

Systemarkitektur og Design

Autonom overvågningsdrone





**Ingeniørhøjskolen
Aarhus Universitet
Finlandsgade 22
8200 Aarhus N**

Projekt titel:

Autonom overvågningsdrone

Projekt:

Bachelorprojekt

Projektperiode:

August 2014 - December 2014

Projektgruppe:

14123

Gruppemedlemmer:

11647 – Rasmus T. Lydiksen
11762 – Anders H. Opstrup
11655 – Kevin Grooters

Vejleder:

Torben Gregersen

Samlet sidetal: 89

Projekt afsluttet: 17-12-2014

Indholdsfortegnelse

Kapitel 1 Intro	1
1.1 Revisionshistorik	1
1.2 Ordforklaring	2
1.3 Indledning	2
1.4 Domain Model	3
Kapitel 2 Hardware	5
2.1 Block definition diagram	5
2.1.1 Udvidet - Block definition diagram	6
2.1.2 Blokbeskrivelse	7
2.2 Internal block diagrammer	8
2.2.1 Overordnet system	8
2.2.2 Drone	9
2.2.3 Flight control board	11
2.2.4 Remote controller	12
2.2.5 Afstandssensorer	13
2.2.6 Motor styring	14
2.3 Hardware design	16
2.3.1 Hardware oversigt	16
Kapitel 3 Software	18
3.1 N+1 view model	18
3.1.1 View beskrivelse	19
3.2 Logical view	20
3.2.1 Iteration #1	20
3.2.2 Iteration #2	30
3.2.3 Iteration #3	44
3.2.4 Iteration #4	52
3.3 Process view	58
3.3.1 Kommunikations timing	59
3.4 Data view	60
3.4.1 Database design	60
3.4.2 Detaljeret database beskrivelse	63
3.5 Deployment view	68
3.5.1 HTTP protokol	69
3.6 Implementation view	70
3.6.1 Opsætning af udviklings IDE til Arduino	70
3.6.2 Opsætning af timere til PWM	76
3.6.3 Opsætning af server	81
3.6.4 Opsætning webapplikation	87

Intro 1

1.1 Revisionshistorik

Rev. Nr	Dato	Initialer	Ændring
1.0	11-09-2014	AO, KG, RL	Oprettet dokument, og lavet udkast til hardware diagrammer.
1.1	16-09-2014	KG, RL	Opdateret block definition og internal block diagrammer.
1.2	22-09-2014	KG	Tilføjet signal beskrivelse til alle hardware diagrammer.
2.1	29-09-2014	AO, KG, RL	Oprettet software afsnit. Påbegyndt logical view ved at tilføje designoverview og pakkediagrammer.
2.2	10-10-2014	AO, KG, RL	Tilføjet sekvens- og klassediagrammer tilhørende første iteration.
2.3	31-10-2014	AO, KG, RL	Tilføjet sekvens- og klassediagrammer tilhørende anden iteration.
2.4	11-11-2014	AO, KG, RL	Tilføjet sekvens- og klassediagrammer tilhørende tredje og fjerde iteration.
2.5	13-11-2014	RL	Tilføjet state machines til alle iterationer.
2.6	14-11-2014	AO	Tilføjet data view.
2.7	17-11-2014	AO, RL	Tilføjet proces view.
2.8	25-11-2014	AO, KG, RL	Tilføjet deployment view.
2.9	28-11-2014	AO, KG, RL	Tilføjet implementation view.

Tabel 1.1: Revisionshistorik

1.2 Ordforklaring

Forkortelse	Betydning	Forklaring
Drone	Drone	Aeroquad ARF quadrocopter og påmonteret hardware.
Webapp	Webapplikation	Applikation som bruges til opsætning af nye flyvninger og monitorering af tidlige flyvninger.
Server	Server	Server dækker over systemets database og tilhørende logik.
Webserver	Server og Webapplikation	Webserver dækker både over webapplikationen og serveren
Main controller	Main controller	Main controller er en arduino mega2560. Main controlleren styrer dronen ud fra flyveopsætning og data fra diverse sensorer.
Flight control board	Flight control board	Boardet er købt sammen med quadrocopter hos aeroquad. Boardet bruges som mellemled mellem main controller og dronens motorer.
3G/GPS	3G/GPS modul	Dette modul er et 3G/GPS shield der påmonteres main controller
bdd	Block definition diagram	bdd bruges til at vise systemets overordnede hardware blokke
ibd	Internal block diagram	ibd bruges til at vise ekstern og intern kommunikation tilhørende en eller flere blokke.
sd	Sekvens diagram	Diagrammet bruges til at vise afvikling og timing af forskellige processer.
stm	State machine diagram	stm viser states i en given proces.

Tabel 1.2: Ordforklaring

1.3 Indledning

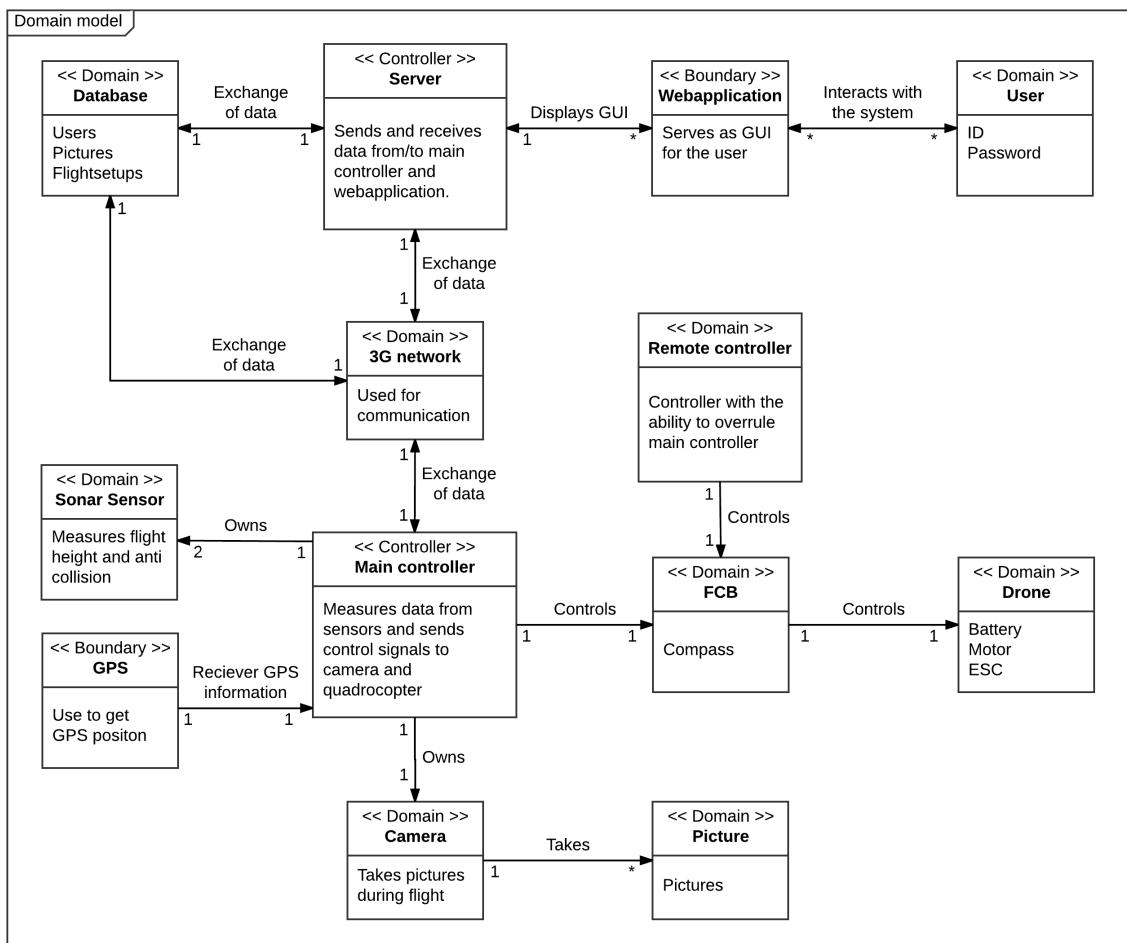
I dette kapitel beskrives systemarkitektur og design, der er udarbejdet på baggrund af kravspecifikationen. Målet med kapitlet er at beskrive og designe de blokke systemet består af. Dels gives der indsigt i hvordan blokkene kommunikerer med hinanden og hvordan den interne og eksterne kommunikation virker. Desuden vises hvordan systemets blokke er designet.

1.4 Domain Model

Domain modellen¹ bruges som en overgang mellem kravspecifikation og systemarkitektur. I kravspecifikation beskrives hvad der sker ved interaktion med systemet. Mens systemarkitekturen bruges til at beskrive systemet i blokke og til at skitsere både interne og eksterne forbindelser. Domain modellen bruges til at beskrive hele systemets domæne. Der kigges ikke på hardware vs. software, der kigges i stedet på "enheder" og deres ansvarsområder.

På figur 1.1 vises domain model tilhørende systemet. De fire øverste enheder i domain modellen dækker ansvarsområder som har med systemets webapplikation at gøre. De resterende enheder er alle tilknyttet ansvarsområder der omhandler dronen.

På domain modellen vises det, at bruger tilgår systemet via en webapplikation. Webapplikationen har forbindelse til en server, som yderligere har forbindelse til dronens main controller. Det vises desuden at kommunikation mellem server og main controller går gennem det mobile 3G netværk. Under flyvning styrer dronens main controller både kamera, GPS, afstandssensorer og dronens motorer.



Figur 1.1: Domain model

¹http://en.wikipedia.org/wiki/Domain_model

Af figur 1.1 ses det at domain modellen er opbygget af tre forskellige typer klasser. Nedenfor beskrives de tre forskellige klasser:

Domain klasser

Domain klasser repræsenterer systemets domæne. Domain klasser er passive elementer, der er ansvarlige for væsentlige delparte af systemets funktionalitet.

Boundary klasser

Boundary klasser repræsenterer use case-aktører og aktørernes grænseflader til systemet. Boundary klasser er elementer der ligger i den yderste periferi af systemet. I nogle tilfælde er boundary klasser "front-end"elementer, der er designet til at udsende output samt at tage imod input fra systemets bruger. I andre tilfælde er boundary klasser "back-end"elementer der bruges til at hjælpe/understøtte controller klasser.

Controller klasser

Controller klasser indeholder systemets domænelogik og står for at kontrollere systemets boundary og domain klasser. Desuden er controller klasserne ansvarlige for at håndtere input og output fra systemets boundary klasser.

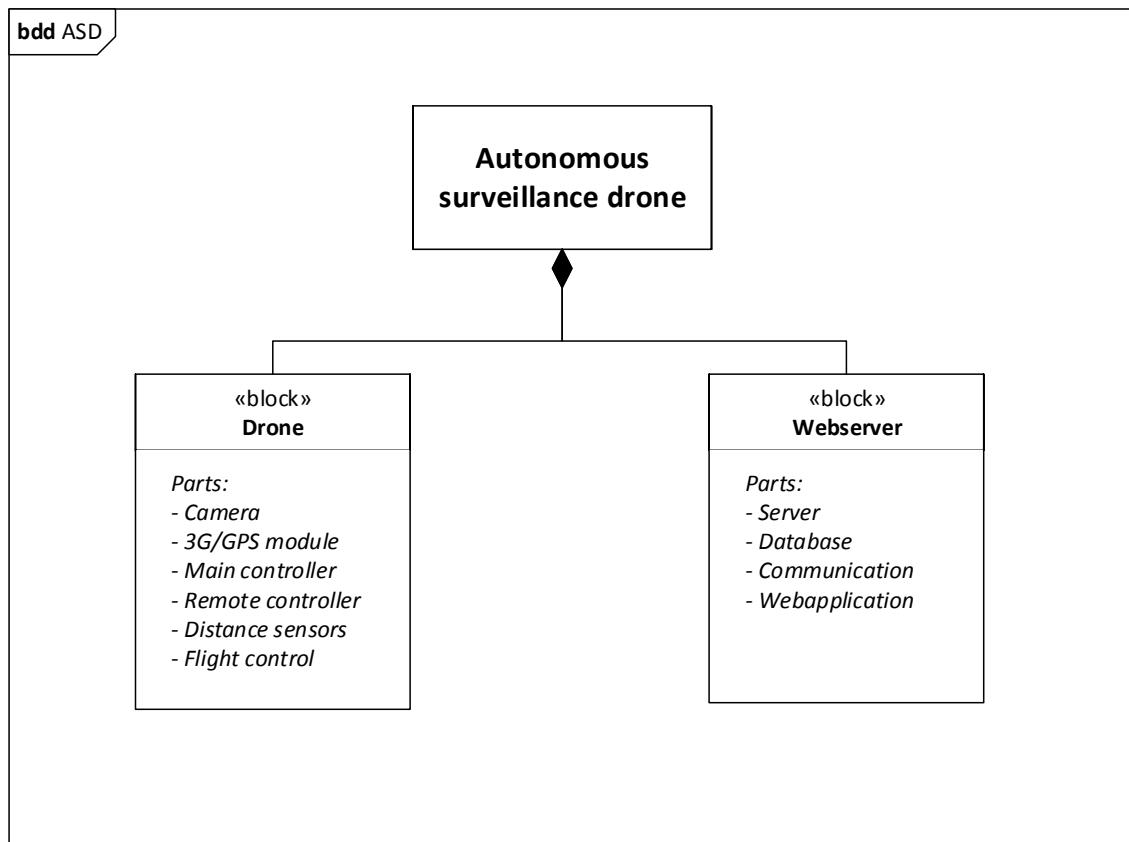
Domain modellen bruges både til beskrivelsen af systemets hardware og software arkitektur. Men der gøres primært brug domain modellen i software arkitektur afsnittet. I software arkitekturen bruges domain modellen bla. til at danne grundlag for softwareklasser, klassenavne og klassernes indbyrdes forhold.

Hardware 2

I det følgende afsnit beskrives systemets hardware vha. SysML diagrammer. Indledningsvis bruges block definition diagrammer til at identificere og beskrive systemets blokke. Senere i afsnittet åbnes udvalgte blokke og de interne og eksterne forbindelser vises med internal block diagrammer. I det følgende vil det block definition diagrammer blive kaldt bdd'er, og internal block diagrammer benævnes ibd'er.

2.1 Block definition diagram

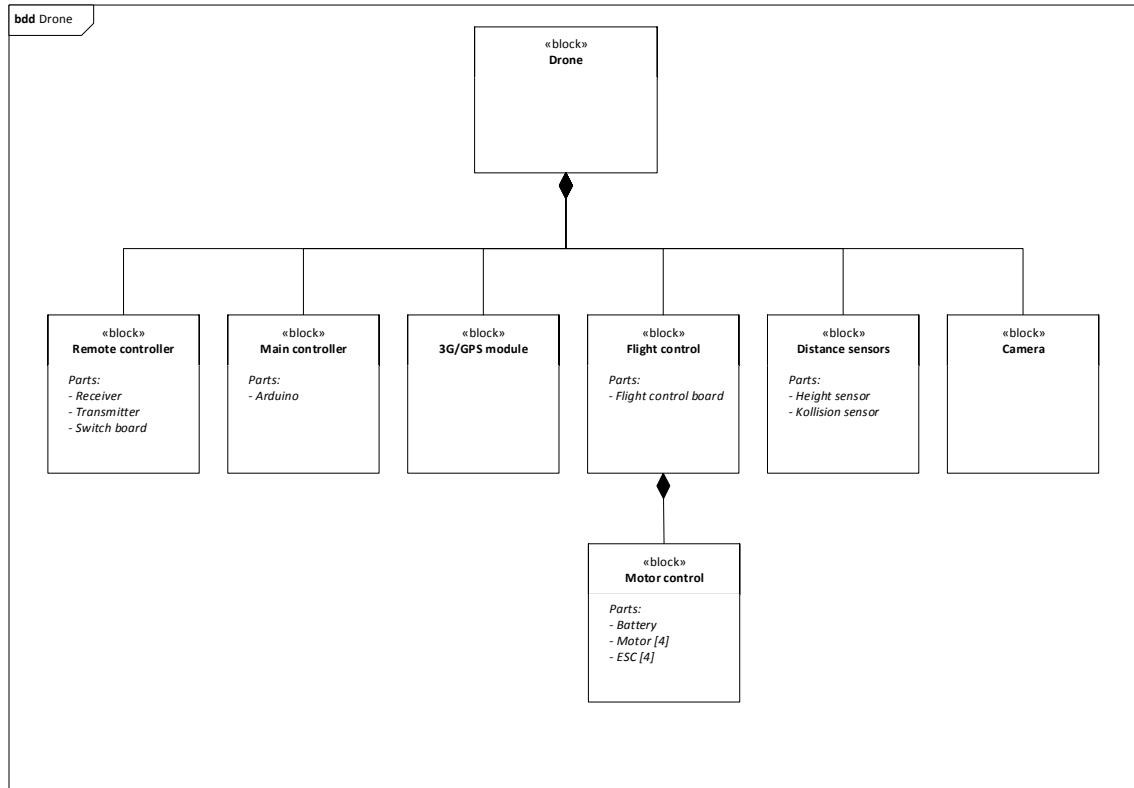
I det overordnede bdd nedenfor vises de blokke systemet består af, samt hvilke parts blokkene har. Helt overordnet set består systemet af blokkene: *Webserver* og *Drone*.



Figur 2.1: bdd - overordnet

2.1.1 Udvidet - Block definition diagram

Da drone blokken er kompliceret og indeholder mange parts kræves en yderlige beskrivelse. På figur 2.2 vises et bdd, der går mere i dybden med drone blokken. På figuren åbnes drone blokken og det vises hvilke blokke dronen er bygget af.



Figur 2.2: bdd - drone

Websværven beskrives ikke nærmere i dette afsnit, da den ikke indeholder så mange parts og fordi websværvens parts har en anden og mere løs kobling end dronens.

2.1.2 Blokbeskrivelse

Main controller

Main controlleren fungerer som dronens hjerne. Ud fra kommunikation med webserver og input fra de forskellige sensorer styrer main controlleren dronens motorer. Skal dronen fx. flyve højere udsendes styrings signaler, som sørger for motorerne øger deres rotationshastighed. Under autonom flyvning bruges PWM signaler udsendt fra main controller til styre dronen.

Remote controller

Denne blok består af receiver, transmitter og switch board. Blokken gør det muligt at switche mellem autonom og manuel styring. Afhængig af hvordan transmitteren (fjernbetjening) er indstillet benyttes henholdsvis manuel eller autonom flyvning.

Flight control

Flight control indeholder flight control boardet. Flight control boardet indeholder alt software til styring, herunder PID regulering. Boardet har indbygget kompas, som bruges under flyvning. Flight control blokkens vigtigste opgave er at videreförige control signaler fra main controller til ESC'er som åbner/lukker for forsyning til dronens motorer.

3G/GPS module

3G/GPS blokken har to funktioner i systemet. For det første er 3G/GPS blokken ansvarlig for opdatering af dronens GPS position. Desuden fungerer blokken som kommunikationslag mellem webserver og main controller. Al information der udvekles mellem webserver og drone går gennem 3G/GPS modulet.

Distance sensorer

Afstands sensorerne bruges både til måling af flyvehøjde og til anti kollision. Sensorerne aktiveres af main controlleren, når højdemåling eller tjek af forhindring foretages. Sensorerne bruger 40 kHz signaler til at måle distancen til jorden eller eventuelle forhindringer.

Camera

Når dronen er i rette position modtager kameraet besked og tager et billede. Billedet tages og sendes videre i systemet til godkendelse. Der tages et nyt billede hvis billedet ikke godkendes.

Motor control

Denne blok består af ESC, batteri og motorer. Blokken styres med PWM signaler, som udsendes fra flight control boardet.

