 3.13: Doxygen installering

Dernæst kan der genereres en konfigurationsfil med følgende kommando:

```
doxygen -g my_proj.conf
```

Listing 3.14: Generate konfigurationsfil

Navnet på konfigurationsfilen kan selv bestemmes. I konfigurationsfilen kan der nu sættes en lang række parametre, derved er det muligt at modificere dokumentationen til det ønskede. Den vigtigste ting er dog at tilføje stien til sourcefilerne i parameteren *INPUT*. Dette gør, at doxygen kan finde sourcefilerne. For at lave hele dokumentationen køres kommandoen:

```
doxygen my_proj.conf
```

Listing 3.15: Generate Dokumentation

Denne kommando vil generere to mapper, en html og en latex. I mappen html er der en fil kaldet index. Denne kan åbnes i en internet browseren for at få dokumentationen vist, som en lokal hjemmeside. Den anden mappe, benyttes ikke i projektet.

3.7.3.6 Funktioner

Der bliver brugt matematik for at regulere dronen. I det følgende vil de mest væsentlige matematiske formler og funktioner blive forklaret.

Funktionerne er implementeret som hjælpefunktioner til de forskellige tråde, der kører i *DroneController*'en. Dette gør, at de kun skal implementeres én gang, og kan dernæst bruges, hvor det er nødvendigt.

Funktionerne er verificeret i afsnit 1.2.3 i Modultesten. Denne findes i Modul- & Integrationstest i bilag.

Kurs ud fra to GPS-koordinater

For at kunne styre dronen hen til et GPS-koordinat, er det nødvendigt, at kunne beregne den nødvendige kurs mod dette GPS-koordinat. Dette gøres i den følgende udregning. Udregningen gør brug af atan2. Atan2 er en fætter til sin- og cos-funktionerne. Atan2 kan fortælle i hvilken kvadrant vinklen er. Dette er derfor meget belejligt, når der skal beregnes en kurs.

```

1 float DroneController::radiansBetweenHeadingAndVector(float heading, float lonUser, float latUser, float
2           lonDrone, float latDrone) {
3     CLOG(DEBUG, "drone") << "RadiansBetweenHeadingAndVector";
4     CLOG(DEBUG, "drone") << setprecision(10) << "Lat User: " << latUser;
5     CLOG(DEBUG, "drone") << " Long User: " << lonUser;
6     CLOG(DEBUG, "drone") << " Lat Drone: " << latDrone;
7     CLOG(DEBUG, "drone") << " Long Drone: " << lonDrone;
8     CLOG(DEBUG, "drone") << " Heading: " << heading;
9
10    float angle;
11    double a = latDrone * PI / 180;
12    double b = lonDrone * PI / 180;
13    double c = latUser * PI / 180;
```

```

13     double d = lonUser * PI / 180;
14
15     double angleCritical = cos(c) * sin(d - b);
16     if (closeToZero(angleCritical))
17         if (c > a)
18             angle = 0;
19         else
20             angle = 180;
21     else {
22         angle = atan2(cos(c) * sin(d - b), sin(c) * cos(a) - sin(a) * cos(c) * cos(d - b));
23         angle = (int) round((angle * 180 / PI + 360)) % 360;
24     }
25
26     angle = angle - heading;
27     if (angle < 0) {
28         angle = angle + 360;
29     }
30     float angleRad = angle * PI / 180;
31
32     CLOG(DEBUG, "drone") << "The angle between drone and point: " << angle;
33     CLOG(DEBUG, "drone") << "The angle between drone and point in radians: " << angleRad;
34
35     return angleRad;
36 }
```

Listing 3.16: Beregning af kurs ud fra to GPS-koordinater

Udregningen i listing 3.16 er taget fra [42].

Funktionen tager 5 parametre, dronens kurs i grader, longitude og latitude på det punkt dronen skal flyve hen til og longitude og latitude på det punkt dronen befinder sig på. På baggrund af de to punkter, kan der beregnes en kurs mod dette punkt i forhold til nord. For at finde ændringen i kurven, trækkes dronens egen kurs fra den fundne kurs. Hvis denne giver et negativt tal bliver der lagt 360 grader til kurven, da en kurs i forhold til nord ikke kan være negativ.

Afstanden i mellem to GPS-koordinater

Udover at kende kurven hen imod et GPS-koordinat, er det også nødvendigt at kende afstanden. På baggrund af afstanden bliver dronens hastighed bestemt. Dette betyder at jo længere dronen er væk fra det ønskede GPS-koordinat, jo hurtigere flyver dronen. Dermed gør det også at jo tættere dronen er på GPS-koordinatet, jo langsommere flyver dronen.

For at finde frem til afstanden i mellem to GPS-koordinater er der brugt haversine formlen. Denne formel gør brug af sin- og cos-funktionerne til at finde vinklen imellem de to GPS-koordinater og så derefter omregne vinklen til en afstand vha. storcircbler.

```

1 float DroneController::distanceBetweenGPSCoordinates(float lonUser, float latUser, float lonDrone, float
2     latDrone) {
3     CLOG(DEBUG, "drone") << "DistanceBetweenGPSCoordinates";
4     CLOG(DEBUG, "drone") << setprecision(10) << "Lat User: " << latUser;
5     CLOG(DEBUG, "drone") << " Long User: " << lonUser;
6     CLOG(DEBUG, "drone") << "Lat Drone: " << latDrone;
7     CLOG(DEBUG, "drone") << " Long Drone: " << lonDrone;
8     float dz, dx, dy, result ;
9     lonUser = lonUser - lonDrone;
10    lonUser = lonUser * PI / 180;
11    latUser = latUser * PI / 180;
12    latDrone = latDrone * PI / 180;
13    dz = sin(latUser) - sin(latDrone);
```

```

14     dx = cos(lonUser) * cos(latUser) - cos(latDrone);
15     dy = sin(lonUser) * cos(latUser);
16     //Returns the distance in meters
17     result = asin(sqrt(dx * dx + dy * dy + dz * dz) / 2) * 2 * EARTH_RADIUS * 1000;
18
19     CLOG(DEBUG, "drone") << "Distance is: " << result << "m";
20     return result;
21 }
```

Listing 3.17: Afstand imellem to GPS-koordinater

Funktionen i listing 3.17 er taget fra [43].

Der er lavet enkelte ændringer i den originale funktion med hensyn til variabel navne. Derudover er resultatet omregnet til meter fra kilometer. Funktionen tager fire parametre, longitude og latitude på det GPS-koordinat dronen skal hen til og longitude og latitude på sit eget GPS-koordinat.

Flytning af GPS-koordinat på baggrund af kurs og afstand

Da hele systemet er bygget op omkring, at brugeren kan indstille en position til dronen, vha. en vinkel og en afstand og en højde, er det nødvendigt at kunne beregne et nyt GPS-koordinat, som dronen skal styre efter på baggrund af disse indstillinger fra brugeren. Højden har dog ikke noget at sige i forhold til det nye GPS-koordinat, da dronen vil styre efter longitude og latitude.

```

1 void DroneController::displacementLocation(float &longitude, float &latitude, float distance, float bearing) {
2     CLOG(DEBUG, "drone") << "DisplacementLocation";
3     CLOG(DEBUG, "drone") << setprecision(10) << "Lat: " << latitude;
4     CLOG(DEBUG, "drone") << " Long: " << longitude;
5     CLOG(DEBUG, "drone") << " Distance: " << distance;
6     CLOG(DEBUG, "drone") << " Heading: " << bearing;
7
8     longitude = longitude + ((distance * sin(bearing * PI / 180) / cos(latitude * PI / 180)) / 111111);
9     latitude = latitude + (distance * cos(bearing * PI / 180) / 111111);
10    CLOG(DEBUG, "drone") << setprecision(10) << "New Lat: " << latitude << " New Long: " <<
11        longitude;
12 }
```

Listing 3.18: Flytning af GPS-koordinat på baggrund af kurs og afstand

Funktionen er taget fra [44]

Da afstanden, dronens eget GPS-koordinat og det GPS-koordinat, som brugeren vælger baseret på vinkel og afstand, aldrig vil være mere end 100 meter, kan der uden problemer ses bort fra Jordens krumning og derved blot antage at jorden er flad. Dette er gjort i denne funktion. Funktionen kan derfor ikke bruges over større afstande, men i dette projekt er præcisionen tilstrækkelig.

Funktionen tager 4 parametre, brugerens longitude og latitude, den ønskede afstand fra brugeren og den kurs brugeren ønsker dronen i.

3.7.3.7 Regulering

På baggrund af funktionerne i 3.7.3.6 bliver dronen reguleret. Reguleringen sker i henholdsvis *takeOff*-tråden, *land*-tråden, *setPosition*-tråden og *follow*-tråden. Alt efter hvilken tråd der er i gang, bliver dronen reguleret forskelligt. Der er dog lavet fire hjælpefunktioner, der beregner den nødvendige ændring i motorkraften og sender denne til

flightController'en via Arduino Nano'en. Dette får dronen til at flyve op/ned, frem/tilbage, fra side til side og dreje rundt om sig selv.

De fire hjælpefunktioner vil blive gennemgået nedenfor.

adjustThrottle

Denne funktion bruges til at styre hastigheden på alle fire motorer. Det er denne funktion, der får dronen til at flyve op eller ned.

```

1 void DroneController::adjustThrottle( float userHeight, float droneHeight ) {
2     float pThrot = 0.005;
3     float throttleChange = 0;
4     throttleChange = 1 + ((pThrot * (userHeight - droneHeight)) / 100);
5     motorPower_ = (int) round(motorPower_ * throttleChange);
6
7     if (motorPower_ > 2000) {
8         motorPower_ = 2000;
9     } else if (motorPower_ < 1200) {
10        motorPower_ = 1200;
11    }
12    CLOG(DEBUG, "drone") << "Motorpower: " << motorPower_;
13    arduinoController_.setChannel(THRO_CHAR, motorPower_);
14 }
```

Listing 3.19: adjustThrottle

Denne funktion tager to parametre, brugerens ønskede højde for dronen og dronens nuværende højde.

Funktionen benytter en P-regulator til at regulere de fire motorer. P-leddet er relativt lavt, 0.005, for at sørge for at dronen regulerer langsomt op til den ønskede højde.

Funktionen vil beregne en fejl i forhold til den ønskede højde og motorerne bliver derved reguleret ind. Hvis der ingen forskel i brugerens valgte højde og dronens nuværende højde, så vil motorkraften forblive den samme. Motorkraften for motorerne kan ikke blive højere end 2000 eller lavere end 1200. Motorkraften kan reelt gå fra 1050 til 2000. Men da dronen først kan begynde at lette ved en motorkraft på over 1200, så er der valgt en nedre grænse for reguleringen på 1200.

adjustElevation

Denne funktion bruges til at styre forholdet i mellem motorkraften på de to forreste motorer og de to bagerste motorer. Der er dog tale om et forhold på baggrund af motorkraften reguleret i *adjustThrottle* 3.19. Dette får dronen til enten at flyve frem eller tilbage. Dette vil sige, at er motorkraften på 1500 på alle fire motorer, og dronen skal flyve fremad, så vil *adjustElevation*-funktionen regulere de forreste motorer en smule ned og de bagerste motorer lidt op. Dette vil give dronen en hældning fremad, som vil få den til at flyve fremad.

```

1 void DroneController::adjustElevation( float radiansToLocation, float distanceToLocation ) {
2     int elevationSpeed = 0;
3
4     elevationSpeed = (int) round((cos(radiansToLocation) * ELEVPOWERAMP * distanceToLocation) +
5                                  ELEVATIONDEFAULT);
6     if (elevationSpeed > 1600)
7         elevationSpeed = 1600;
8     if (elevationSpeed < 1400)
9         elevationSpeed = 1400;
10    CLOG(DEBUG, "drone") << "elevationSpeed: " << elevationSpeed;
```

```

10     arduinoController_.setChannel(ELEV_CHAR, elevationSpeed);
11 }
```

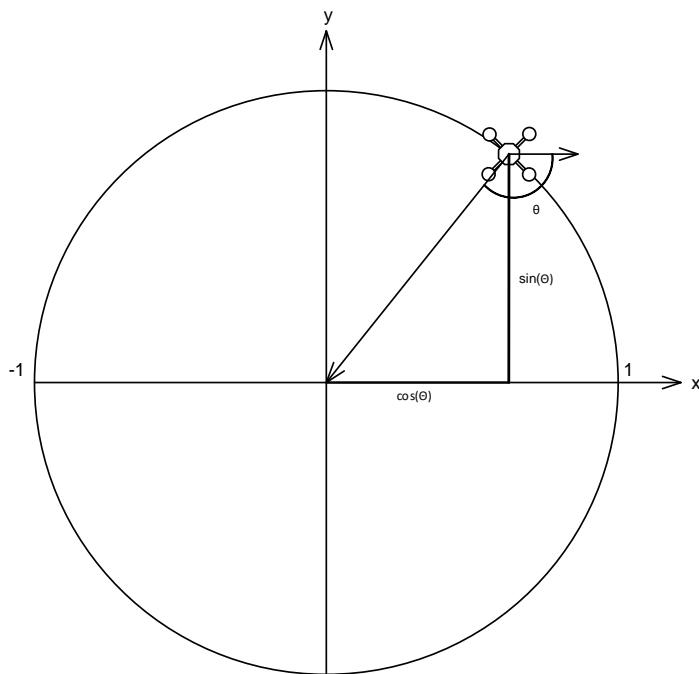
Listing 3.20: *adjustElevation*

Denne funktion tager to parametre, kursen til det ønskede GPS-koordinat og afstanden til GPS-koordinatet.

Kursen er i radianer, og bruges til at bestemme om dronen skal flyve frem eller tilbage. Dette bestemmes vha. cos-funktionen, da denne vil give et positivt tal, hvis kursen ligger i 1. eller 4. kvadrant, hvorimod den giver et negativt tal hvis kursen ligger i 2. eller 3. kvadrant. I beregningen af elevationspeed bruges $\cos(\text{kursen})$ og afstanden, samt nogle konstanter. Dette betyder, at giver $\cos(\text{kursen})$ et positivt tal og alt efter hvor tæt dette tal er på nul, samt hvor stor afstanden er til GPS-koordinatet vil hastigheden fremad bestemmes. Hvis $\cos(\text{kursen})$ giver et negativt tal, vil dronen skulle flyve bagud, og da afstanden altid vil være positiv, vil denne sammen med størrelsen af det negative tal, bestemme hastigheden bagud.

Variablen ELEVPOWERAMP er en define, som er sat til 10. Denne bruges til at forstærke ændringen i forholdet der ligges til ELEVATIONDEFAULT, som er sat til 1500. 1500 er defaultværdien og gør at dronen ikke flyver frem eller tilbage.

På figur 3.69 kan det ses hvordan $\cos(\text{kursen})$ vil ændre sig alt efter i hvilken kvadrant kursen ligger i. $\cos(\text{kursen})$ kan gå fra -1 til 1. Hastigheden fremad er sat til maksimalt at være 1600 og hastigheden bagud er sat til maksimalt til at være 1400.



Figur 3.69: Forklaring af dronens flyvereguleringer.