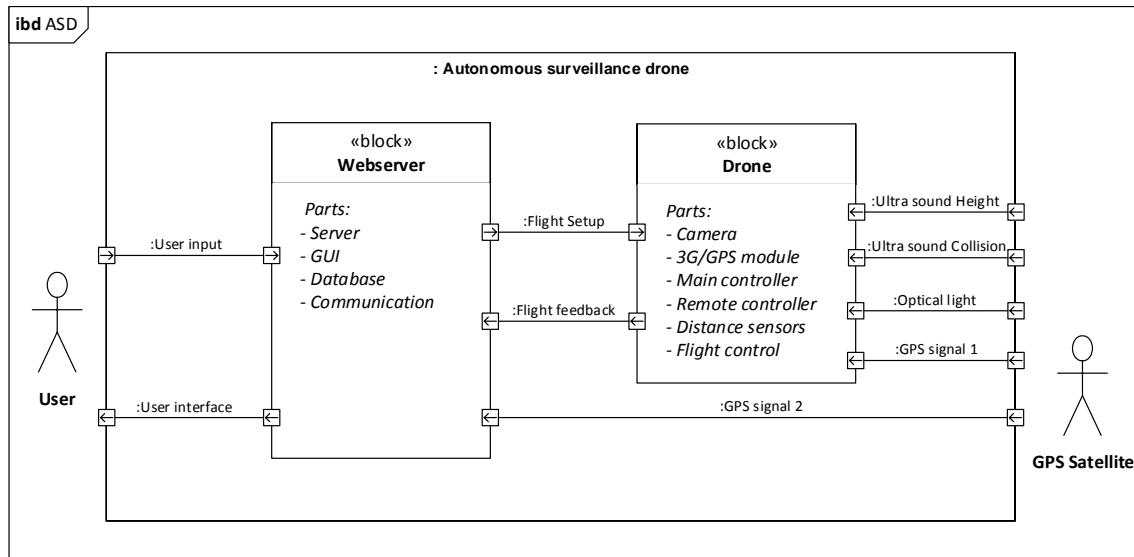
Webserver

Denne blok indeholder server, webapplikation, database og kommunikation. Ved interaktion med webapplikation er det muligt for bruger at tilgå server. Sammenspil mellem webapplikation og server gør det muligt for bruger at lave en ny flyveopsætning eller undersøge en tidligere flyvning. Server kommunikerer med dronen og gemmer løbende vigtig information i databasen.

2.2 Internal block diagrammer

2.2.1 Overordnet system

På figur 2.3 vises et overordnet ibd diagram. Diagrammet beskriver hvordan systemets to største blokke kommunikerer. Det beskrives hvilke signaler blokkene sender til hinanden og hvordan blokkene påvirkes af systemets aktører.



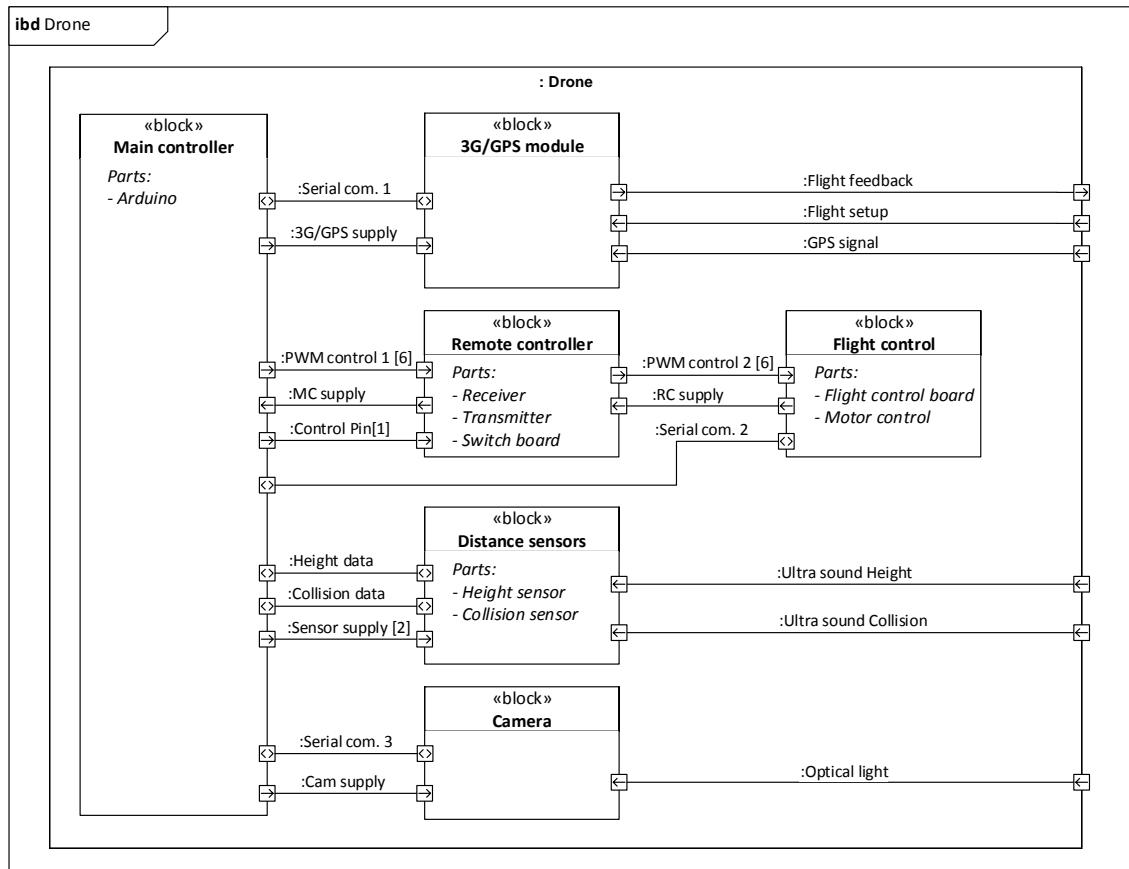
Figur 2.3: ibd - ASD

Signal navn	Signal beskrivelse	Out	In
User input	Via <i>webserver</i> opsætter bruger ny flyvning eller undersøger en tidligere.	Bruger.	Webserver.
User interface	Via GUI kan bruger se hvad der sker i <i>webserver</i> .	Webserver.	Bruger.
Flight setup	Fra <i>webserver</i> sendes flyveopsætningen til dronen.	Webserver.	Dronen.
Flight feedback	Informationer fra dronen til <i>webserver</i> .	Dronen.	Webserver.
GPS signal 1	GPS signal.	GPS-satellit.	Dronen.
GPS signal 2	GPS signal.	GPS-satellit.	Webserver.
Optical light	Lys til at tage billeder.	Drone omgivelser.	Drone.
Ultra sound Height	Lydsignaler reflekteres tilbage til modtagerdel af ultralyds sensor.	Drone omgivelser.	Drone.
Ultra sound Collision	Lydsignaler reflekteres tilbage til modtagerdel af ultralyds sensor.	Drone omgivelser.	Drone.

Tabel 2.1: Forbindelser tilhørende: Ibd - ASD

2.2.2 Drone

På figur 2.4 vises et ibd over drone. Diagrammet viser bla. hvordan alle blokke i dronen er forbundet til main controlleren. Via 3G/GPS modulet sendes og modtages information til og fra drone. Modtaget information bruges til at fortælle dronen hvor der skal flyves til, hvor der skal tages billeder og om de billeder der tages under flyvning godkendes.



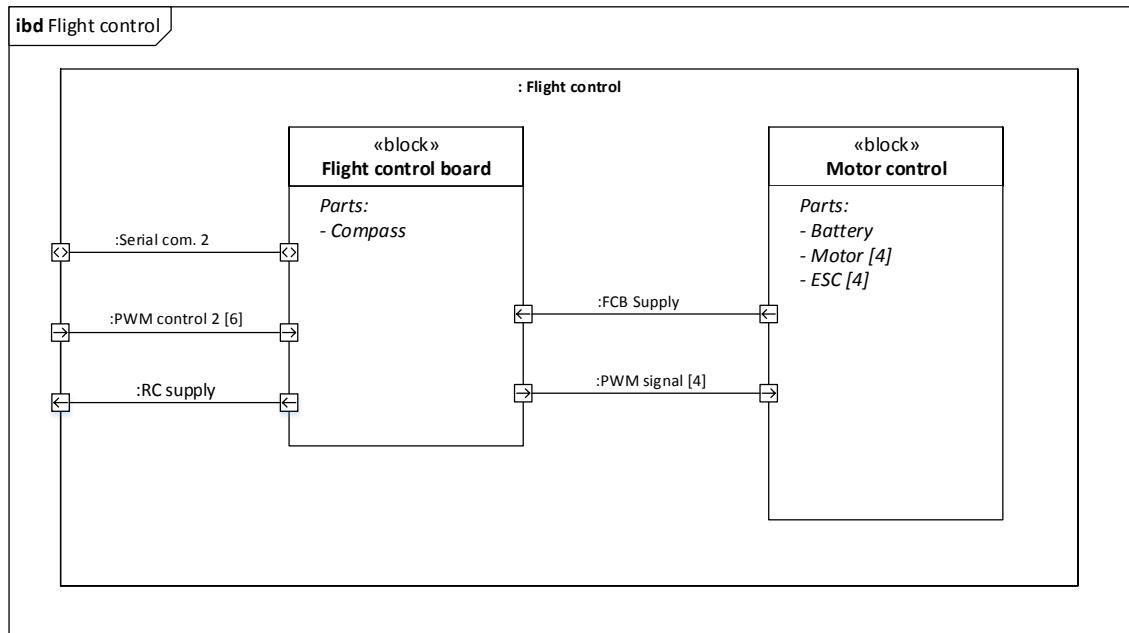
Figur 2.4: ibd - drone

Signal navn	Signal beskrivelse	Out	In
Flight feedback	Handshake og billeder	3G/GPS module.	Webserver.
Flight setup	Handshake & Flyveopsætning	Webserver.	3G/GPS module.
GPS signal	GPS signal	GPS-satellitter	3G/GPS module.
3G/GPS supply	5V DC	Arduino	3G/GPS module.
Serial com. 1	RX / TX signaler. Styrer 3G modul.	Main controller.	3G/GPS module.
PWM control 1 [6]	244 Hz 6 kanals signal. Duty cycle 25-50%. Periode på 4ms	Main controller.	Remoto controller.
MC supply	5V DC	Flight control.	Main controller
Control pin [1]	Skifter mellem 5V og 0V	Main controller	Remoto controller.
PWM control 2 [6]	50 Hz 6 kanals signal. Duty cycle 5-10%. Periode på 20ms	Remote controller.	Flight control.
RC supply	5V DC	Flight control	Remote controller.
Serial com. 2	RX / TX signaler. Aflæsning kompas.	Main controller.	Flight control.
Height data.	Trigger & echo signal der indikerer afstanden.	Main controller.	Distance sensors.
Collision data.	Trigger & echo signal der indikerer afstanden.	Main controller.	Distance sensors.
Sensor supply [2]	5V DC.	Main controller.	Distance sensors.
Ultra sound Height	Afstand vha. ultralyd. (8 bursts af 40kHz)	Distance sensor.	Omgivelserne.
Ultra sound Collision	Afstand vha. ultralyd. (8 bursts af 40kHz)	Distance sensor.	Omgivelserne.
Serial com. 3	RX / TX signal. Styrer kamera.	Main controller.	Camera.
Cam supply	5V DC	Main controller.	Camera.
Optical light	Kamera billede.	Camera.	Omgivelserne.

Tabel 2.2: Forbindelser til: **Ibd** - drone

2.2.3 Flight control board

På figur 2.5 vises ibd til Flight control. Flight control består af Flight control board og motor control. Flight control boardet kommunikerer med main controlleren via seriel kommunikation og 6 PWM signaler. Kompas information sendes via seriel kommunikation, og PWM signalerne bestemmer hvordan motorerne skal rotere.



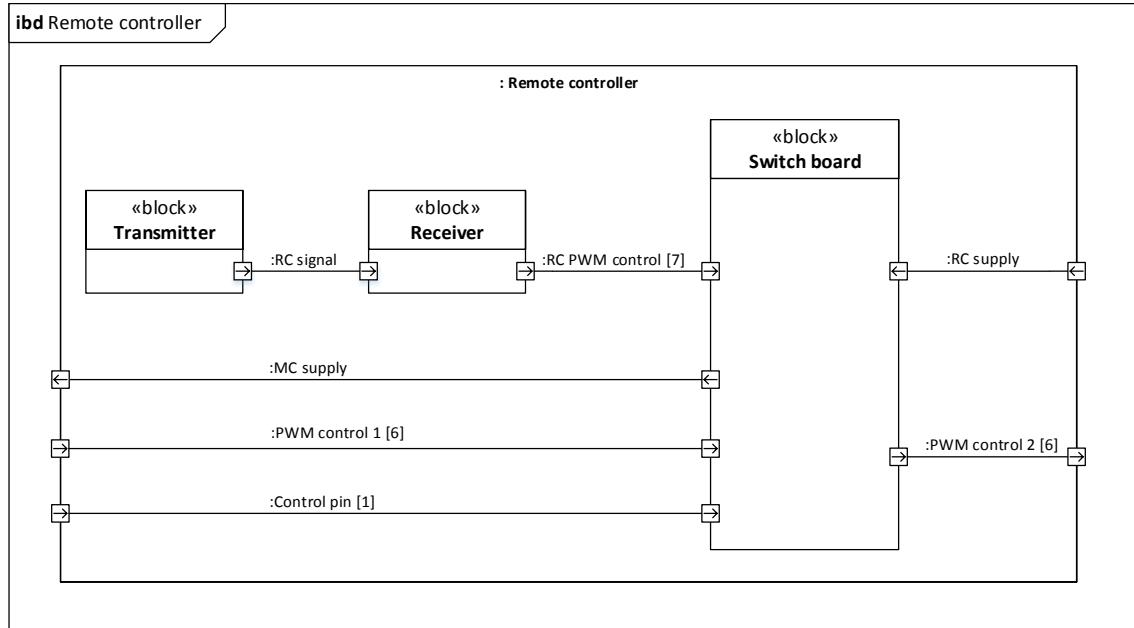
Figur 2.5: ibd - flight control board

Signal navn	Signal beskrivelse	Out	In
Serial com. 2	RX / TX. signal.	Arduino.	Flight control board.
PWM control 2 [6]	6 PWM signaler. Enten 244 Hz eller 50 Hz	Arduino.	Flight control board.
FCB supply	Forsyning til flight control boardet. 5V DC	Motor control.	Flight control board.
PWM signal [4]	400 Hz 4 kanals signal. Duty cycle mellem 40-80 %.	Flight control board.	Motor control.
RC supply	5V forsyning.	Flight control board.	Remote controller.

Tabel 2.3: Forbindelser til: Ibd - flight control board.

2.2.4 Remote controller

På figur 2.6 vises ibd til Remote controller, der består af receiver, transmitter og switch board. Blokken gør det muligt at switche mellem autonom og manuel styring. Afhængig af det signal receiveren modtager fra transmitteren benyttes henholdsvis manuel eller autonom flyvning. Når der flyves autonom, benyttes PWM signaler fra main controller til styring, og under manuel flyvning benyttes signaler fra receiveren.



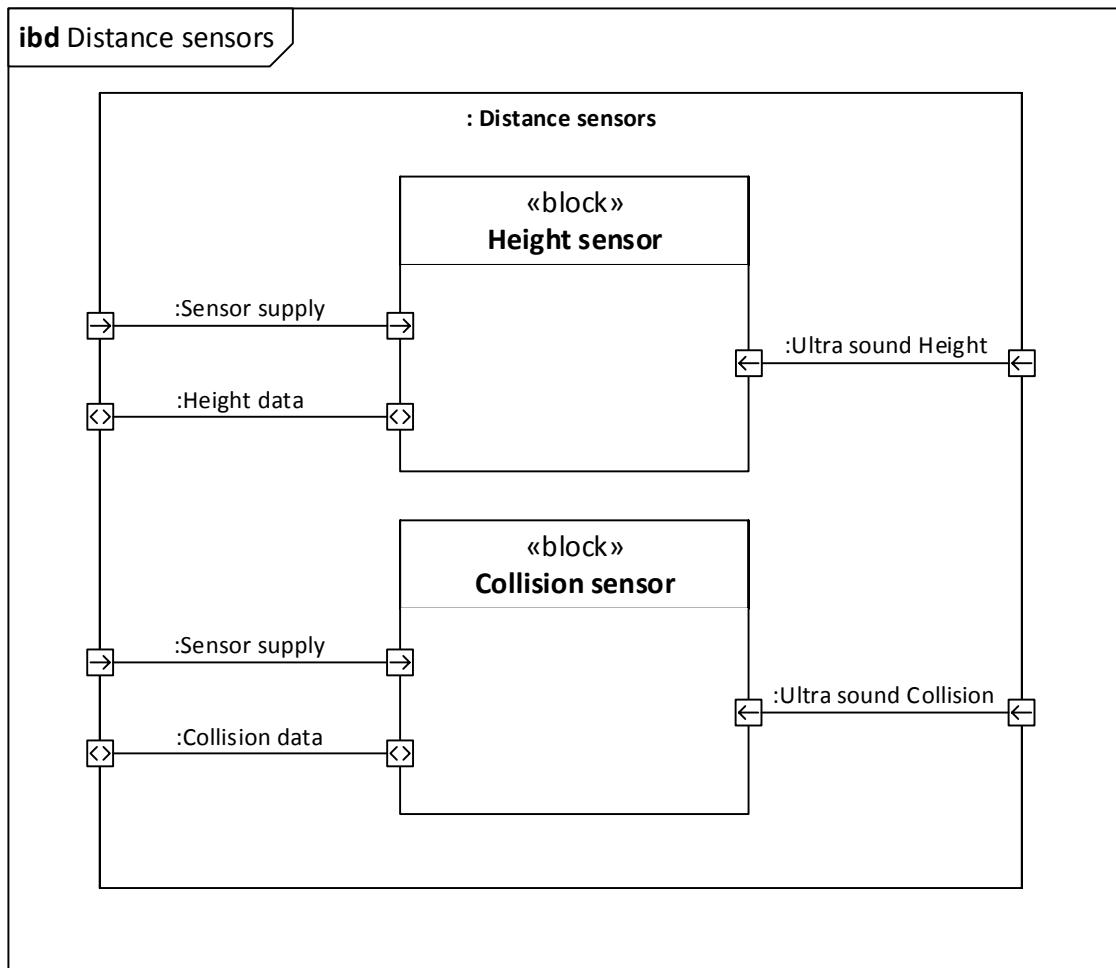
Figur 2.6: Ibd - Remote controller

Signal navn	Signal beskrivelse	Out	In
RC supply	5V DC til switch board.	Flight control.	Switch board.
PWM control 2 [6]	6 PWM signaler. Enten 244 Hz eller 50 Hz	Switch board.	Flight control.
RC PWM control [7]	6 PWM signaler på 50 Hz + GND	Reciever.	Switch board.
RC signal	2,4 GHz trådløs kommunikation	Reciever.	Switch board.
MC supply	5V DC.	Switch board.	Main controller.
PWM control 1 [6]	6 PWM signaler på 244 Hz.	Main controller.	Switch board.
Control pin [1]	Skifter mellem 5V og 0V	Main controller.	Remoto controller.

Tabel 2.4: Forbindelser til: Ibd - remote controller.

2.2.5 Afstandssensorer

På figur 2.7 vises et ibd over afstandssensor blokken. Blokken består af to forskellige sensorer, en sensor til højdemåling og en til antikollision. Begge sensorer fungerer uafhængig af hinanden, de aktiveres enkeltvis når main controller sender trigger signal til dem.