I tilfældet på figur 3.69 vil dronen udregne vinklen mellem dronens fremadretning og vectoren til punktet denne skal til. Herefter udregnes $\cos(\text{vinklen})$ for at vurdere om

dronen skal flyve frem eller tilbage. I dette tilfælde vil drone begynde at flyve tilbage, da cos(vinklen) giver et negativ tal.

adjustAile

Denne funktion bruges til at styre forholdet i mellem motorkraften på de to motorer på dronens venstre side og de to motorer på dronens højre side. Det er ligeledes et forhold på baggrund af motorkraften reguleret i *adjustThrottle* 3.19. Dette får dronen til enten at flyve til den ene side eller den anden side. Dette vil sige, at er motorkraften på 1500 på alle fire motorer, og dronen skal flyve til venstre, så vil *adjustAile*-funktionen regulere de to motorer på venstre side en smule ned og de to motorer på højre side lidt op. Dette vil give dronen en hældning til venstre, som vil få den til at flyve til venstre.

```

1 void DroneController::adjustAile( float radiansToLocation, float distanceToLocation ) {
2     int aileSpeed = (int) round((sin(radiansToLocation) * AILEPOWERAMP * distanceToLocation) +
3                                 AILEDEFAULT);
4     if (aileSpeed > 1600)
5         aileSpeed = 1600;
6     if (aileSpeed < 1400)
7         aileSpeed = 1400;
8     CLOG(DEBUG, "drone") << "aileSpeed: " << aileSpeed;
9     arduinoController_.setChannel(AILE_CHAR, aileSpeed);
}
```

Listing 3.21: adjustAile

Denne funktion tager de samme to parametre som i *adjustElevation* 3.20. Den eneste forskel er, at i stedet for cos-funktionen, bruger denne funktion sin-funktionen.

adjustRudder

Denne funktion bruges til at dreje dronen omkring sin egen vertikale akse. Dette gøres ved, at de to diagonale motorer bliver justeret op og de to andre bliver justeret ned. Derved vil dronen dreje rundt. Dette bliver brugt til at holde den rigtige kurs.

```

1 void DroneController::adjustRudder(float angleBetweenHeadings) {
2     int rudderSpeed = 0;
3     float pRudd = 0.05;
4     rudderSpeed = (int) round((1 + ((pRudd * angleBetweenHeadings)*(-1) / 100)) * RUDDERDEFAULT);
5     if (angleBetweenHeadings > 150 || angleBetweenHeadings < -150) {
6         if (rudderSpeed > 1700)
7             rudderSpeed = 1700;
8         if (rudderSpeed < 1300)
9             rudderSpeed = 1300;
10    } else {
11        if (rudderSpeed > 1600)
12            rudderSpeed = 1600;
13        if (rudderSpeed < 1400)
14            rudderSpeed = 1400;
15    }
16    CLOG(DEBUG, "drone") << "Rudderspeed: " << rudderSpeed;
17    arduinoController_.setChannel(RUDD_CHAR, rudderSpeed);
18 }
```

Listing 3.22: adjustRudder

Funktionen tager kun en parametere, en vinkel. Dette er vinklen mellem brugerens kurs og dronens kurs. Funktionen vil så på baggrund af denne vinkel styre hvor hurtigt dronen skal

dreje omkring sin egen vertikale akse. Hvis vinklen er større end 150 grader eller mindre end -150 grader, er det valgt at regulere endnu kraftigere, dette er gjort for at dronen hurtigere kommer ind på ret kurs.

3.7.3.8 Dronens Communicator

Communicator-klassen er dronens kommunikationsformidler med omverden. Denne klasse står for at håndtere al kommunikationen ind og ud af dronen. Dette omfatter følgende;

- GPS
- 3G
- Bluetooth

Communicator'en har en instans af *GPSHandler*-klassen, som denne har mulighed for at benytte. *Communicator*'ens hovedansvar er, at håndtere 3G og Bluetooth. Dette er gjort, således at der kun eksisterer én funktion til at modtage data og én funktion til at sende data. Klassen håndterer selv, hvorvidt dronen benytter 3G eller Bluetooth. På denne måde, er det ikke *DroneController*'ens ansvar, at benytte den korrekte funktion til at hente data.

På denne måde er koden også sværere at benytte forkert. Dette sikrer mod eventuelle fejl, ved brug af forkerte funktioner i koden.

3.7.3.9 Aeroquad seriell kommunikation

Kommunikationen mellem FlightController'en og dronens Raspberry Pi 3 er, som tidligere nævnt, seriell kommunikation. Da alt softwaren på FlightController'en er lavet af Aeroquad, er dette ikke et serielt interface, projektet har defineret. Gennem Aeroquad's dokumentation kunne relevante serielle kommandoer findes [45].

For at kunne benytte den serielle kommunikation på dronen er biblioteket wiringSerial.h benyttet. Dette er tidligere omtalt i afsnit 3.7.3.4. Dette giver muligheden for at åbne en seriell port i koden og sende karakterer igennem til FlightController'en. Jævnfør Aeroquad skal den serielle kommunikationen til FlightController'en køre med en baudrate på 115200. I projektet er følgende kommandoer benyttet;

Karakter	Beskrivelsen
'z'	Når FlightControlleren modtager denne karakter begynder denne kontinuerligt at sende højden fra barometeret eller en på monteret sonar tilbage. Her benyttes kun barometer værdien.
'\$'	Når FlightControlleren modtager denne karakter begynder denne kontinuerligt at sende spændingen på batteriet tilbage.
'j'	Når FlightControlleren modtager denne karakter begynder denne kontinuerligt at sende rå magnetometer værdier tilbage for x,y og z akserne.
'i'	Når FlightControlleren modtager denne karakter begynder denne kontinuerligt at sende accelerometer værdier tilbage for x,y og z akserne.
'X'	Når FlighControlleren modtager denne karakter stopper denne med at sende.

Tabel 3.13: Aeroquad serielle kommandoer

Ved at benytte en seriell kommunikationssniffer [46] kunne det verificeres, hvilke kommandoer Aeroquad's Configurator sender når denne opretter forbindelse til FlightController'en. Dette bliver ligeledes gjort i koden, som det første efter at den serielle port er åbnet. Følgende kommandoer skal sendes i denne rækkefølge;

- '!', 'X', 'a', 'b', 'c', 'd', 'v', 'e', 'f', 'n', 'p', 'X'

Dette sikre, at koden har en forbindelsen til FlightController'en, før denne fortsætter i koden.

3.7.3.10 Kompas tilt kompensation

FlightController'en har et magnetometer, som dronen benytter, når denne ikke følger applikationen. Da dronen kan tilte omkring flere akser, vil dette forstyrre den værdi, som kan udregnes på baggrund af de rå magnetometer værdier. Det er derfor nødvendigt, at lave en tilt kompensering af magnetometeret.

```

1 // accX, accY and accZ are measured A/D values assumed to be centered around zero
2 roll = atan2(accX, accZ);
3 pitch = atan2(accY, accZ);
4 // rawX, rawY and rawZ are measured values straight from the magnetometer
5 CMx = (rawX * cos(pitch)) + (rawY * sin(roll) * sin(pitch)) - (rawZ * cos(roll) * sin(pitch));
6 CMy = (rawY * cos(roll)) + (rawZ * sin(roll));
7 heading = ((atan2(CMy, CMx))*(180/PI))-90;
8
9 if(heading<0)
10    heading = 360 + heading;

```

Listing 3.23: Kompas tilt kompensation

Kode i Listing 3.23 er fundet på Aeroquad's forum [47], hvor stifteren af Aeroquad omtaler netop dette emne. Dette er ikke kode, der er udarbejdet i projektet, dog vigtigt at fremhæve, da det betyder en væsentlig mere stabil magnetisk kurs. Den magnetiske kurs benyttes, når dronen skal holde et given GPS-koordinat eller dreje sig ind i forhold til applikationens kurs. Den magnetiske kurs benyttes ikke ved forfølgelse af brugeren. Her er det kursten fra GPS'en, der benyttes.

3.7.3.11 Arduino Nano seriell kommunikation

Arduino Nano'en benyttes i systemet til at generere PWM signaler til FlightController'en. Dette giver muligheden for, at emulere systemets oprindelige transmitter og modtager. Koden på Arduino'en er lavet med biblioteket Servo [48] og Arduino's serielle bibliotek. Koden forbinder 5 pins til 5 Servo instanser og herefter kan udgangen sættes med funktion *writeMicroseconds*. Gennem et oscilloskop kunne udgangene på modtager måles, for at bestemme PWM signalet med transmitteren i forskellige positioner. Dette viste en periodetid på 20ms og en duty cycle mellem 5 og 10%. Arduino Nano'en kan modtage følgende serielle kommandoer;

Kommando	Beskrivelsen
TXXXX;	Denne kommando sætter PWM på "Throttle" signalet
AXXXX;	Denne kommando sætter PWM på "Aileron" signalet
EXXXX;	Denne kommando sætter PWM på "Elevator" signalet
RXXXX;	Denne kommando sætter PWM på "Rudder" signalet
GXXXX;	Denne kommando sætter PWM på "GEAR" signalet
S	Sender signal til FlightControlleren således at denne aktiverer motorene
O	Sender signal til FlightControlleren således at denne deaktiverer motorene

Tabel 3.14: Beskrivelser af kommandoer til Arduino Nano'en

Når enten T,A,E,R eller G sendes skal denne karakter have en værdi tilknyttet vist ved XXXX i tabel 3.14. Denne værdi indikerer, hvor længe signalet er højt i løbet af en periodetid. Hvis 1000 sendes, er signalet højt i et 1ms ud af 20ms, 2000 giver således et højt signal i 2ms ud af 20ms.

3.7.3.12 GPS data

Fra GPS kan der modtages forskellige typer af beskeder. Disse er defineret af NMEA og er efterhånden blevet standarden inden for GPS[49]. Dronen benytter to typer af disse beskeder, hhv. en GPGGA og en GPRMC. Grunden til, at dronen benytter netop disse to beskeder, er fordi de indeholder den information, dronen har behov for fra GPS systemet.

Dronen henter en GPGGA besked for at validere, hvorvidt dronen har et fix eller ej. Dette er oplyst i feltet "Quality" i beskedten. GPRMC beskeden benyttes, da denne indeholder dronens position i form af latitude og longitude. GPRMC beskeden indeholder også dronens kurs.

Listing 3.24 viser formatet af de to beskeder, dronen benytter.

```
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47
$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A
```

Listing 3.24: GPGGA

For at kunne finde et koordinat over alt på jorden er denne delt ind i henholdsvis latitude og longitude linjer. Latitude har 0° omkring ækvator, nord polen er derfor 90° N og syd polen 90° S. Longitude har 0° ned gennem Greenwich i sydøst London. Herfra kan der gås 180° W eller 180° E omkring Jordkloden.

Latitude og Longitude er i formatet "timer,minutter" i beskederne. For lettere at kunne arbejde med koordinaterne omregnes disse til "decimal grader" som er et andet format koordinaterne kan udtrykkes i. Dette gøres med følgende udregning [50].

$$D_{dec} = D + \frac{M}{60} + \frac{S}{3600}$$

Det sidste led undlades, da sekunderne og minutterne er samlet i et komma i beskederne. Hvis latitude og longitude, fra den viste GPGGA-beskeden, skulle omregnes, ville det se sådan ud:

$$\text{Latitude} = 48^\circ + \frac{07.038}{60} = 48.1173^\circ$$

$$\text{Longitude} = 011^\circ + \frac{31.000}{60} = 11.5167^\circ$$

Dronen kan således benytte disse komma tal i stedet for tallet fra beskeden.

3.7.4 Application

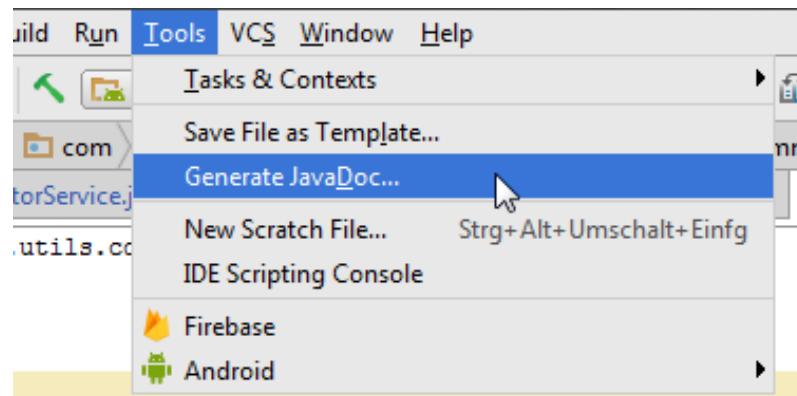
I det følgende afsnit vil opsætningen af projektet, samt implementeringen af de vigtigste dele af applikationen blive gennemgået.

3.7.4.1 IDE

For at programmere applikationen, benyttes Android Studio 2.2.2, hvor der både blev udviklet det grafiske interface, samt logikken bag dette. Under udviklen blev det valgt at debugge på en Android mobiltelefon. For at gøre dette skal mobiltelefonens indstillinger ændres således at Developer options bliver tilgængelig og USB debugging kan aktiveres.

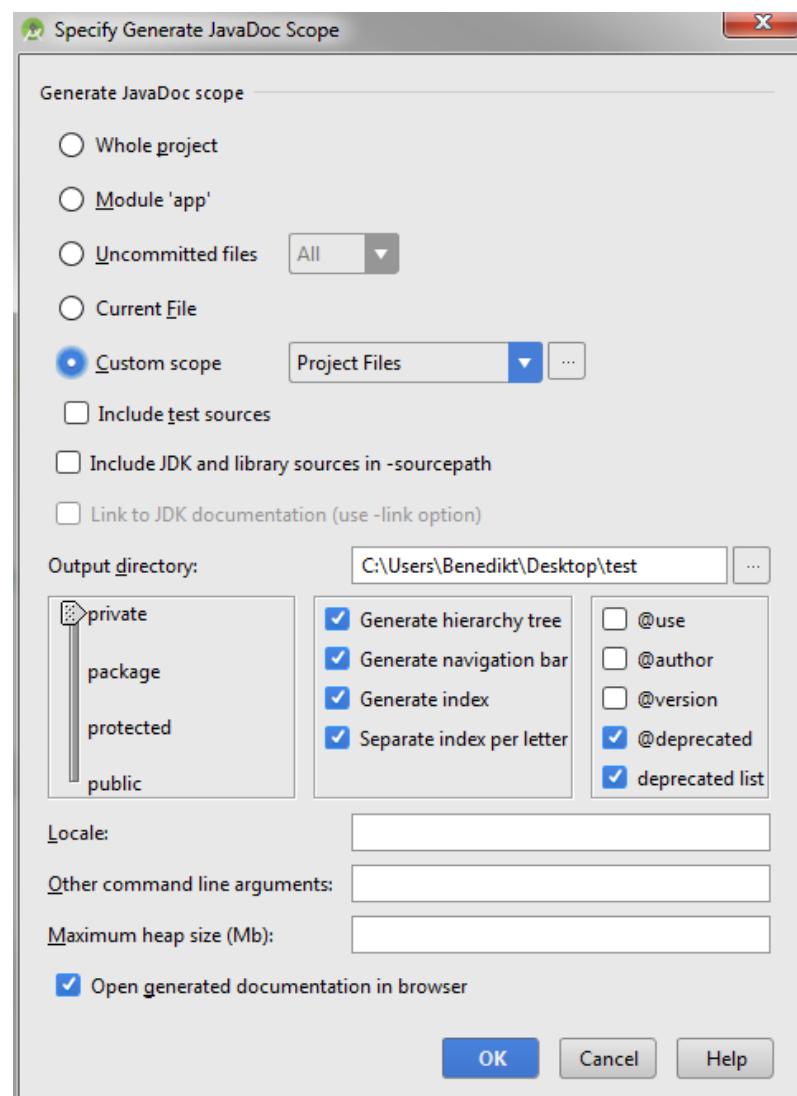
3.7.4.2 JavaDoc

For at lave funktions og klassebeskrivelser, blev der valgt at bruge Android Studio's tool JavaDoc [51]. For at åbne toollet, skal tool fanebladet åbnes og Generate JavaDoc vælges.