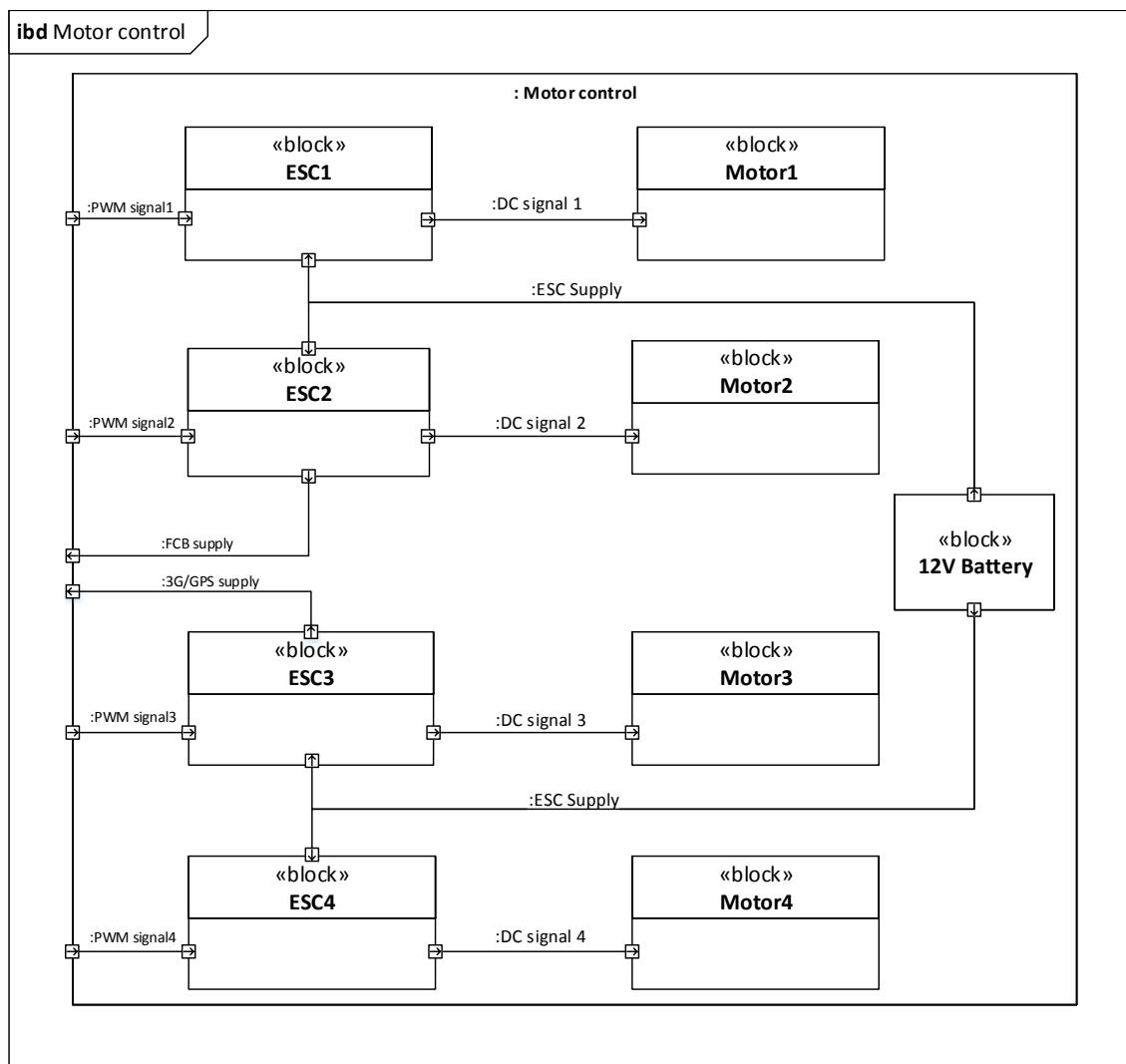
Figur 2.7: ibd - distance sensor

Signal navn	Signal beskrivelse	Out	In
Sensor supply	5V DC.	Arduino.	Højde sensor.
Height data	RX / TX signal.	Arduino.	Højde sensor.
Ultra sound Height	Måling af flyvehøjde	Omgivelserne	Højde sensor.
Sensor supply	5V DC.	Arduino.	Kollision sensor.
Collision data	RX /TX signal	Arduino.	Kollision sensor.
Ultra sound Collision	Detektion af objekt	Omgivelserne	Kollision sensor.

Tabel 2.5: Forbindelser til: Ibd Distance sensor

2.2.6 Motor styring

På figur 2.8 vises det interne blok diagram for motor control blokken. Blokken består af 4 ESC'er, 4 motorer og et 12V batteri. 12V batteriet står for forsyning til hele blokkken, mens de PWM signaler der sendes ind i ESC'erne bestemmer hvor meget forsyning motorerne tildeles. Den PWM der sendes til ESC'er sendes med en duty cycle mellem 40-80%. Ved ændring af duty cyclen, ændres forsyningen til dronens motorer. Ved en ændring af forsyning ændres hastigheden hvorved motorerne roterer.



Figur 2.8: Ibd - Motor control

Signal navn	Signal beskrivelse	Out	In
PWM signal1	400 Hz signal med en duty cycle på 40-80 %	Flight control.	ESC1.
PWM signal2	400 Hz signal med en duty cycle på 40-80 %	Flight control.	ESC2.
PWM signal3	400 Hz signal med en duty cycle på 40-80 %	Flight control.	ESC3.
PWM signal4	400 Hz signal med en duty cycle på 40-80 %	Flight control.	ESC4.
DC signal 1	Et varierende DC.	ESC1.	Motor1.
DC signal 2	Et varierende DC.	ESC2.	Motor2.
DC signal 3	Et varierende DC.	ESC3.	Motor3.
DC signal 4	Et varierende DC.	ESC4.	Motor4.
FCB supply	5V DC.	ESC2	Flight control.
3G/GPS supply	5V DC.	ESC3.	3G/GPS module.
ESC supply	12V DC.	Battery.	ESC1/ESC2.
ESC supply	12V DC.	Battery.	ESC3/ESC4.

Tabel 2.6: Forbindelser til: **Ibd** - Motor control

2.3 Hardware design

Til projektet er der købt mange færdige hardware moduler. Switch boardet er det eneste hardware der er blevet designet.

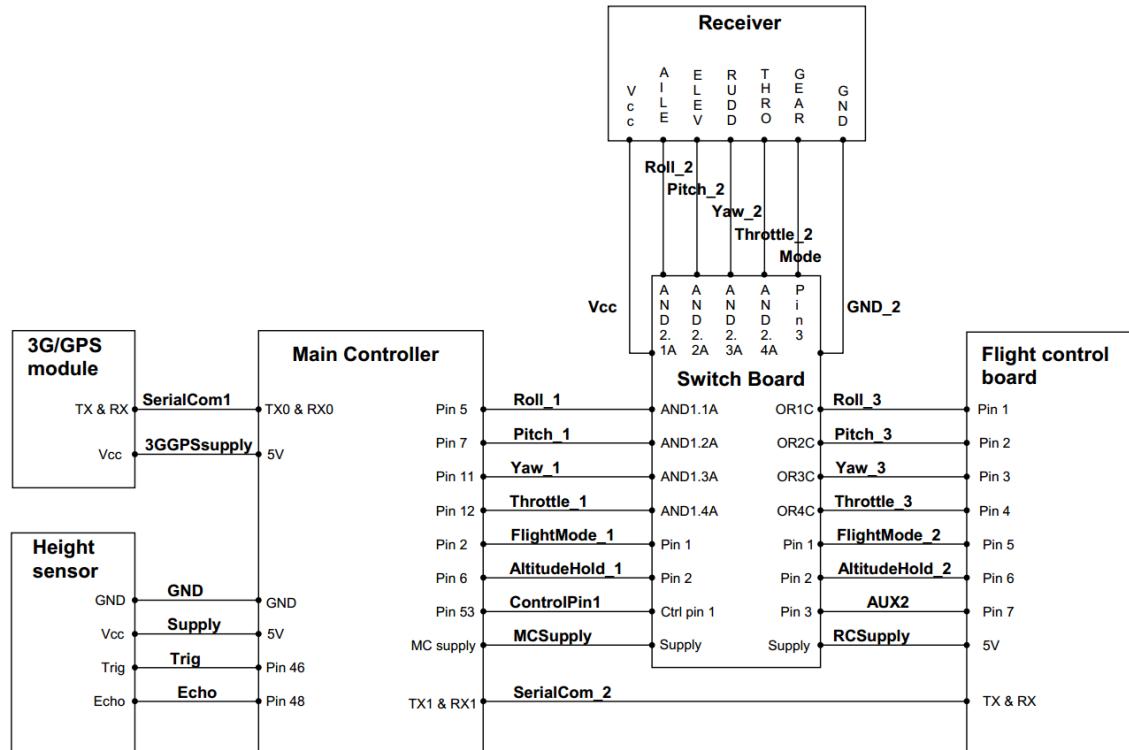
2.3.1 Hardware oversigt

I dette afsnit beskrives hvordan systemets forskellige hardware dele sammensættes. Først i afsnittet vises en stykliste der beskriver systemets hardware dele, dernæst vises hvordan hardware delene er koblet sammen.

Stykliste

- 3G/GPS module
- Ultralydssensor
- Main controller
- Flight control board
- Fjernbetjening + Receiver
- Switch board

De fem første moduler er færdigkøbte hardware moduler, kun switch boardet er designet og implementeret af gruppen. Til konstruktion af switch boardet er der brugt: To AND gates af typen 74HC08, en OR gate af typen 74HC32 og en INVERTER af typen 74HC04.

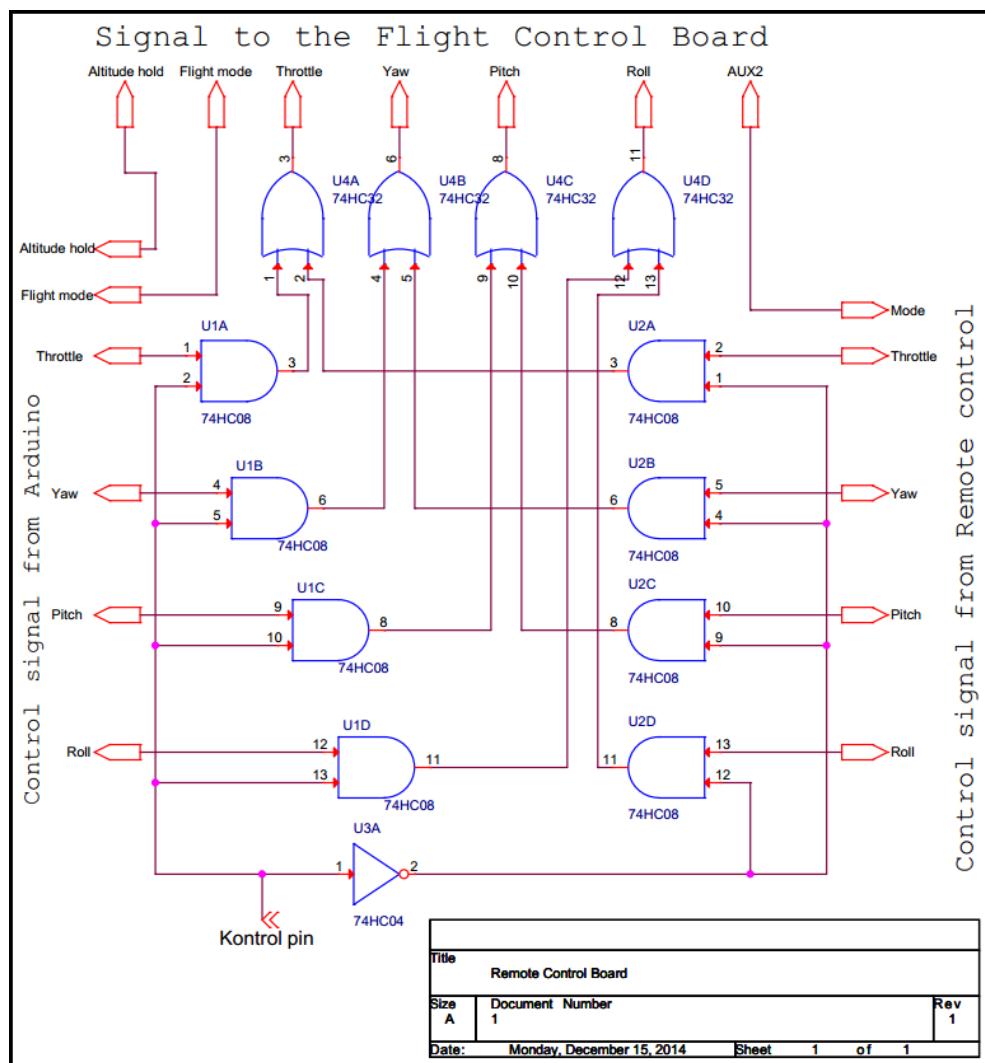


Figur 2.9: Switch board design

Switch board

I dette afsnit beskrives hardware designet af switch boardet. Boardet er designet så det kan switche mellem manuel og autonom flyvning. Når der flyves manuelt lader switch boardet styringssignaler fra remote controller styre dronen og når der flyves autonomt er det signaler fra main controller der styrer dronen.

På figur 2.10 vises designet af switch boardet. Styringssignaler fra main controller og remote controller sendes begge til switch boardet, hvor de sendes ind i 2-input AND gates. AND gates på højre side håndterer signaler fra remote controller og på venstre side håndteres signaler fra main controller. Hvilket output der kommer på de forskellige AND gates afhænger af signalet der sendes ind på kontrol pin. Outputtet fra AND gates med samme styringssignal (fx. Throttle) sendes til OR gates, som sender det rette styringssignal videre til flight control boardet.



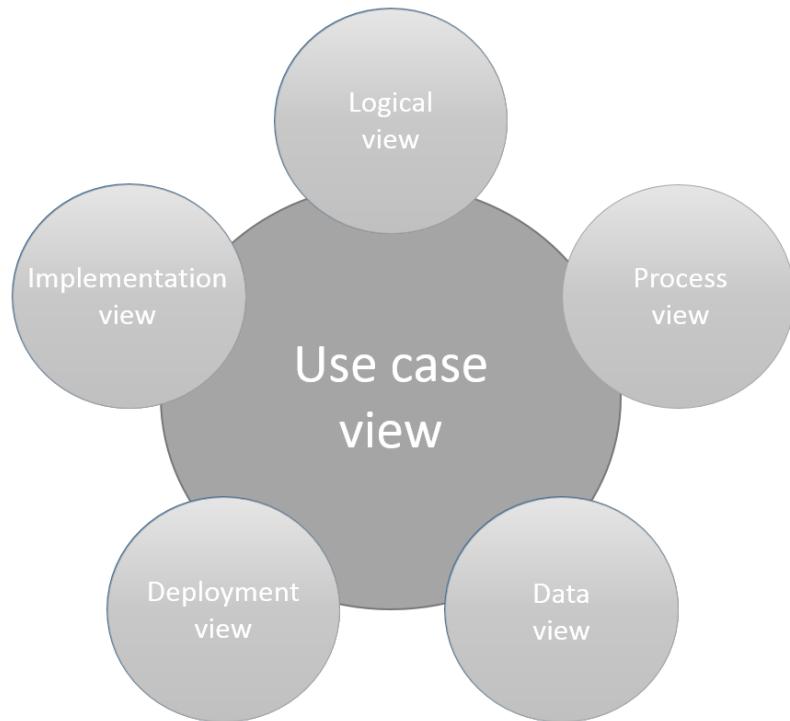
Figur 2.10: Switch board design

Software 3

I det følgende afsnit beskrives systemet softwarearkitektur vha. N+1 view modellen og en række UML diagrammer. I afsnittet gøres der bla. brug af state machines, designoverview-, sekvens-, klasse- og pakkediagrammer. N+1 view modellen tager udgangspunkt i de use cases der er beskrevet i kravspecifikationen, hvilket betyder at softwarearkitekturen er fuldt ud use case baseret.

3.1 N+1 view model

N + 1 view modellen bruges til at beskrive software fra forskellige views. Modellen tager altid udgangspunkt i et use case view. Derefter er det op til udviklerholdet at bestemme hvor mange views der skal bruges for at beskrive systemet på tilstrækkelig vis. Til beskrivelse af dette system anvendes en 5 + 1 view model. På figur 3.1 vises den anvendte model



Figur 3.1: 5 + 1 view model

3.1.1 View beskrivelse

Use case view

Use case viewet består af use case beskrivelser, der er udviklet ud fra brugers synspunkt og som bruges til at beskrive systemets forskellige brugsscenarier. Alle udarbejdede use case beskrivelser forefindes i kravspecifikationen.

Logical view

Logical viewet bygges op af følgende fem diagrammer: Designoverview, pakke-, sekvens-, klasse- og state machine diagrammer. Først udformes designoverview. Dernæst laves pakke og sekvensdiagrammer, og når de er udformet laves klassediagrammer. Til sidst udarbejdes state machines, som bruges til at beskrive flow mellem forskellige states.

Process view

Process viewet beskriver de forskellige processer/tråde i systemet og hvordan samspillet imellem disse er. Primært beskrives sammenspil mellem processer eller tråde der kører sideløbende med hinanden.

Data view

I data viewet beskrives layout af data der gemmes og hvordan det lagres. Desuden beskrives hvordan data sendes rundt i systemet og hvordan serverens database tilgås.

Deployment view

I deployment viewet vises hvilke software pakker der bruges i systemet og hvor de bruges. Desuden beskrives hvilke protokoller der er anvendt i systemet, fx. layout af meddelelser med header/start/stop.

Implementation view

I implementation viewet vises og beskrives hvilke værktøjer der er benyttet til projektet og hvordan disse værktøjer er sat op. Det beskrives også hvilke filer systemet er bygget af og hvordan disse filer skal linkes sammen. Desuden beskrives elementer som uden forklaring, ville være svære at forstå og bruge for udefrakommende. Målet med implementation viewet er at gøre det klart hvordan projektet er udarbejdet, og derved gøre det muligt for udefrakommende at genoptage arbejde på projektet.

3.2 Logical view

Logical view'et er udarbejdet efter de 4 iterationer defineret i kravspecifikationen. Den første iteration dækker systemets grundfunktionalitet, mens de øvrige iterationer bruges til at tilføje funktionalitet til systemet. I logical view'et vil der forekomme diagrammer som er gråskraveret. De elementer der er gråskraverede er udarbejdet i tidligere iteration. Iteration 1 og 2 er designet og implementeret, mens iteration 3 og 4 kun er designet.

Som beskrevet i afsnit 1.1.1 består logical viewet af designoverview, pakke-, sekvens-, klasse- og state machine diagrammer. Fælles for alle diagrammerne er, at de udformes ved brug af applikationsmodellen.

Applikationsmodel

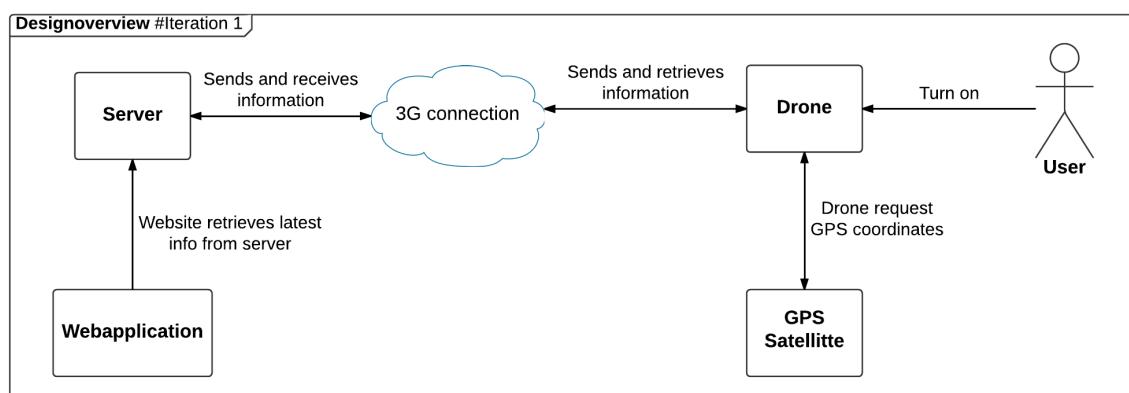
Applikationsmodellen tager udgangspunkt i use cases beskrevet i kravspecifikationen og domain modellen, se figur 1.1. Use cases og domain modellen bruges til at identificere softwarepakke og deres tilhørende klasser. Designoverview og sekvensdiagrammer fortæller og viser hvordan de forskellige klasser kommunikerer, mens state machines bruges til at vise systemflow.