Figur 3.70: Start JavaDoc

For at generere en JavaDoc fil, der kun viser de filer, der blev udvilket i dette projekt, vælges følgende indstillinger:



Figur 3.71: JavaDoc indstillinger

Når der trykkes på OK, generes et JavaDoc dokument, der indeholder klasserne og funktionerne og deres beskrivelse, der blev tilføjet i koden med JavaDoc kommentarer.

```

1 /**
2  * Is called when the activity is visible to the user. It's responsibility is to bind to
3  * the CommunicatorService and start it
4  */
5 @Override
6 protected void onStart() {
7     super.onStart();
8     Intent intent = new Intent(this, CommunicatorService.class);
9     startService(intent);
10    bindService(intent, serviceConnection, Context.BIND_AUTO_CREATE);
11 }

```

Listing 3.25: Eksempel for JavaDoc kommentar

3.7.4.3 MainActivity

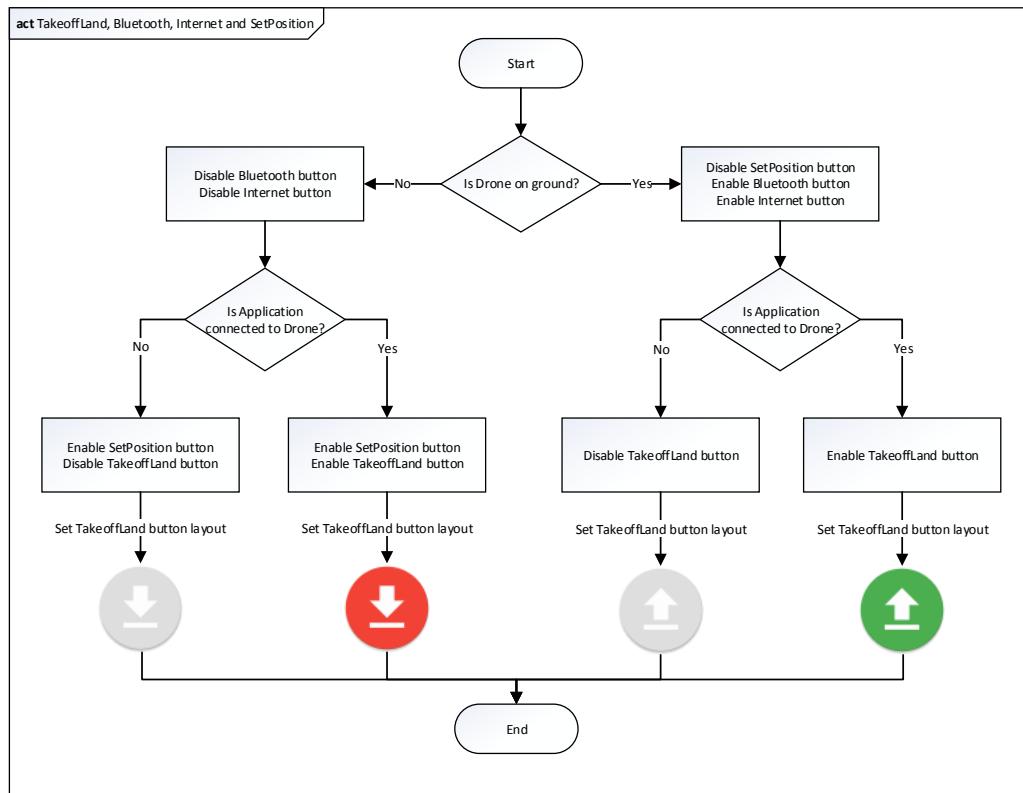
MainActivity står for at håndtere brugerens input og er dermed bindeleddet mellem brugeren og *CommunicatorService*, som er systemets backend. En stor del af *MainActivity*'s logik er at tilpasse UI'en, så den passer til systemts samlede tilstand.

Brugergrænseflade

Systemets tilstand bliver sat i en række forskellige status variabler. For at opdatere brugergrænsefladen løbende, kaldes den private funktion *uiUpdate()*, når der ændres i en af status variablerne. Funktionen er opdelt i fire dele:

- Håndterer knapperne TakeoffLand, Bluetooth, Internet og SetPosition.
- Håndterer Follow knappen.
- Styrer fejlbeskytterne og TryAgain knappen.
- Håndterer billedet der viser dronens batteristatus.

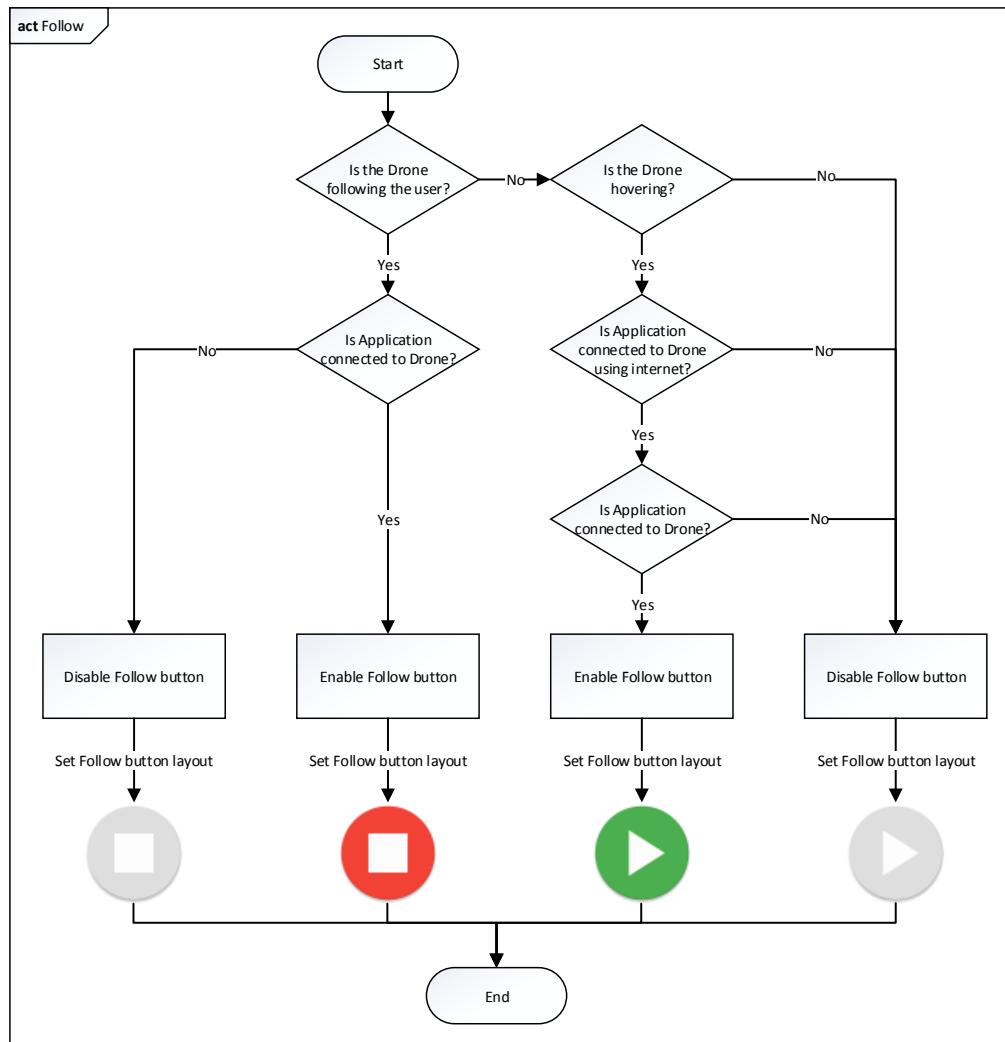
I diagram 3.72 ses en oversigt over den første del af funktionen.



Figur 3.72: Activity diagram, over første del af uiUpdate() funktionen

Som nævnt før, bliver knapperne deaktiviteret, når applikationen mister forbindelsen til dronen, for at signalere brugeren at han ikke har mulighed for at bruge funktionaliteten. Det ses på figur 3.72 at brugeren ikke har mulighed for at skifte forbindelsestype, når dronen er i luften, da dette vil føre til en mistet forbindelse.

I diagram 3.73, ses hvordan layoutet for Follow knappen bliver håndteret.



Figur 3.73: Activity diagram, over anden del af uiUpdate() funktionen

Det ses, at knappen ikke kan benyttes når applikationen er forbundet til dronen via Bluetooth. Denne funktionalitet blev valgt kun at være tilgængelig når applikationen er forbundet til dronen via internettet.

3.7.4.4 BtHandler

Bluetooth forbindelsen bliver etableret ved hjælp af en *BluetoothSocket* [52]. Klassen er trådsikker og kan derfor tilgås fra flere parallel kørende tasks. Der oprettes en sikker Bluetooth forbindelse til dronen.

```

1 public ErrorCode connect() {
2     BluetoothDevice btDevice = BluetoothAdapter.getDefaultAdapter().getRemoteDevice(macAddr);
3     try {
4         btSocket = btDevice.createRfcommSocketToServiceRecord(BT_UUID);
5     } catch (IOException e) {
6         e.printStackTrace();
7         return ErrorCode.DRONE_FAILED;
  
```

```

8     }
9     try {
10        // Connect to device, will throw exception when it fails
11        btSocket.connect();
12    } catch (IOException connectException) {
13        // Try to close the socket when no connection could be established
14        try {
15            btSocket.close();
16        } catch (IOException closeException) { return ErrorCode.DRONE_FAILED; }
17        return ErrorCode.DRONE_FAILED;
18    }
19    return ErrorCode.DRONE_CONNECTED;
20 }
```

Listing 3.26: Forbind til dronen via Bluetooth

Ved en sikker forbindelse bliver enhederne autentificeret og data krypteret, så der ikke kan være en Man-In-The-Middle.

Inden applikationen forbinder til dronen verificeres der dog, at Bluetooth er tændt og at dronen og mobiltelefonen er paired.

```

1 public ErrorCode checkConnection() {
2     BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();
3     if (btAdapter == null)
4         return ErrorCode.NO_BT_SUPPORT;
5
6     if (!btAdapter.isEnabled())
7         return ErrorCode.BT_OFF;
8
9     if (macAddr.equals(""))
10    {
11        final Set<BluetoothDevice> pairedDevices = btAdapter.getBondedDevices();
12        for (BluetoothDevice device : pairedDevices)
13            if (device.getName().equals("drone" + id)) {
14                macAddr = device.getAddress();
15                return null;
16            }
17
18        return ErrorCode.BT_NO_MAC;
19    }
20    return null;
21 }
```

Listing 3.27: Verificer Bluetooth forbindelse

Det ses, at der som det første bliver kontrolleret, om enheden understøtter Bluetooth, hvorefter der verificeres at det er tændt.

Dernæst gennemgås mobiltelefonens liste af paired enheder for at finde dronens mac adresse. I denne liste bliver der ledt efter en enhed, hvis navn passer til det indtastede drone id. Hvis der ikke findes en passende enhed, informeres brugeren om at han er nødt til at paire med dronen inden han kan benytte Bluetooth.

For at skrive og læse via Bluetooth bruges henholdsvis en output og en input stream. For at sende, bliver en *DroneMessage*, lavet til en JSON string, som sendes med output streamen i enkelte bytes. Når der læses, sker dette i den omvendte rækkefølge. Der læses altså enkelte bytes, som laves til en JSON string, der konverteres til et *DroneMessage*-objekt.

3.7.4.5 IspHandler

For at bruge internettet, verificeres som det første, om applikationen har forbindelse til internettet. Til dette bruges en *ConnectivityManager* [53], der har adgang til mobiltelefonens netværkstatus.

```

1 public ErrorCode checkConnection() {
2     ConnectivityManager connMgr =
3         (ConnectivityManager) service.getSystemService(Context.CONNECTIVITY_SERVICE);
4     NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
5     return (networkInfo != null && networkInfo.isConnected()) ? ErrorCode.NO_INTERNET;
6 }
```

Listing 3.28: Verificer internet forbindelse

For at skrive til serveren skal der skrives til følgende URL: "http://shadowbac-x1.gear.host/api/shadowbac". Ud over dette sendes en *identifier* parameter med, som beskriver hvilken drone vi vil, hente data fra. Derudover er det muligt at bruge den yderlige parameter *Command*, til at få serveren til at filtrere beskederne for en bestemt kommando, således at kun beskeder med denne kommando bliver returneret.

Dette bliver brugt i applikationen, når der skal modtages et *START_SIGNAL_ACK*. Da det her er muligt at dronen sender sin batteri status for hurtigt efterfølgende, og *START_SIGNAL_ACK* beskeden ellers ville gå tabt.

I modsætning til Bluetooth er der ikke en konstant forbindelse der holdes åben, når *IspHandler*-klassen benyttes.

For at forbinde til dronen, bliver der derfor sendt en besked, som der forventes et svar på. Der ventes i 30 sekunder på et svar fra dronen. Når der modtages en besked, som indeholder *START_SIGNAL_ACK* kommandoen, verificeres det, at denne blev sendt efter applikationen har sendte *START_SIGNAL*'et til serveren.

For at kunne sammenligne tiden af den sidst sendte besked med tiden der ligger i den modtagende besked, bruges tiden, fra HTTP responsen, når der skrives til serveren.

```

1 public boolean writeMessage(DroneMessage msg) {
2     OutputStreamWriter osw = null;
3
4     try {
5         // Setup HTTP command
6         HttpURLConnection connection = (HttpURLConnection) postUrl.openConnection();
7         connection.setReadTimeout(10000);
8         connection.setConnectTimeout(15000);
9         connection.setDoOutput(true);
10        connection.setRequestMethod("POST");
11        connection.setRequestProperty("Content-Type", "application/json");
12
13        // Preparing JSON object
14        msg.setIdentifier(postId);
15        osw = new OutputStreamWriter(connection.getOutputStream());
16        osw.write(msg.getJSONString());
17        osw.flush();
18
19        // Get response from database
20        int response = connection.getResponseCode();
21        lastWriteTime = connection.getHeaderFieldDate("Date", 0);
22
23        return response == HttpURLConnection.HTTP_OK;
24    } catch (IOException e) {
25        e.printStackTrace();
26    } finally {
```

```

27     try {
28         if (osw != null) {
29             osw.close();
30         }
31     } catch (IOException e) {
32         e.printStackTrace();
33     }
34 }
35 return false;
36 }
```

Listing 3.29: Skriv en besked til serveren via internettet

Det ses, at klassen *HttpURLConnection* [54] bliver brugt til at oprette en forbindelse til serveren. Denne klasse er god til at sende og modtage simple bekseder som dem, der bliver sendt i vores system. For hver skrivning og læsning bliver der oprettet en ny forbindelse. Dette bliver gjort, da en enkelt instanser af klassen ikke er trådsikker. Som beskrevet før ses det at HTTP response headerens felt "Date" bliver gemt i variablen *lastWriteTime*. Denne bruges til tidssammenligningen i *connect()* funktionen.

readMessage() funktionen minder meget om *writeMessage(DroneMessage msg)* funktionen. Request metoden bliver dog ændret til "GET" og der bruges en input stream i stedet.