3.2.1 Iteration #1

I iteration 1 arbejdes der med blokke som dækker systemets mest grundlæggende funktionalitet. Batteri, ESC'er, motorer og sensorer tilsluttes dronen, og dronen gøres desuden i stand til oprette forbindelse til webapplikationen via 3G-shield'et. Server opsættes, så basal kommunikation mellem dronen og server er muligt, bla. skal det være muligt for dronen at uploadre sin nuværende position til server. For flere detaljer vedrørende opsætning og funktionalitet af server henvises til data view'et.

User story

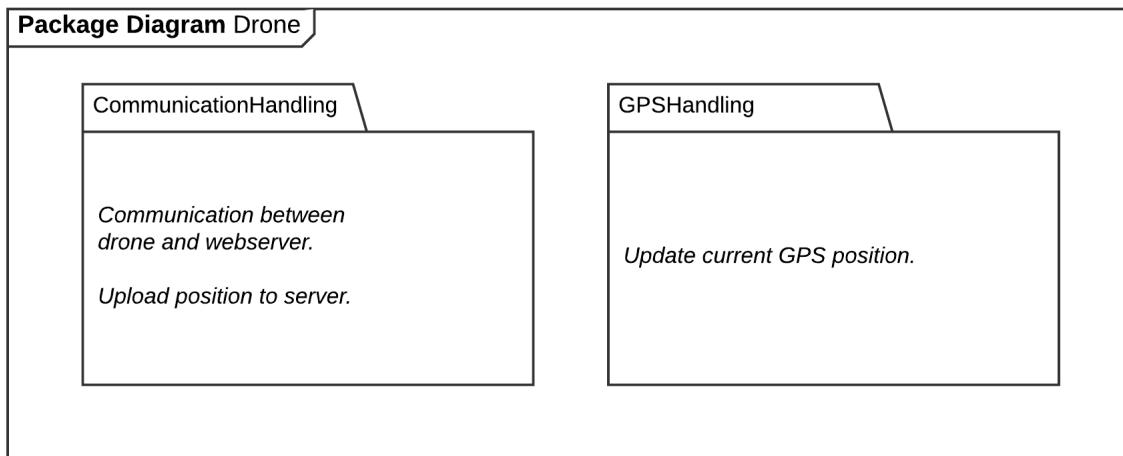
Bruger tænder dronen ved at tilslutte batteri. Main controller samt 3G/GPS module initialiseres og nuværende GPS position opdateres. Herefter opretter dronen forbindelse til webapplikation og sender information om sin nuværende GPS position. Fra webapplikationen er det muligt for bruger løbende at observere hvorvidt dronen er online og på hvilken GPS position dronen sidst har befundet sig.



Figur 3.2: Designoverview

Pakkediagram drone

I dette afsnit vises pakkediagram tilhørende drone. De pakker der vises i pakkediagrammet består af en eller flere klasser, der med stort samspil udfører opgaver indenfor et fælles ansvarsområde. På hver pakke findes en lille beskrivelse, der tydeliggør pakkens ansvarsområde.



Figur 3.3: Pakkediagram drone

CommunicationHandling

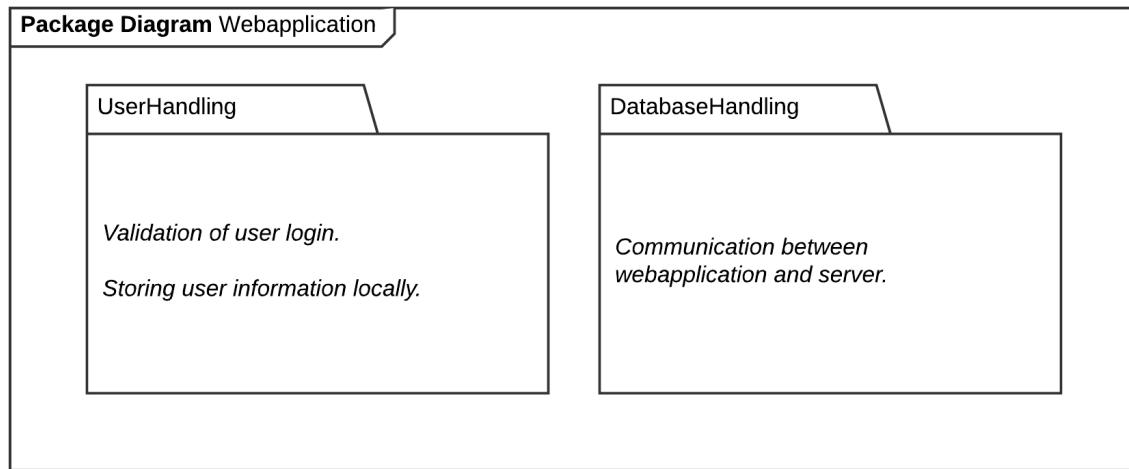
Pakkens ansvar er kommunikation imellem drone og server. I denne iteration er der fokus på, at dronen skal kunne sende sin nuværende GPS position til server.

GPSHandling

Pakkens ansvar er håndtering af GPS. Dels er pakken ansvarlig for opstart og initiering af GPS, og desuden bruges pakken hver gang dronens nuværende GPS position skal opdateres.

Pakkediagram webapplikation

I dette afsnit vises pakkediagram tilhørende webapplikationen. De pakker der vises i pakkediagrammet består af en eller flere klasser, der med stort samspil udfører opgaver indenfor et fælles ansvarsområde. På hver pakke findes en lille beskrivelse, der tydeliggør pakkens ansvarsområde.



Figur 3.4: Pakkediagram webapplikationen

UserHandling

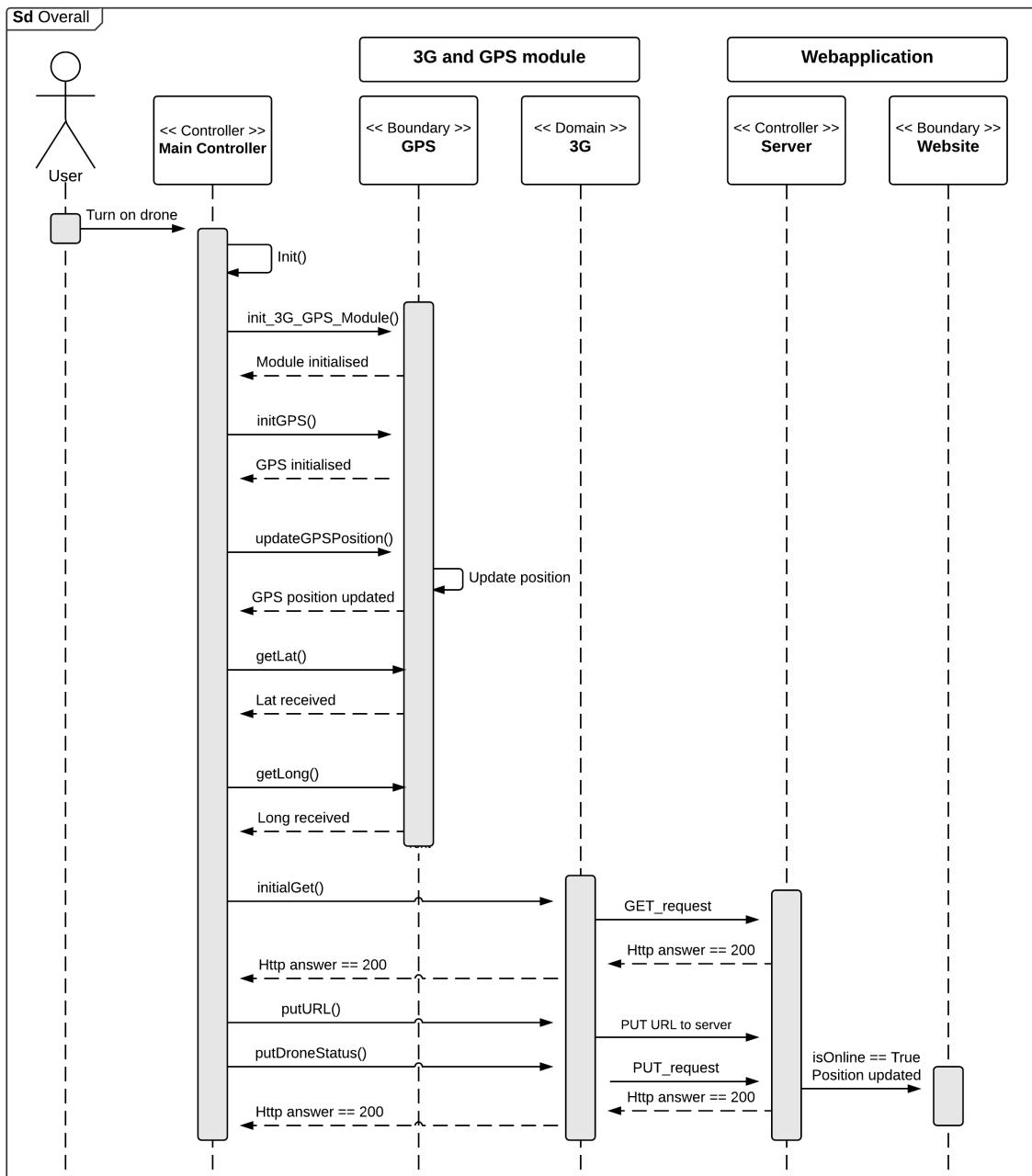
Pakkens ansvar er validering af login/log ud på webapplikationen. Pakken har også ansvaret for at hente og gemme data om den pågældende bruger.

DatabaseHandling

Pakkens ansvar er kommunikation imellem database og server.

Sekvensdiagram drone

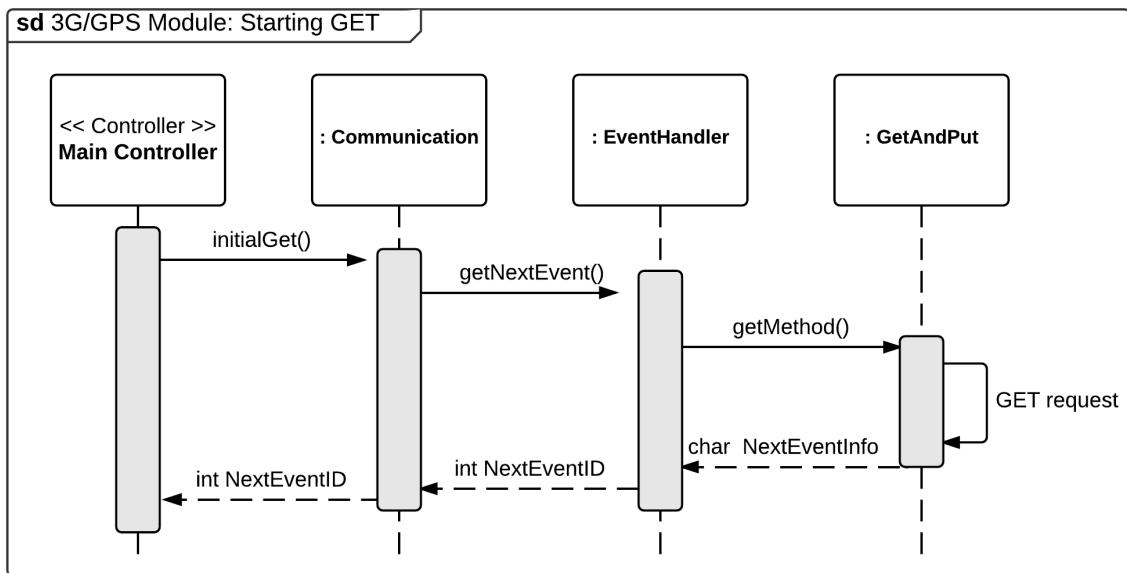
På sekvensdiagrammet på figur 3.5, vises hvilke klasser der indgår og bruges i første iteration. Af sekvensdiagrammet fremgår det, at sekvensen først startes når bruger tilslutter batteri og tænder dronen. Når der er tilkoblet forsyning initialiseres main controller samt 3G/GPS modulet og nuværende GPS position (longitude og latitude) opdateres. Dronens nuværende GPS position sendes via PUT request til webapplikation. PUT requestet bruges dels til at fortælle webapplikationen at dronen er online og dels til at give webapplikationen information om dronens nuværende position.



Figur 3.5: Overordnet sekvensdiagram

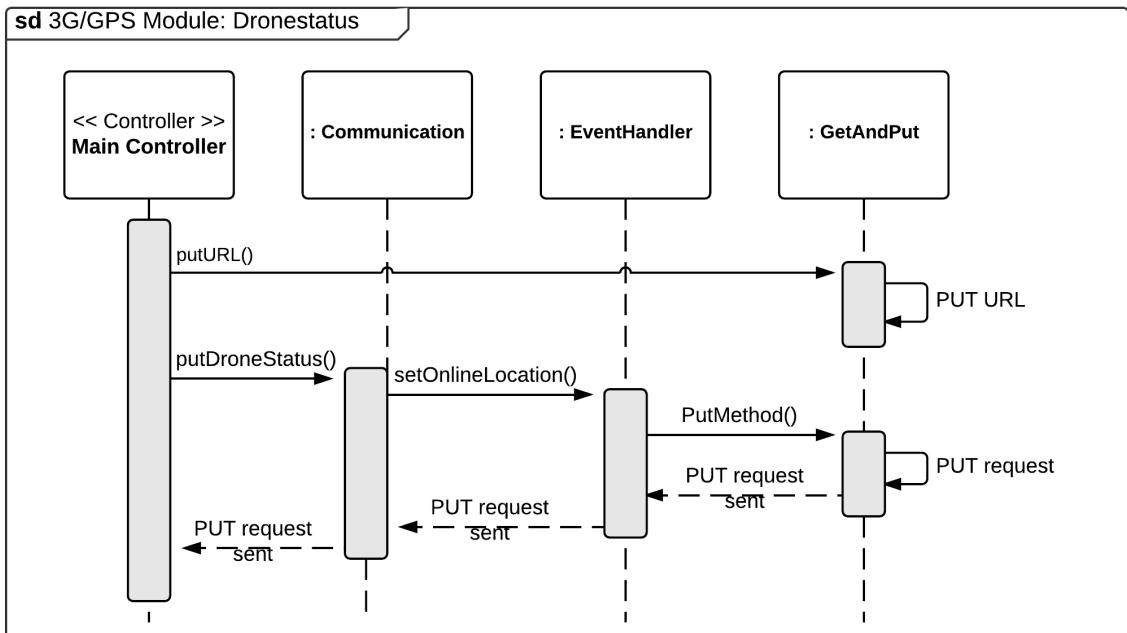
Da boundary klassen *3G* benytter tre underliggende klasser til håndtering af GET og PUT requests er sekvensdiagrammet på figur 3.5 simplificeret. Sekvensdiagrammerne på figur 3.6 og 3.7 vises detaljeret hvordan *3G* og underliggende klasser bruges til GET og PUT.

Sekvensdiagrammet på figur 3.6 viser hvilke klasser der anvendes til håndtering af GET.



Figur 3.6: Sekvensdiagram - initialget

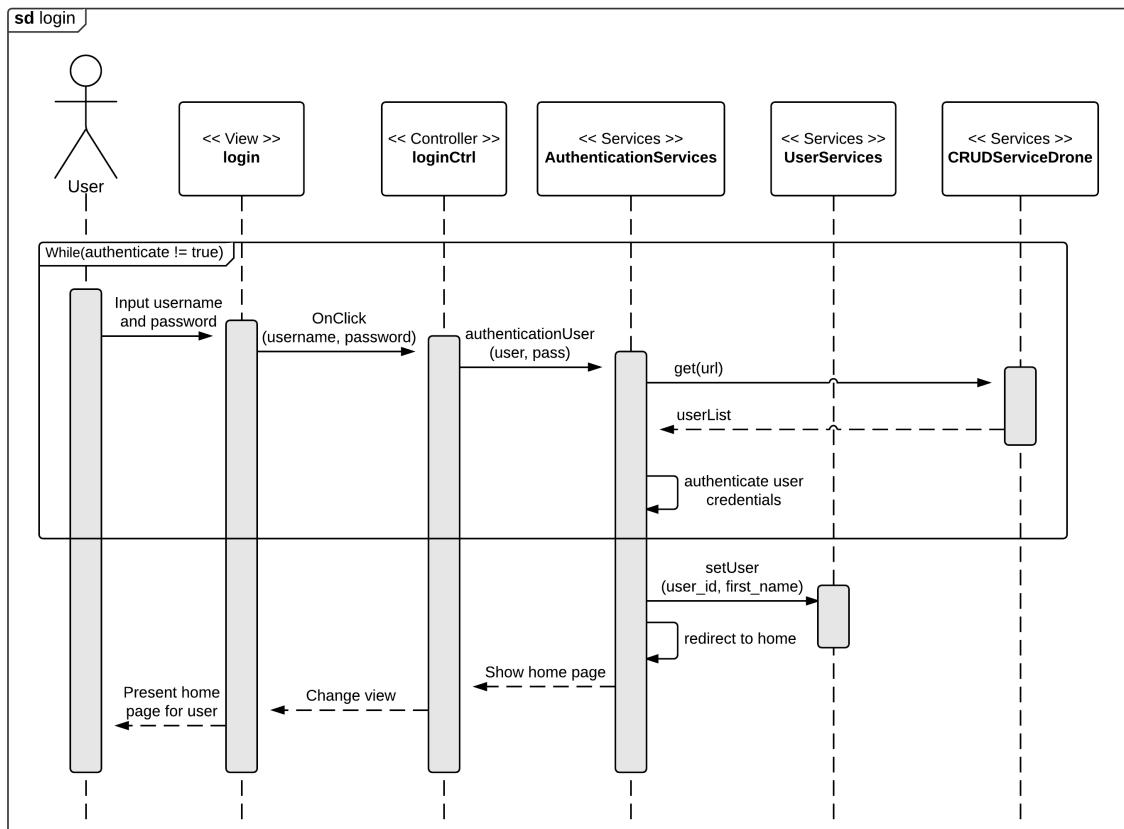
Sekvensdiagrammet på figur 3.7 viser hvilke klasser der anvendes til håndtering af PUT.



Figur 3.7: Sekvensdiagram - putDroneStatus

Sekvensdiagram webapplikation

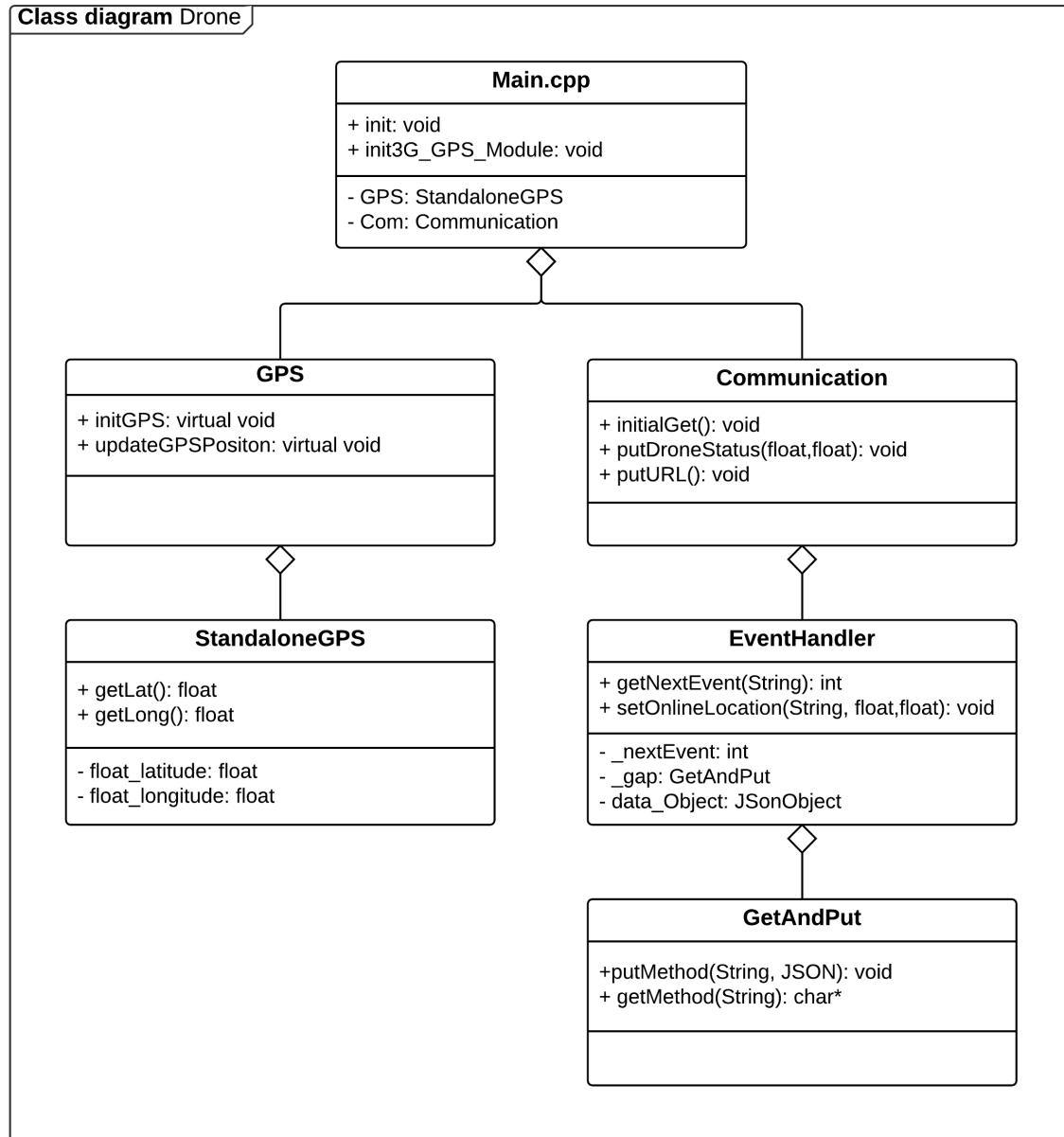
På figur 3.8 ses sekvensdiagrammet over login på webapplikationen. På diagrammet vises det at systemets bruger først bliver ført til næste side ved succesfuld login. Diagrammet viser også kommunikationen med CRUDServiceDrone og hvordan AuthenticationServices håndterer bruger data.



Figur 3.8: Sekvensdiagram - login

Klassediagram drone

På figur 3.9 vises et klassediagram som viser iterationens klasser, samt deres tilhørende metoder og attributter. På den følgende side forefindes en kort beskrivelse kasserne og deres ansvarsområder.



Figur 3.9: Klassediagram drone

Main.cpp

Main.cpp filen bruges til at sætte arduino board korrekt op, bla. sættes baudrate på de forskellige serielle forbindelser. Desuden bruges Main.cpp til at kalde og eksekverer forskellige klasse, objekter og funktioner.

GPS

GPS klassen er implementeret som en abstract klasse. Init og updateGPSPosition er lavet som virtuelle metoder, hvilket betyder de skal implementeres i alle afledte klasser. GPS klassen er lavet fordi 3G/GPS modulet kan bruges i 3 forskellige GPS modes.

StandaloneGPS

Denne klasse er ansvarlig for al kommunikation med GPS'en når standalone mode er valgt.

GetAndPut

GetAndPut klassen er den klasse der er tættest på hardwaren. Klassen indeholder http metoder der bruges til kommunikation mellem dronen og serveren.

Communication

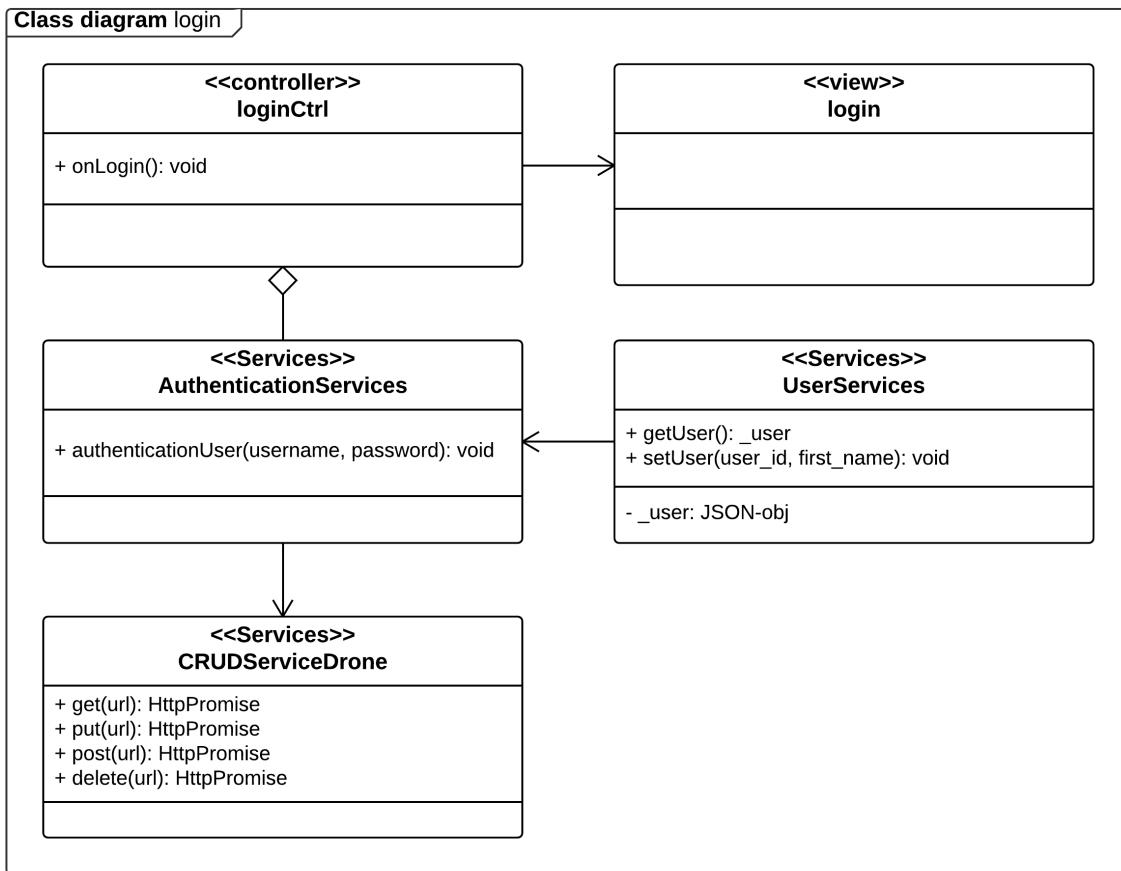
Communication klassen styrer alt der har med 3G kommunikation at gøre.

EventHandler

EventHandler klassen håndterer Events, og bruges som bindelede mellem communication- og GetAndPut klassen. EventHandleren sorterer eventID'et fra data der modtages fra server og returnerer værdien af eventID til communication klassen.

Klassediagram webapplikation

Figur 3.10 viser klassediagrammet tilhørende webapplikation til iteration 1.



Figur 3.10: Klassediagram webapplikation

login

Denne klasse indeholder view'et til webapplikationen, og udgør sammen med loginCtrl klassen GUI på webapplikationen.

loginCtrl

LoginCtrl er en controller klasse som er forbundet med view klassen via two-way databinding. Desuden uddeleger klassen opgaver til de underliggende services.

AuthenticationServices

AuthenticationServices klassen er ansvarlig for at afgøre om bruger er authenticated eller ej. Hvis bruger er authenticated bliver han/hun redirected til webapplikations home-page. Klassen gør brug af UserServices klassen til at gemme information om den bruger der er logget ind på webapplikationen.

UserServices

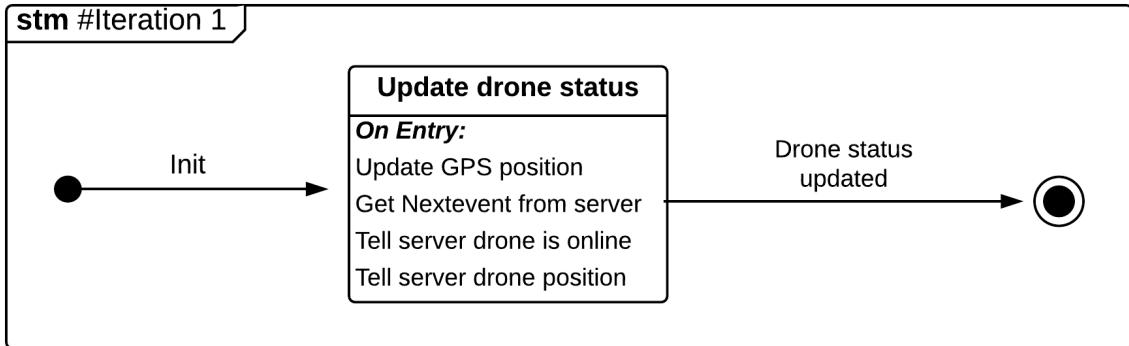
UserServices bruges til at gemme information om hvilken bruger der er logget ind på webapplikationen.

CRUDServicesDrone

Denne klasse bruges som bindeled imellem server og webapplikation, og indeholder logik til serveren. CRUDServicesDrone bruges når der sendes Get, Put og Post til serveren.

State machine diagram

I state machine diagrammet på figur 3.11, vises de forskellige states der eksisterer i iteration 1. Der eksisterer givetvis kun 1 state i iteration 1, men state machinen er medtaget fordi den bygges ovenpå den i de efterfølgende iterationer.



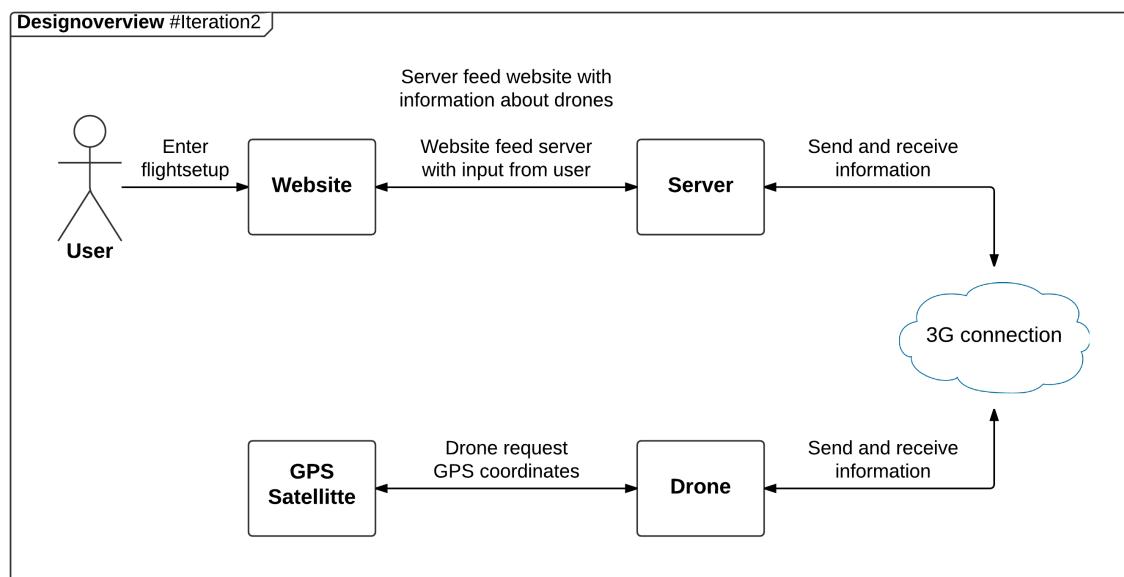
Figur 3.11: State machine

3.2.2 Iteration #2

I iteration 2 udvides systemet, så bruger kan logge på webapplikationen og lave flyveopsætninger, som dronen skal flyve ud fra. Funktionaliteten af webapplikationen udvides, så det bliver muligt at lave flyveopsætninger. Når en flyveopsætning er lavet, skal den gøres tilgængelig for dronen, hvilket kræver tilføjelser i kommunikationen mellem drone og server. Desuden skal dronen i denne iteration kunne finde egen GPS position, flyvehøjde og orientering. Ud fra viden om egen position, flyvehøjde og orientering skal dronen flyve til de lokationer som er fastsat i flyveopsætningen.

User story

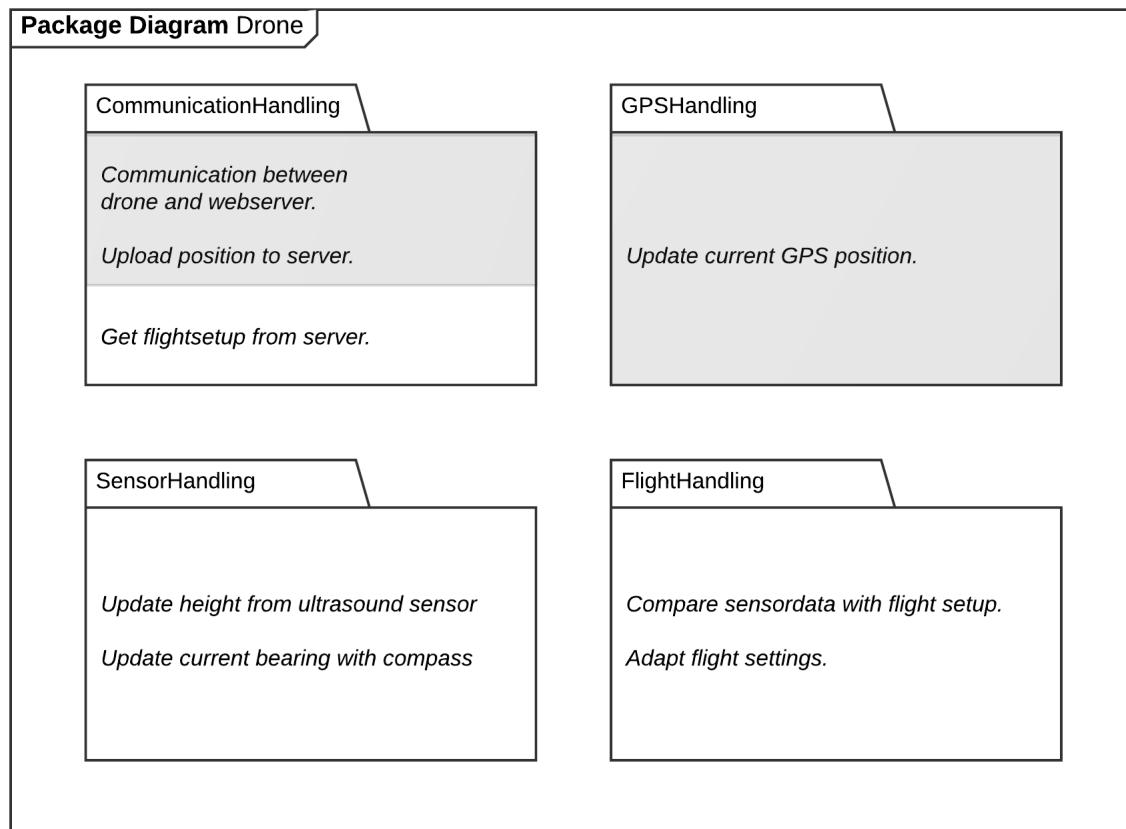
Bruge logger på webapplikationen med sit brugernavn og password. Når der er logget korrekt ind vises bruger sin eller sine droner på en liste. Ved at trykke på en drone vises bruger information om den pågældende drone. Informationen består af waypoints, om der skal tages billede ved de forskellige waypoints samt indstilling af flyvehøjde. Bruger klikker på en drone fra listen og klikker derefter på kortet for at tilføje waypoints som tilsammen udgør et event. Bruger tildeler eventet navn og en kommentar og trykker "publish to drone". Når en drone er tændt og færdig initialiseret fortæller den server om sin nuværende position og kontrollerer om en ny flyveopsætning er tilgængelig. Er der en ny flyveopsætning tilgængelig hentes den og dronen påbegynder flyvning.



Figur 3.12: Designoverview #iteration 2

Pakkediagram drone

I dette afsnit vises pakkediagram tilhørende drone. De pakker der vises i pakkediagrammet består af en eller flere klasser, der med stort samspil udfører opgaver indenfor et fælles ansvarsområde. På hver pakke findes en lille beskrivelse, der tydeliggør pakkens ansvarsområde.



Figur 3.13: Pakkediagram drone

CommunicationHandling

Pakkens ansvar er kommunikation imellem drone og server. Efter denne iteration skal dronen både kunne hente flyveopsætninger fra server og sende sin nuværende GPS position til server.

GPSHandling

Pakkens ansvar er håndtering af GPS. Dels er pakken ansvarlig for opstart og initiering af GPS, og desuden bruges pakken hver gang dronens nuværende GPS position skal opdateres.

SensorHandling

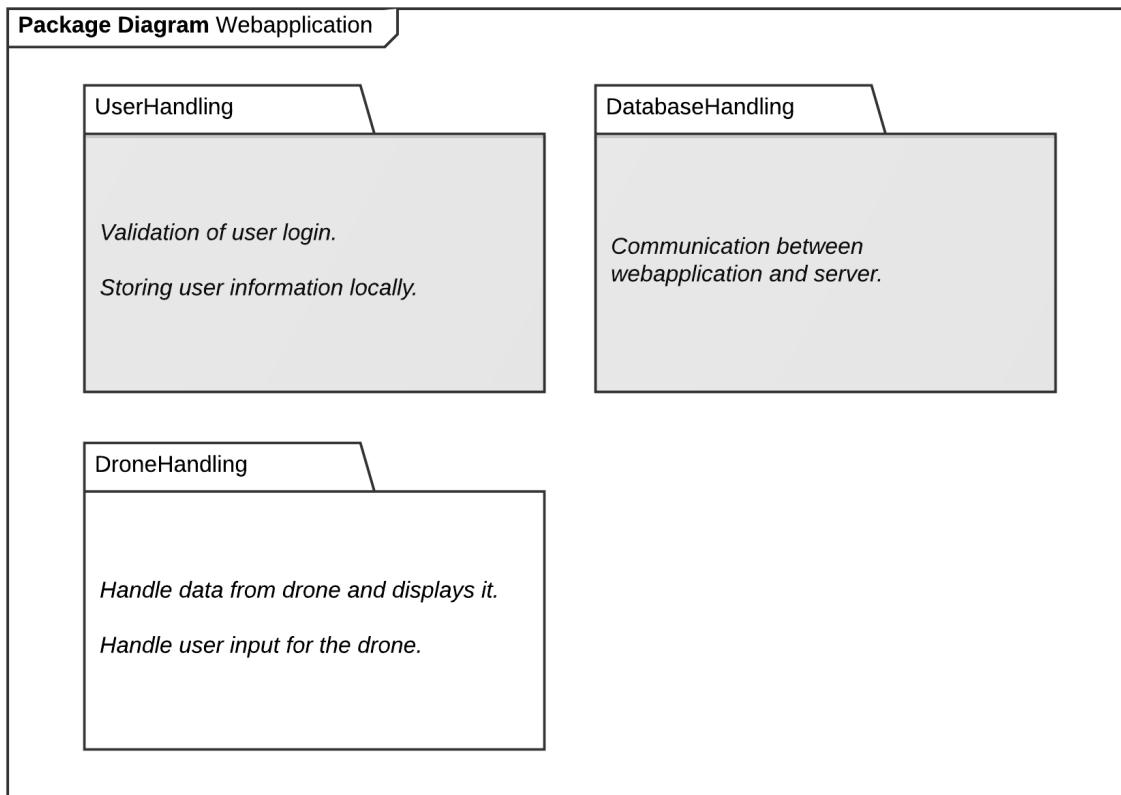
Pakken er ansvarlig for indsamling af sensor data. I denne iteration skal pakken bruges til aflæsning af højdemåler og kompasset på flight control boardet.