3.7.4.6 GpsHandler

For at få brugerens lokation, benytter applikationen sig af Google Play service's Location API [16]. *GpsHandler*-klassen implementerer API'ets interfaces

```

1 public class GpsHandler implements
2     GoogleApiClient.ConnectionCallbacks,
3     GoogleApiClient.OnConnectionFailedListener,
4     LocationListener
```

Listing 3.30: GpsHandler interfaces

For at få en lokation, opretter *GpsHandler*-klassen en *LocationRequest* [19], hvori der defineres, i hvilke intervaller og med hvilken nøjagtighed man ønsker at modtage en GPS-lokation.

```

1 public GpsHandler(final Context context) {
2     this.context = context;
3     locationRequest = new LocationRequest();
4     locationRequest.setInterval(5000);
5     locationRequest.setFastestInterval(1000);
6     locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
7
8     googleApiClient = new GoogleApiClient.Builder(context)
9         .addConnectionCallbacks(this)
10        .addOnConnectionFailedListener(this)
11        .addApi(LocationServices.API)
12        .build();
13 }
14
15 public void startListening() {
16     googleApiClient.connect();
17 }
18
19 public void stopListening() {
20     googleApiClient.disconnect();
21 }
```

Listing 3.31: Indstillinger af den ønskede GPS-lokation

PRIORITY_HIGH_ACCURACY er indstillingen, der leverer det mest præcise resultat, dog også det største strømforbrug.

Det ses at to intervaller bliver sat. *setInterval(5000)* definerer, at applikationen ønsker et nyt GPS punkt hvert 5 sekund. *setFastestInterval(1000)*, definerer hvad den mindste tid mellem to GPS koordinater skal være.

Da Android styresystemet sender GPS-koordinaterne, kan det være at de ikke bliver modtaget efter præcis 5 sekunder. Hvis en anden applikation f.eks. vil have et GPS-punkt hvert 4 sekunder, er det muligt, at styresystemet sender lokationen ud til alle applikationer for at spare på strømmen.

Efter applikationen er forbundet til Google Play service, bliver callback funktionen *onConnected(Bundle bundle)* kaldt, hvori det kontrolleres, om mobiltelefonens GPS indstillinger svarer til de ønskede. Hvis ikke, bedes bruger om at ændre dem.

Den vigtigste callback funktion, der skal implementeres, er *onLocationChanged(Location location)*. Denne funktion bliver kaldt, hvis Android styresystemet har sendt et nyt GPS-punkt til applikationen.

```

1 public void onLocationChanged(Location location) {
2
3     if (!location.hasBearing()) {
4         // The two locations are the same
5         if (location.getLatitude() == lastLocation.getLatitude() &&
6             location.getLongitude() == lastLocation.getLongitude()) {
7             location.setBearing(lastLocation.getBearing());
8         } else {
9             location.setBearing(lastLocation.bearingTo(location));
10        }
11    }
12
13    lastLocation = location;
14
15    Intent intent = new Intent(GPS_READY);
16    LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
17 }
```

Listing 3.32: Indlæsning af nye GPS koordinater

Da ikke alle *Location*-objekter, der bliver sendt, indeholder en retning, bliver denne beregnet ud fra det sidste GPS-koordinat, hvis nødvendigt. Hvis der modtages det samme punkt to gange, bliver retningen af det tidligere punkt overtaget i det nye.

3.7.4.7 HeadingSensorHandler

For at beregne brugerens retning i forhold til nord, når han har mobiltelefonen fremme, bruges mobiltelefonens magnetometer og accelerometer. For at registrere en listener på disse sensorer, bruges en *SensorManager* [20].

```

1 public void startListening() {
2     sensorManager.registerListener(sl, sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
3                                     SensorManager.SENSOR_DELAY_UI);
4     sensorManager.registerListener(sl, sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
5                                     SensorManager.SENSOR_DELAY_UI);
6 }
7 public void stopListening() {
8     sensorManager.unregisterListener(sl);
```

```
8 }
```

Listing 3.33: Registrere SensorEventListener

Objektet *sl*, er en *SensorEventListener* [55]. Når der oprettes en instans af denne klasse, skal funktionerne *onSensorChanged(SensorEvent sensorEvent)* og *onAccuracyChanged(Sensor sensor, int i)* overskrives. Den sidste er dog implementeret tomt, da den ikke bruges til noget i vores system.

```
1 public void onSensorChanged(SensorEvent sensorEvent) {
2     // Inspired by:
3     // http://stackoverflow.com/questions/20339942/get-device-angle-by-using-getorientation-function
4     if (sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
5         gravity = sensorEvent.values;
6     if (sensorEvent.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD)
7         geomagnetic = sensorEvent.values;
8     if (gravity != null && geomagnetic != null) {
9         float R[] = new float[9];
10        float I[] = new float[9];
11        boolean success = SensorManager.getRotationMatrix(R, I, gravity, geomagnetic);
12        if (success) {
13            float orientation[] = new float[3];
14            SensorManager.getOrientation(R, orientation);
15            userHeading = (int) (orientation[0] * (180/PI));
16            if(userHeading < 0)
17                userHeading = 360 + userHeading;
18
19            Intent intent = new Intent(HEADING_READY);
20            LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
21        }
22    }
23 }
```

Listing 3.34: Omregning af sensorværdier til mobiltelefonens retning

Det ses at når der blev modtaget data fra begge sensorer, beregnes en bruger-retning og *MainActivity* informeres om at sensor data er klar.

For at beregne brugerens retning, bliver rotationmatrixen *R* fundet ud fra gravitations og geomagnetiske vektorer. *R* bruges i funktionen *getOrientation(R, orientation)*, til at finde mobiltelefonens azimut, pitch og roll.

Applikation har dog kun brug for at kende azimut værdien, som omregnes fra radianer til grader og konverteres fra signed -180°- 180°formatet til et unsigned 0°- 360°.

Når *CommunicatorService* kalder *getHeading()* metoden returneres blot *userHeading* variablen.