FlightHandling

Pakkens ansvar er kontrol og styring af drone under flyvning. Ved sammenligning af sensor data og data fra flyveopsætning tilpasses flyvehøjde, orientering mm

Pakkediagram webapplikation

I dette afsnit vises pakkediagram tilhørende webapplikation. De pakker der vises i pakkediagrammet består af en eller flere klasser, der med stort samspil udfører opgaver indenfor et fælles ansvarsområde. På hver pakke findes en lille beskrivelse, der tydeliggør pakkens ansvarsområde. De dele af pakkerne der er gråskraveret, er funktionalitet udarbejdet i tidlige iteration.



Figur 3.14: Pakkediagram webapplikationen

UserHandling

Pakkens ansvar er validering af login/log ud på webapplikationen. Pakken har også ansvaret for at hente og gemme data om de forskellige brugere.

DatabaseHandling

Pakkens ansvar er kommunikation imellem databasen og serveren.

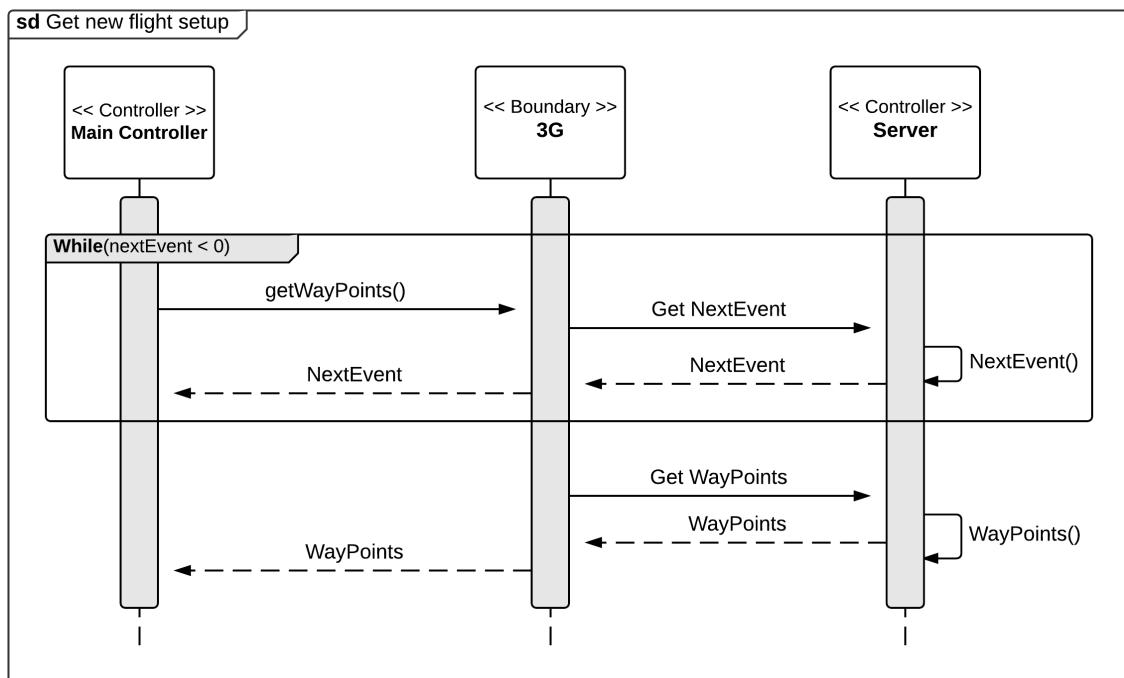
DroneHandling

Pakken er ansvarslig for håndtering af events, waypoints og forskellige droner.

Sekvensdiagram drone

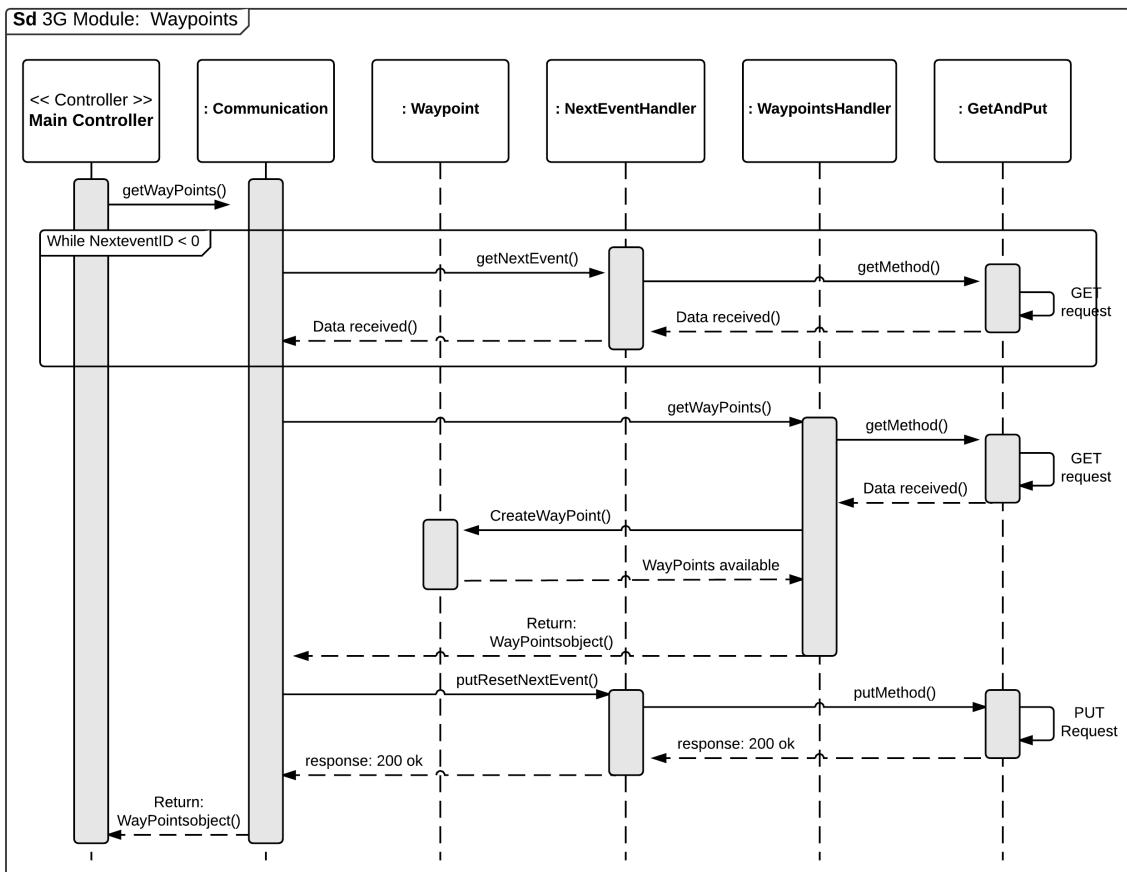
Til iteration 2 er systemsekvensen for drone beskrevet med flere mindre sekvensdiagrammer i stedet for et stort. Denne fremstilling gør det muligt at repræsentere systemets funktionalitet på mere overskuelig vis, hvilket bør øge forståelsen af diagrammerne. Det første sekvensdiagram beskriver kommunikation mellem main controller, 3G og server. Det næste går mere i dybden med 3G modulet og underliggende klasser. Det sidste diagram viser hvordan dronen agerer under flyvning.

Af figur 3.15 fremgår det hvordan dronens main controller via 3G-shieldet kommunikerer med serveren for at kontrollere om en ny flyveopsætning er tilgængelig. Når en flyveopsætning sættes tilgængelig hentes den af dronen.



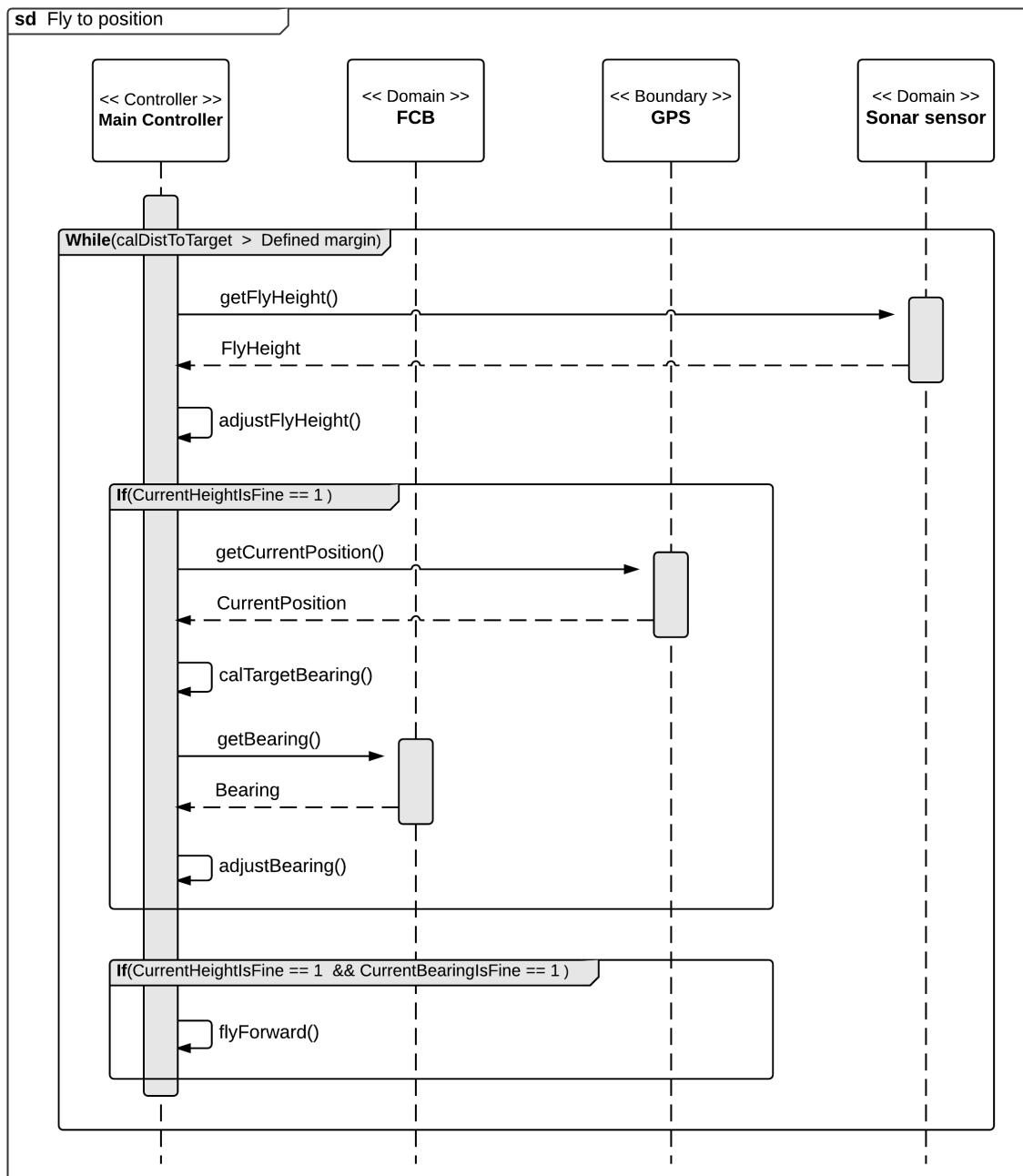
Figur 3.15: Sekvensdiagram - hent flyveopsætning

Sekvensdiagrammet på figur 3.16 viser en mere detaljeret beskrivelse af hvordan 3G og underliggende klasser fungerer. 3G og de underliggende klasser håndterer al kommunikation mellem drone og server.



Figur 3.16: Sekvensdiagram - Detaljeret kommunikation

Af figur 3.17 fremgår det hvordan dronen agerer når den flyver autonomt mod en given GPS position. Main controlleren indsamler kompas data fra flight control boardet, latitude og longitude fra GPS og flyvehøjde fra ultralyds sensor. De indhentede data processeres og bruges til at korrigere og optimere de nuværende flyveindstillinger. Som det fremgår af while loopet fortsætter flyvningen indtil dronen er tilpas tæt på den GPS destination bruger valgte i flyveopsætningen.

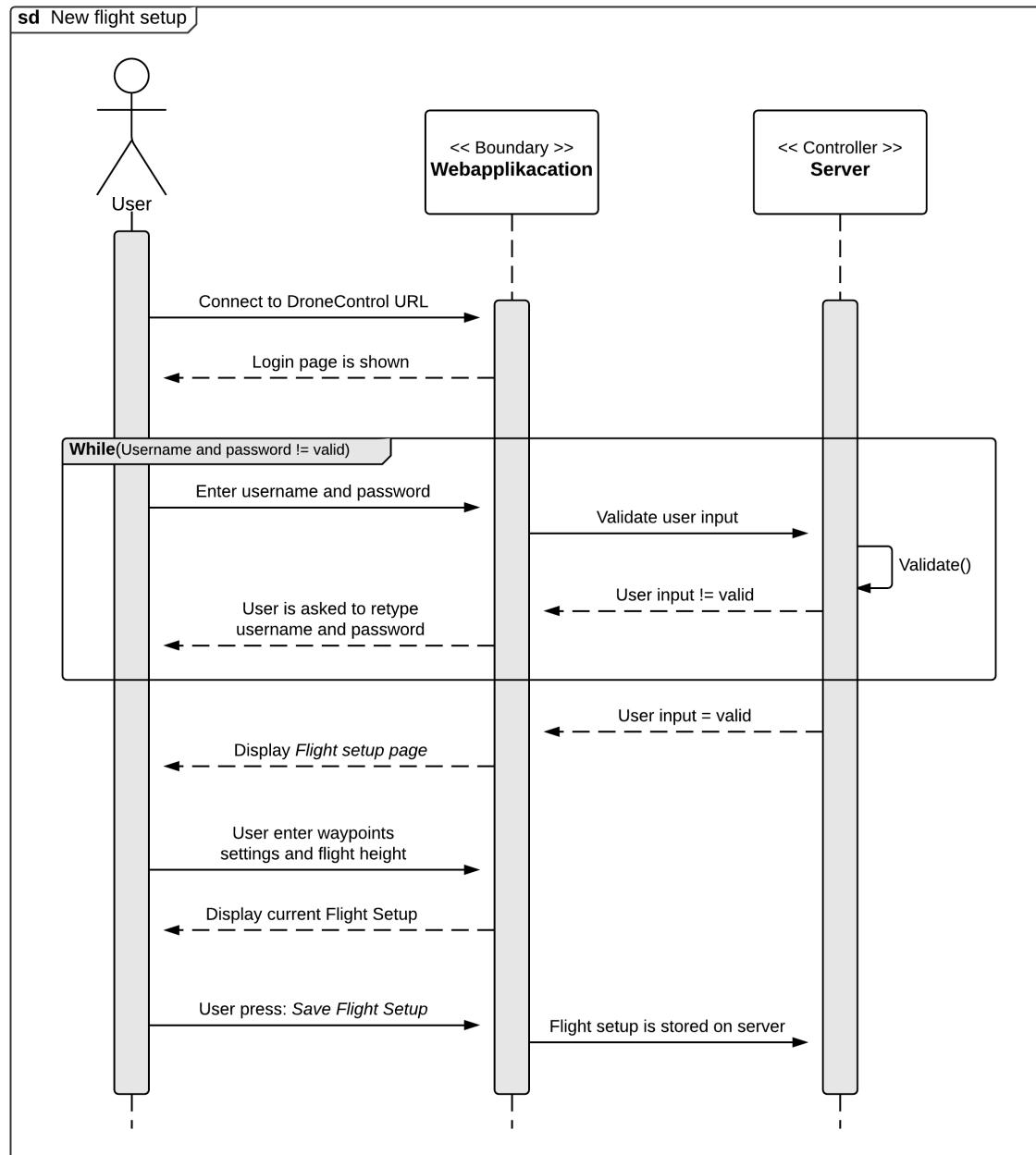


Figur 3.17: Sekvensdiagram - Drone under flyvning

Sekvensdiagram webapplikation

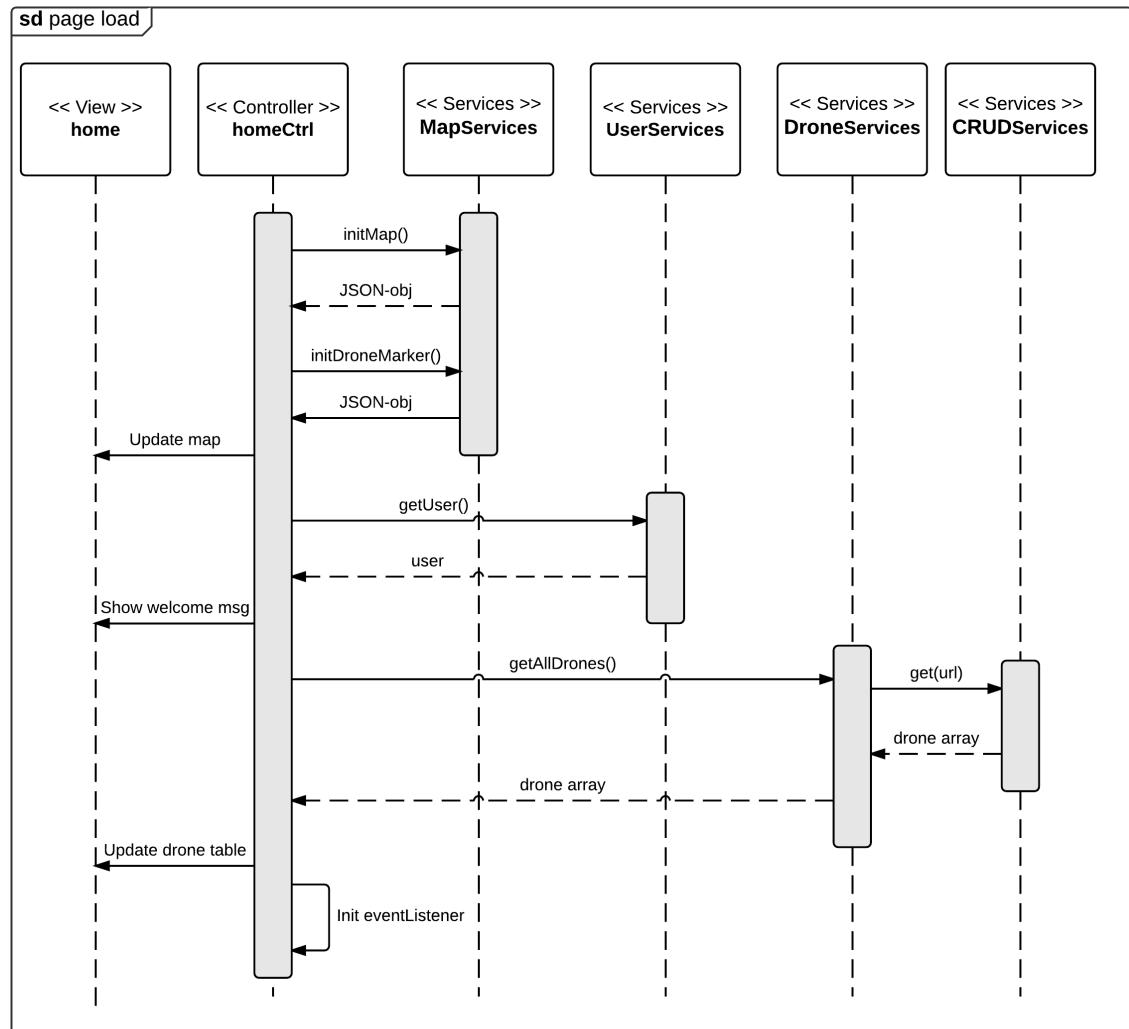
Der er udformet 4 forskellige sekvensdiagrammer for webapplikationen tilhørende iteration 2. Det første sekvensdiagram viser hvordan bruger logger ind på webapplikationen og opretter en flyveopsætning. Det næste sekvensdiagram viser hvordan webapplikationen fungerer og reagerer på bruger input.