References

- [1] ST Microelectronics. *L78S05CV Datasheet*. 1. udg. ST Microelectronics. <http://www.st.com/content/ccc/resource/technical/document/datasheet/e9/be/53/a3/1f/6f/4f/75/CD00000449.pdf/files/CD00000449.pdf/jcr:content/translations/en.CD00000449.pdf>, mar. 2014.
- [2] AeroQuad. *AeroQuad Manual*. Sep. 2016. URL: <http://aeroquad.com/showwiki.php?title=AeroQuad-Manual>.
- [3] Banggood. *Mini DC-DC Converter Step Down Module Adjustable Power Supply*. Nov. 2016. URL: <http://www.banggood.com/Mini-DC-DC-Converter-Step-Down-Module-Adjustable-Power-Supply-p-920327.html?rmmds=search>.
- [4] HobbyKing.com. *Turnigy Accucel-8 150W 7A Balancer/Charger*. Sep. 2016. URL: http://www.hobbyking.com/hobbyking/store/_7523_Turnigy_Accucel_8_150W_7A_Balancer_Charger.html.
- [5] Brian Schneider. *A Guide to Understanding LiPo Batteries*. Sep. 2016. URL: <http://rogershobbycenter.com/lipoguide/>.
- [6] Craig Larman. *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development*. third. Chapter 39 regarding N+1 View Model. Pearson Education (Us), okt. 2004.
- [7] Android. *Android Activity Overview*. Okt. 2016. URL: <https://developer.android.com/guide/components/activities.html>.
- [8] Android. *Android Service Overview*. Okt. 2016. URL: <https://developer.android.com/guide/components/services.html>.
- [9] Android. *Android OnClickListener*. Dec. 2016. URL: <https://developer.android.com/reference/android/view/View.OnClickListener.html>.
- [10] Android. *Android AsyncTask*. Dec. 2016. URL: <https://developer.android.com/reference/android/os/AsyncTask.html>.
- [11] Android. *Android startActivityForResult*. Dec. 2016. URL: [https://developer.android.com/reference/android/app/Activity.html#startActivityForResult\(android.content.Intent,%20int\)](https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent,%20int)).
- [12] Android. *Android SharedPreferences*. Dec. 2016. URL: <https://developer.android.com/reference/android/content/SharedPreferences.html>.
- [13] Android. *Android LocalBroadcastManager*. Dec. 2016. URL: <https://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager.html>.
- [14] Android. *Android OnSeekBarChangeListener*. Dec. 2016. URL: <https://developer.android.com/reference/android/widget/SeekBar.OnSeekBarChangeListener.html>.
- [15] Android. *Android SeekBar*. Dec. 2016. URL: <https://developer.android.com/reference/android/widget/SeekBar.html>.

- [16] Google. *Google Play service location API*. Dec. 2016. URL: <https://developers.google.com/android/reference/com/google/android/gms/location/package-summary>.
- [17] Android. *Android ProgressBar*. Dec. 2016. URL: <https://developer.android.com/reference/android/widget/ProgressBar.html>.
- [18] Google. *GoogleApiClient*. Dec. 2016. URL: <https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient>.
- [19] Google. *LocationRequest*. Dec. 2016. URL: <https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest>.
- [20] Android. *Android SensorManager*. Dec. 2016. URL: <https://developer.android.com/reference/android/hardware/SensorManager.html>.
- [21] Oracle. *Java synchronized*. Dec. 2016. URL: <https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html>.
- [22] Android. *Android ConditionVariable*. Dec. 2016. URL: <https://developer.android.com/reference/android/os/ConditionVariable.html>.
- [23] Microsoft. *Download SQL Server Management Studio (SSMS)*. Sep. 2016. URL: <https://msdn.microsoft.com/en-us/library/mt238290.aspx>.
- [24] Margaret Rouse. *TCP (Transmission Control Protocol)*. Aug. 2014. URL: <http://searchnetworking.techtarget.com/definition/TCP>.
- [25] Wikipedia. *Hypertext Transfer Protocol*. Okt. 2016. URL: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.
- [26] Wikiepedia. *Bluetooth low energy*. Nov. 2016. URL: https://en.wikipedia.org/wiki/Bluetooth_low_energy.
- [27] Official U.S. Government information about the Global Positioning System (GPS) og related topics. *Space Segment*. Okt. 2016. URL: <http://www.gps.gov/systems/gps/space/>.
- [28] NETPLAN. *GPS - hvordan virker det?* Okt. 2016. URL: <http://www.netplan.dk/index.php/nyt/artikler/10-sectionnetplan/catnyheder/28-gps-hvordan-virker-det>.
- [29] Wikipedia. *HTTPs*. Okt. 2016. URL: <https://en.wikipedia.org/wiki/HTTPS>.
- [30] Microsoft. *Visual Studio 2015*. Sep. 2016. URL: <https://www.microsoft.com/en-us/download/details.aspx?id=48146>.
- [31] Microsoft. *.NET Downloads*. Sep. 2016. URL: <https://www.microsoft.com/net/download>.
- [32] Postman. *Postman*. Sep. 2016. URL: <https://www.getpostman.com/>.
- [33] GearHost. *GearHost*. Sep. 2016. URL: <https://www.gearhost.com/>.
- [34] Arduino. *Previous IDE Releases*. Sep. 2016. URL: <https://www.arduino.cc/en/Main/OldSoftwareReleases>.
- [35] AeroQuad. *AeroQuad Software*. Jul. 2015. URL: <http://aeroquad.com/showwiki.php?title=AeroQuad-Software>.

- [36] Unknown. *Fix for error 129*. Sep. 2016. URL: <http://www.madwizard.org/download/electronics/msys-1.0-vista64.zip>.
- [37] *Win32 Disk Imager*. Apr. 2016. URL: <https://sourceforge.net/projects/win32diskimager/>.
- [38] Boost C++ libraries. doc/html/boost_asio/example/http/client/sync_client.cpp. Nov. 2011. URL: http://www.boost.org/doc/libs/1_47_0/doc/html/boost_asio/example/http/client/sync_client.cpp.
- [39] Albert Huang. *An Introduction to Bluetooth Programming - RFCOMM sockets*. Nov. 2008. URL: <https://people.csail.mit.edu/albert/bluez-intro/x502.html>.
- [40] WiringPi. *WiringPi*. Nov. 2016. URL: <http://wiringpi.com/>.
- [41] Doxygen. *Doxygen*. Sep. 2016. URL: <http://www.stack.nl/~dimitri/doxygen/>.
- [42] Haishi. *Bearing between two points on the Earth – the code and the math*. Sep. 2009. URL: <http://haishibai.blogspot.dk/2009/09/bearing-between-two-points-on-earth.html>.
- [43] RosettaCode. *Haversine formula*. Nov. 2016. URL: https://rosettacode.org/wiki/Haversine_formula#C.
- [44] Whuber. *Calculating Latitude/Longitude X miles from point?* Jul. 2012. URL: <http://gis.stackexchange.com/questions/5821/calculating-latitude-longitude-x-miles-from-point>.
- [45] Aeroquad. *Serial Commands: AeroQuad Ground Station Interface Definitions*. Feb. 2015. URL: <http://aeroquad.com/showwiki.php?title=Serial-Commands:-AeroQuad-Ground-Station-Interface-Definitions>.
- [46] HDD Software. *Free Serial Analyzer*. Dec. 2016. URL: <https://freeserialanalyzer.com/features>.
- [47] Mikro. *3 Axis Magnetometer*. Jun. 2009. URL: <http://aeroquad.com/showthread.php?88-3-Axis-Magnetometer>.
- [48] Arduino. *Servo library*. Nov. 2016. URL: <https://www.arduino.cc/en/Reference/Servo>.
- [49] NMEA. *National Marine Electronics Association*. Nov. 2016. URL: <http://www.nmea.org/>.
- [50] Wikipedia. *Decimal degrees*. Nov. 2016. URL: https://en.wikipedia.org/wiki/Decimal_degrees.
- [51] Oracle. *JavaDoc*. Dec. 2016. URL: <http://www.oracle.com/technetwork/articles/java/index-137868.html>.
- [52] Android. *Android BluetoothSocket*. Dec. 2016. URL: <https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html>.
- [53] Android. *Android ConnectivityManager*. Dec. 2016. URL: <https://developer.android.com/reference/android/net/ConnectivityManager.html>.
- [54] Android. *Android HttpURLConnection*. Dec. 2016. URL: <https://developer.android.com/reference/java/net/HttpURLConnection.html>.

- [55] Android. *Android SensorEventListener*. Dec. 2016. URL: <https://developer.android.com/reference/android/hardware/SensorEventListener.html>.