Af figur 3.18 fremgår det hvordan bruger opretter en ny flyveopsætning. Det vises desuden hvilke interaktioner der foretages mellem bruger og webapplikation, samt hvordan server løbende indgår og benyttes i sekvensen.



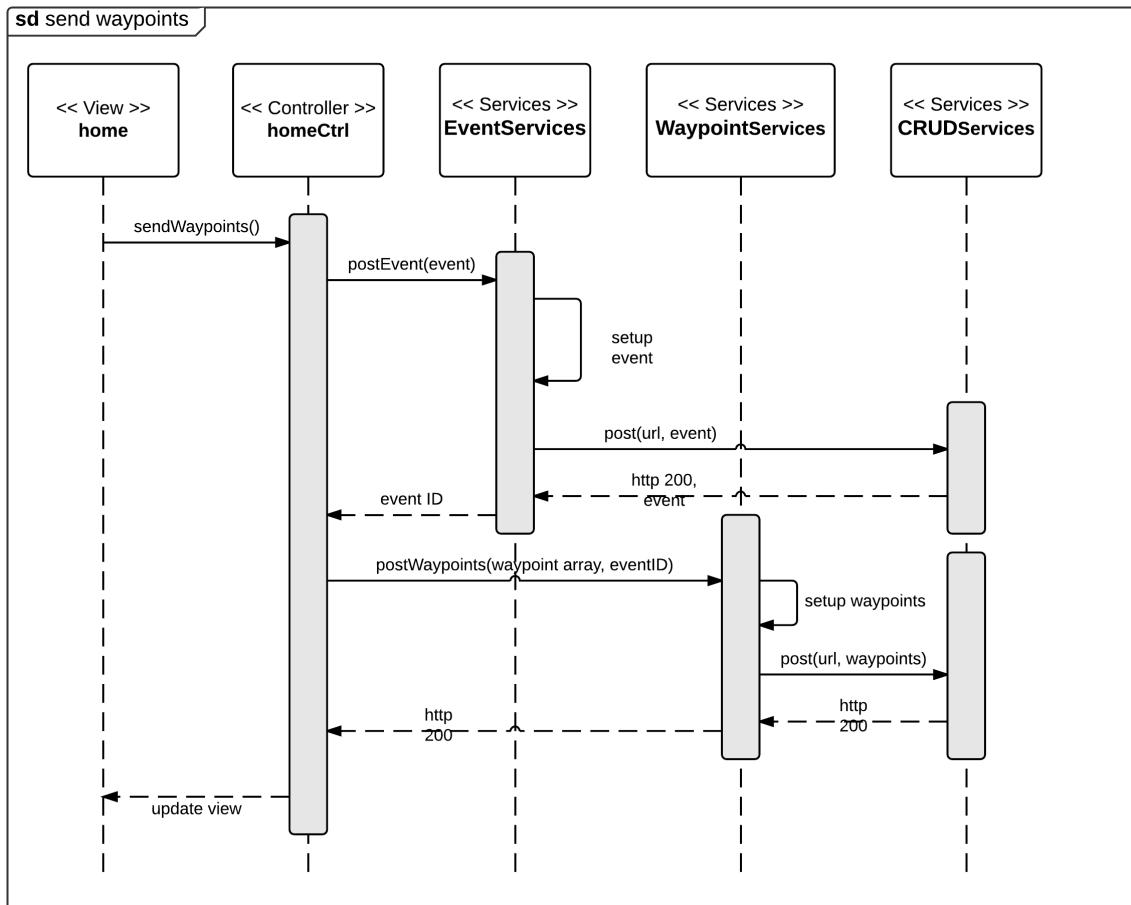
Figur 3.18: Sekvensdiagram

Sekvensdiagrammet på figur 3.19 viser hvordan klasserne i webapplikationen interagere med hinanden ved et page load.



Figur 3.19: Sekvensdiagram page load

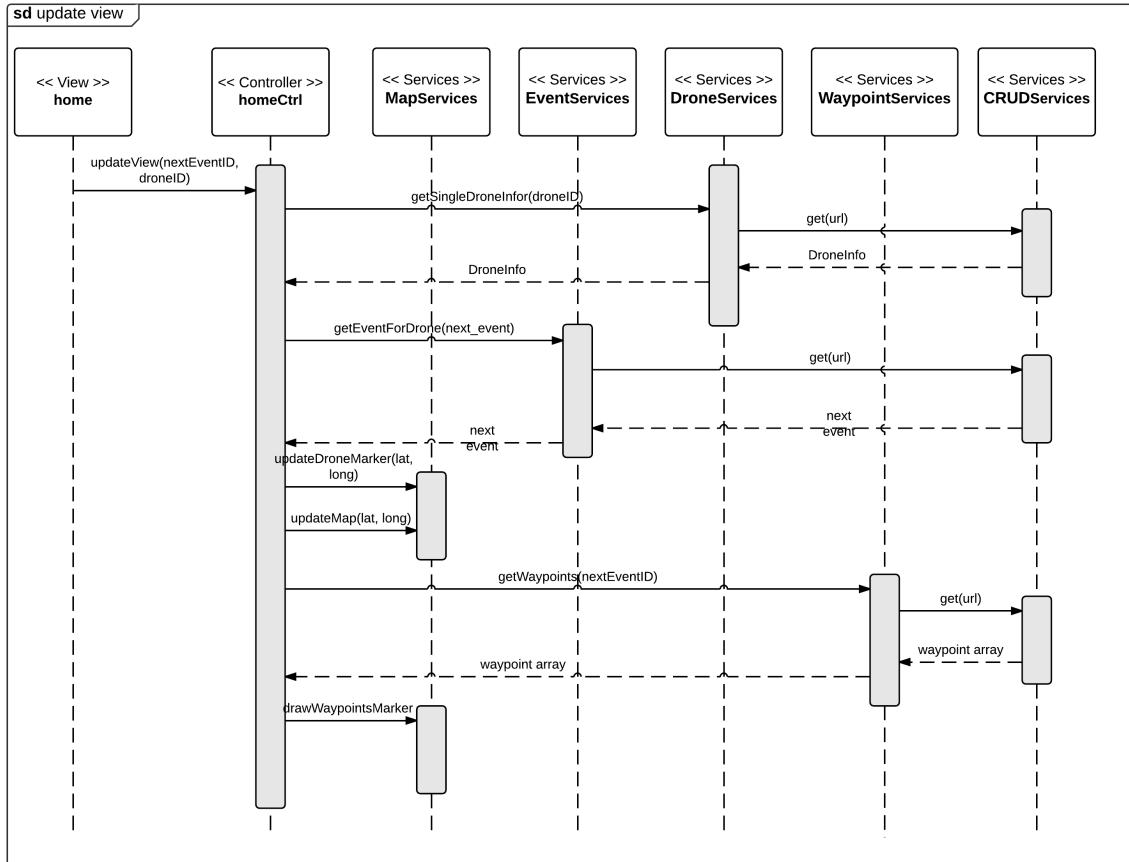
På figur 3.20 hvad der sker, når bruger gemmer en flyveopsætning. Hver service har et bestemt ansvar og de kommunikerer ud til CRUD-servicesen. Denne opbygning sikrer at logikken for data håndtering skubbes ud i diverse services.



Figur 3.20: Sekvensdiagram send waypoints

På figur 3.21 ses hvilke hændelser der finder sted, når bruger trykker på forskellige droner i tabellen.

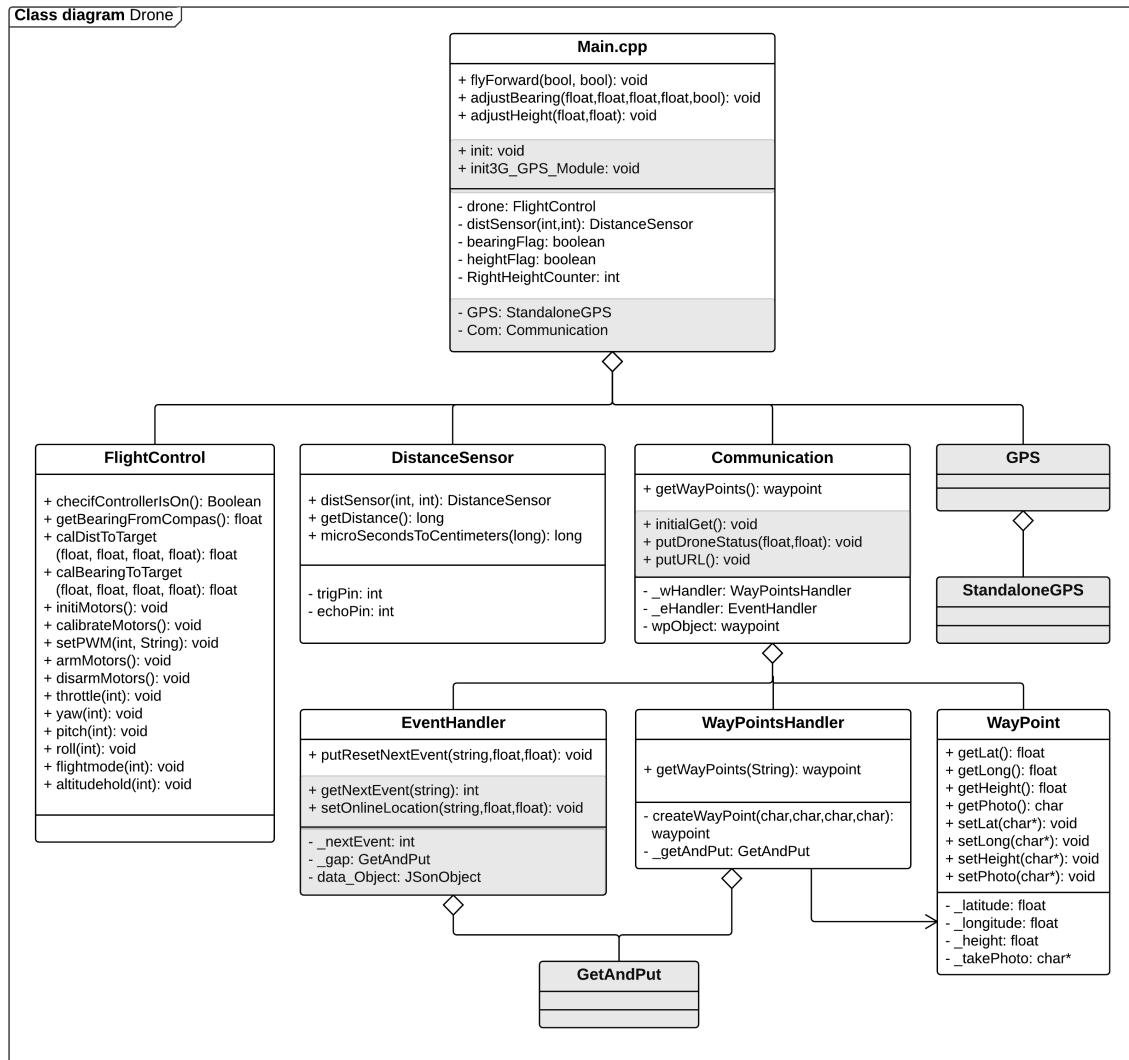
Når et klik på tabellen finder sted, skal view'et opdateres i forhold til den givne drone. Har dronen et event, skal eventet og dertilhørende waypoints vises. Hvis der ikke er noget event tilknyttet dronen, får bruger mulighed for at oprette et nyt.



Figur 3.21: Sekvensdiagram opdater

Klassediagram drone

På figur 3.22 ses klassediagrammet for drone. I det følgende findes en forklaring af klasserne og deres ansvarsområder.



Figur 3.22: Klassediagram drone

FlightControl

FlightControl klassen står for alt styring af dronen. Bla. står FlightControl for kalibrering og styring af motorerne, samt aflæsning af højdesensor, GPS og kompas.

DistanceSensor

DistanceSensor klassen bruges til at kontrollere de sensorer der er monteret på dronen. DistanceSensor klassen bruges udelukkende til kontrol af sensore til højdemåling.

WayPointsHandler

WayPointsHandler klassen håndterer de waypoints der hentes ned fra server. Klassen tager de hentede waypoints og gør dem tilgængelige for main.cpp. WayPointsHandleren gør brug af set-metoder fra WayPoint klassen.

WayPoint

WayPoint klassen bruges til at hente waypoints.

Main.cpp

Main.cpp filen bruges til at sætte arduino board korrekt op, bla. sættes baudrate på de forskellige serielle forbindelser. Desuden bruges Main.cpp til at kalde og eksekverer forskellige klasse, objekter og funktioner.

GPS

GPS klassen er implementeret som en abstract klasse. Init og updateGPSPosition er lavet som virtuelle metoder, hvilket betyder de skal implementeres i alle afledte klasser. GPS klassen er lavet fordi 3G/GPS modulet kunne bruges i 3 forskellige GPS modes.

StandaloneGPS

Denne klasse er ansvarlig for al kommunikation med GPS'en når standalone mode er valgt.

GetAndPut

GetAndPut klassen er den klasse der er tættest på hardwaren. Klassen indeholder http metoder der bruges til kommunikation mellem dronen og serveren.

Communication

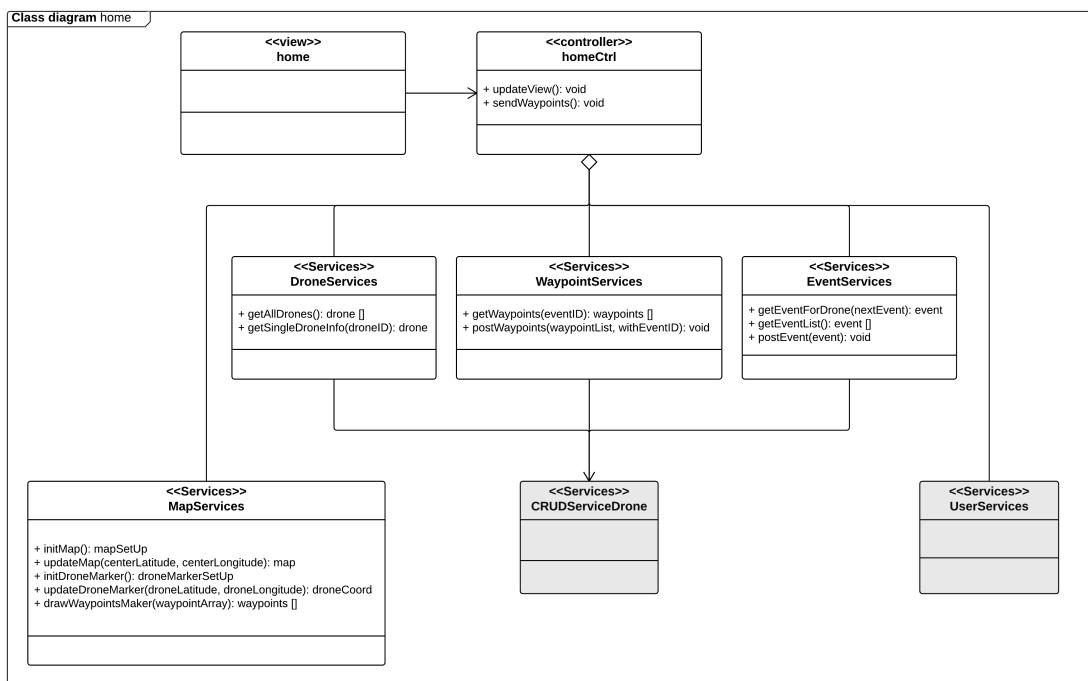
Communication klassen styrer alt der har med 3G kommunikation at gøre.

EventHandler

EventHandler klassen håndterer Events, og bruges som bindeled mellem communication- og GetAndPut klassen. EventHandleren sorterer eventID'et fra data der modtages og returnerer værdien af eventID til communication klassen.

Klassediagram webapplikation

På figur 3.23 ses klasse diagrammet tilhørende iteration 2 for webapplikationen. Funktionaliteten af systemet er blevet udvidet, så der nu er overblik over droner i systemet og deres status. Desuden er der mulighed for at oprette events til bestemte droner.



Figur 3.23: Klassediagram home

home Denne klasse er view'et tilhørende iteration to. Denne klasse sammen med homeCtrl skaber det endelig view som brugeren ser i sin browser når han besøger siden.

homeCtrl HomeCtrl klassen er controller klassen til iteration to. Det er den eneste klasse der har direkte forbindelse til view'et, homeCtrl klassen deler også hukommelse med view'et igennem two-way-binding med scopes.

DroneServices Denne klasse indeholder logikken om drone håndteringen. Den har ansvaret for at hente informationer omkring droner fra serveren via CRUDServiceDrone klassen.

WaypointServices Denne klasse indeholder logikken om waypoint håndtering. Denne klasse henter henter waypoints tilhørende et event til controller klassen. Klassen bliver også brugt til at sende waypoints til serveren via CRUDServiceDrone klassen.

EventServices Denne klasse indeholder logikken om event håndtering. Klassen bruges til at hente en event liste, et enkelt event for en givet drone og sende et nyoprettet event til serveren via CRUDServiceDrone klassen.

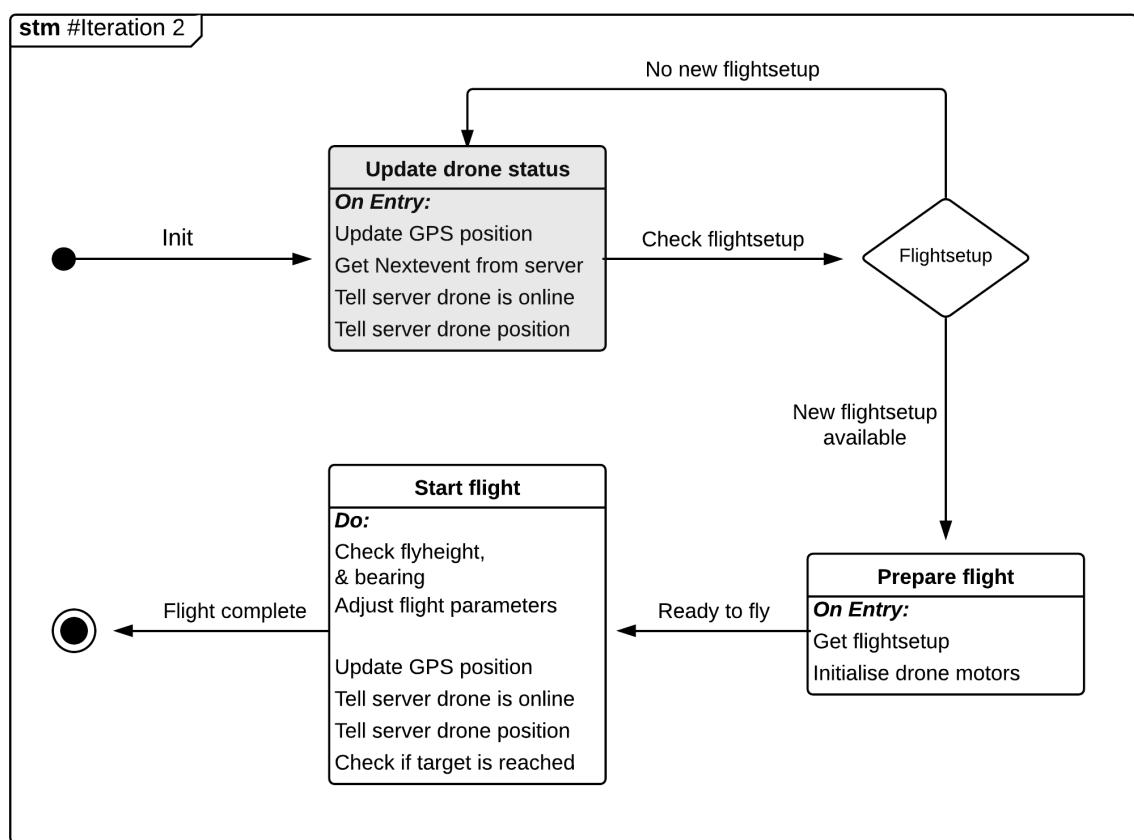
MapServices Denne klasse indeholder logikken om map håndtering. Klassen bruges til at opdatere kortet i view'et, tegne waypoints og dronen på kortet.

CRUDServicesDrone Denne klasse bruges som bindeled imellem server og webapplikation, og indeholder logik til serveren. CRUDServicesDrone bruges når der sendes Get, Put og Post til serveren.

UserServices Denne klasse blev oprettet i iteration et og bruges til at gemme information omkring hvilke bruger der er logget ind i systemet.

State machine diagram

I state machine diagrammet på figur 3.24, vises de forskellige states der eksisterer til iteration 2 og hvordan flowet imellem dem ser ud.



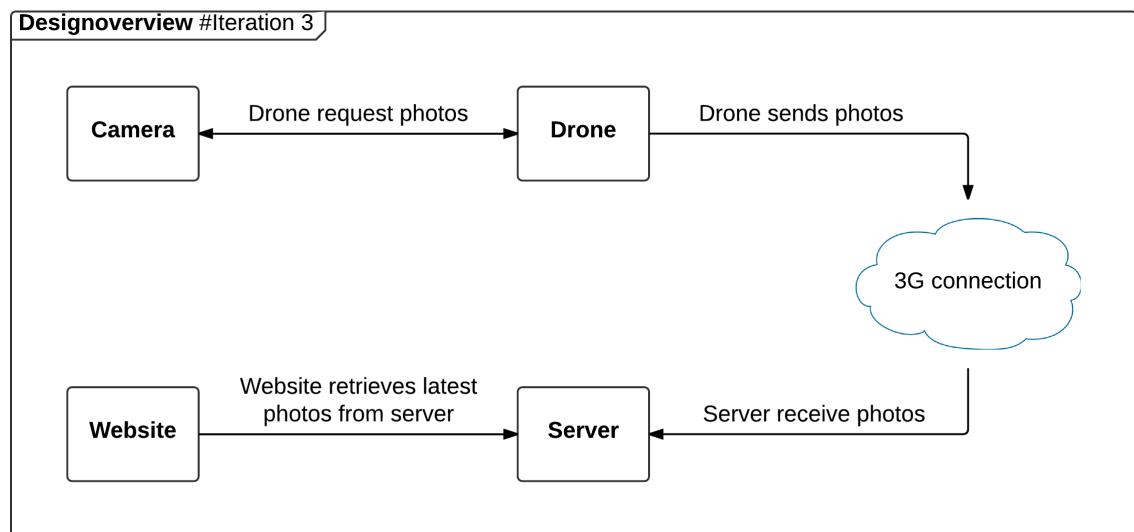
Figur 3.24: State machine #iteration 2

3.2.3 Iteration #3

I iteration 3 arbejdes med kamera blokken, som håndterer billedtagning, forsendelse af billeder og billede visning. Der monteres kamera på dronen, så den under flyvning kan tage billeder ved de GPS lokationer som bruger har defineret. Alle billeder der tages under flyvning sendes via mobilnet til server. Websiden henter automatisk de seneste billeder fra serveren og gør disse tilgængelige for bruger. Hvordan systemet er tiltænkt at bruges beskrives i user story nedenfor:

User story

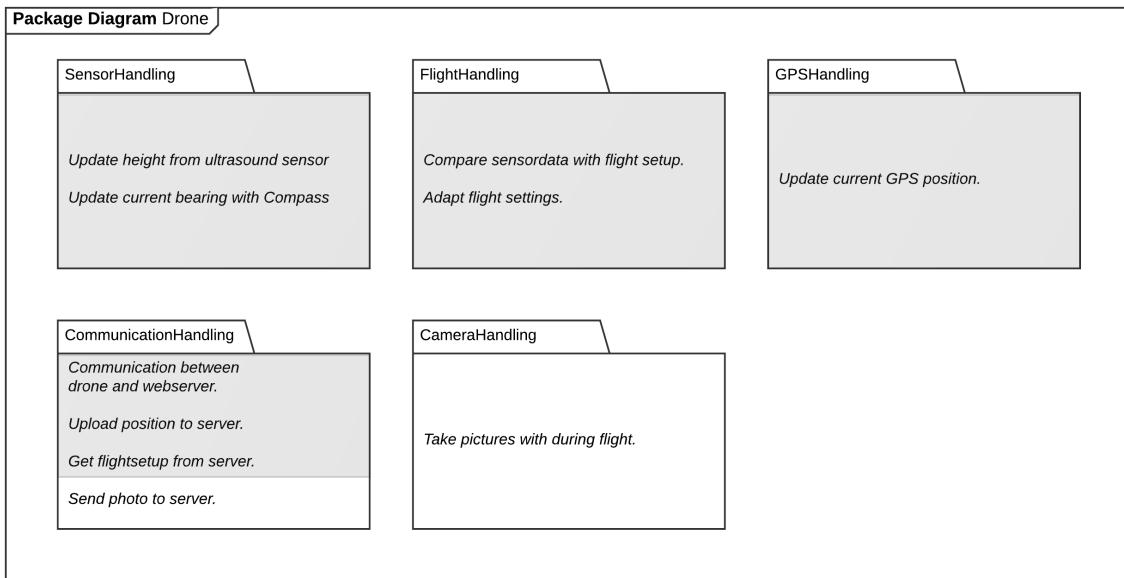
Under flyvning tager dronen billeder som sendes via 3G/GPS modulet til serveren. Websiden kontrollerer løbende hvilke billeder der er findes på serveren og gør disse tilgængelig for bruger. Login er påkrævet før bruger kan få lov at se information fra forskellige flyvninger.



Figur 3.25: Designoverview #iteration 3

Pakkediagram drone

I dette afsnit vises pakkediagram tilhørende drone. Hver pakke i pakkediagrammet består af en eller flere klasser, der med stort samspil udfører opgaver indenfor et fælles ansvarsområde. På hver pakke findes en lille beskrivelse, der tydeliggør pakkens ansvarsområde.



Figur 3.26: Pakkediagram drone

SensorHandling

Pakken er ansvarlig for indsamling af sensor data. I denne iteration skal pakken bruges til aflæsning af højdemåler og kompasset på flight control boardet.

FlightHandling

Pakkens ansvar er kontrol og styring af drone under flyvning. Ved sammenligning af sensor data og data fra flyveopsætning tilpasses flyvehøjde, orientering mm

GPSHandling

Pakkens ansvar er håndtering af GPS. Dels er pakken ansvarlig for opstart og initiering af GPS, og desuden bruges pakken hver gang dronens nuværende GPS position skal opdateres.

CommunicationHandling

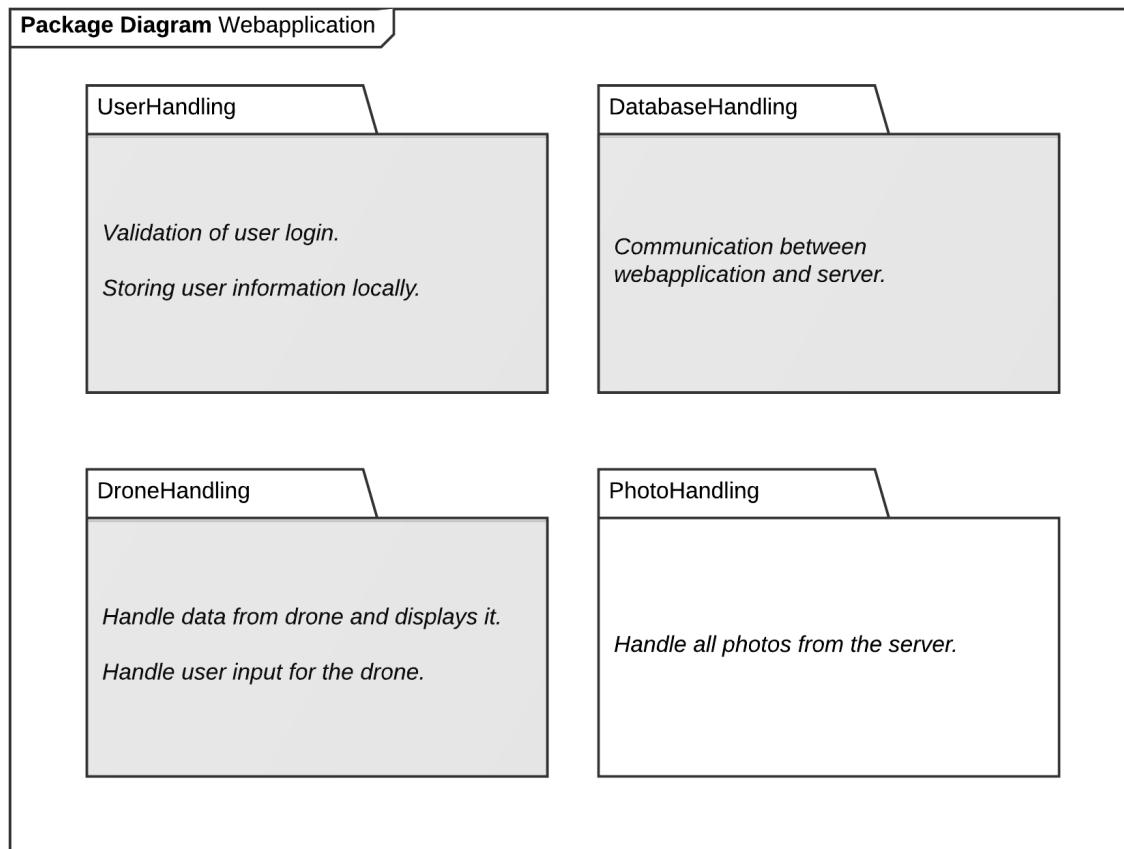
Pakkens ansvar er kommunikation imellem drone og server. Efter denne iteration skal dronen kunne hente flyveopsætninger fra server, sende sin nuværende GPS position til server og sende billeder til server.

CameraHandling

Pakkens ansvar er håndtering af kamera. Pakken bruges til at starte, initierer, bruge og slukke kameraet.

Pakkediagram webapplikation

I dette afsnit vises pakkediagram tilhørende webapplikation. Hver pakke i pakkediagrammet består af en eller flere klasser, der med stort samspil udfører opgaver indenfor et fælles ansvarsområde. På hver pakke findes en lille beskrivelse, der tydeliggør pakkens ansvarsområde.



Figur 3.27: Pakkediagram webapplikation

UserHandling

Pakkens ansvar er validering af login/log ud på webapplikationen. Pakken har også ansvaret for at hente og gemme data om brugeren.

DatabaseHandling

Pakkens ansvar er kommunikation imellem databasen og serveren.

DroneHandling

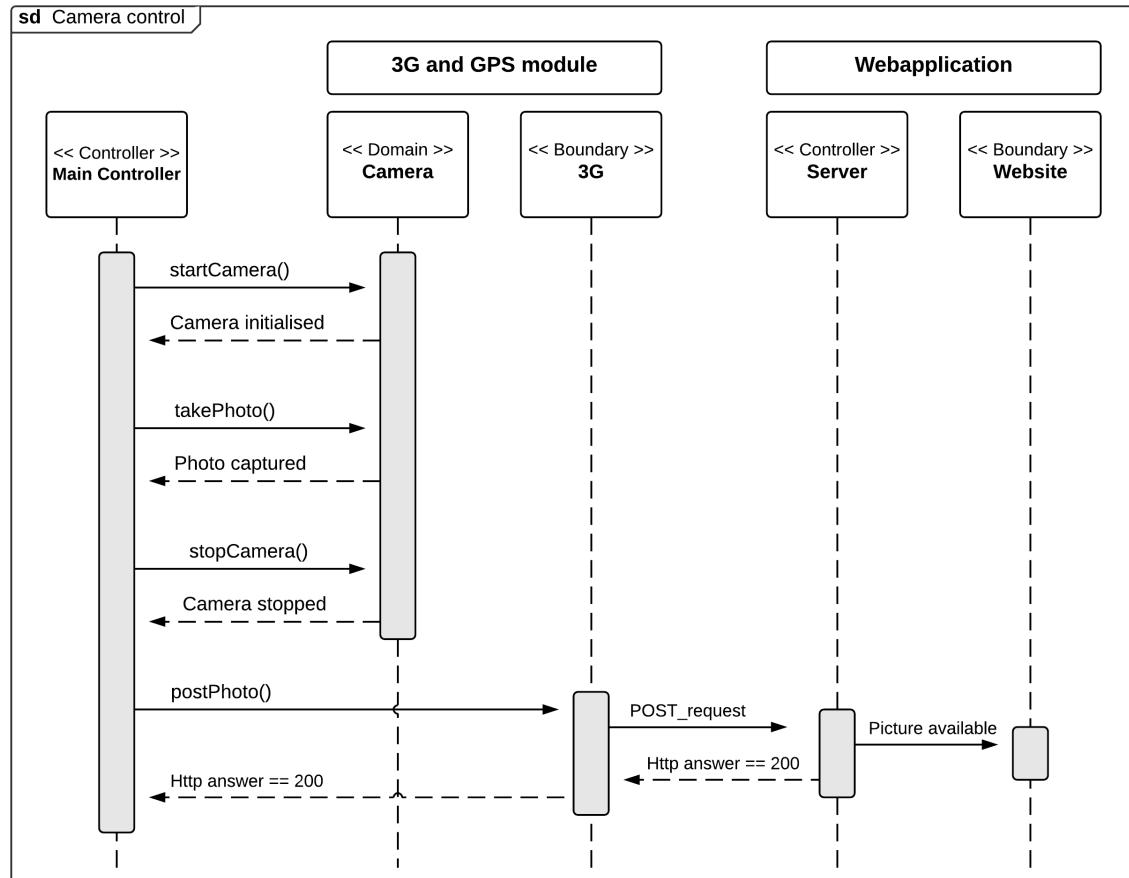
Pakken er ansvarslig for håndtering af events, waypoints og forskellige droner.

PhotoHandling

Pakkens ansvar er at håndtere de billeder dronen har taget under flyvning, præsentere dem for brugeren og sende dem videre i systemet.

Sekvensdiagram

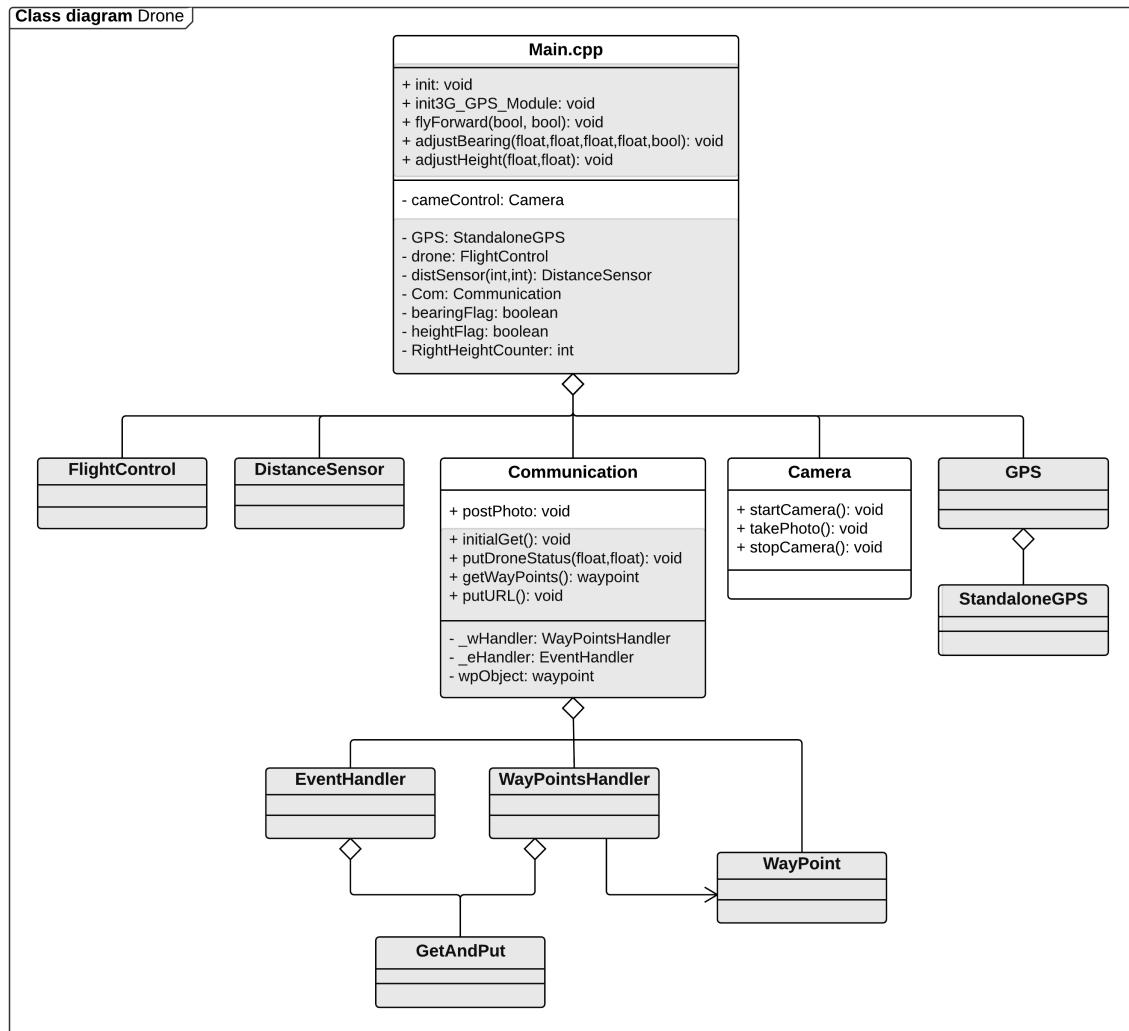
På sekvensdiagrammet på figur 3.28, vises hvilke klasser der indgår og bruges i tredje iteration. På sekvensdiagrammet vises det, hvordan kamera delen af 3G/GPS modulet håndteres og hvordan nye billeder sendes via en post request til serveren.



Figur 3.28: Sekvensdiagram #iteration 3

Klassediagram drone

Figur 3.30 viser et klassediagram tilhørende iteration 3. Klassediagrammet viser foruden main.cpp filen den nye Camera klasse og de nye metoder i Communication klassen.



Figur 3.29: Klassediagram #iteration 3

Camera

Camera klassen er ansvarlig for at tage billeder og sende dem videre i systemet.

FlightControl

FlightControl klassen står for alt styring af dronen. Bla. står FlightControl for kalibrering og styring af motorerne, samt aflæsning af højdesensor, GPS og kompas.

DistanceSensor

DistanceSensor klassen bruges til at kontrollere de sensorer der er monteret på dronen. DistanceSensor klassen bruges udelukkende til kontrol af sensore til højdemåling.

WayPointsHandler

WayPointsHandler klassen håndterer de waypoints der hentes ned fra server. Klassen tager de hente waypoints og gør dem tilgængelige for main.cpp. WayPointsHandleren gør brug af set-metoder fra WayPoint klassen.

WayPoint

WayPoint klassen bruges til at hente waypoints.

Main.cpp

Main.cpp filen bruges til at sætte arduino board korrekt op, bla. sættes baudrate på de forskellige serielle forbindelser. Desuden bruges Main.cpp til at kalde og eksekverer forskellige klasse, objekter og funktioner.

GPS

GPS klassen er implementeret som en abstract klasse. Init og updateGPSPosition er lavet som virtuelle metoder, hvilket betyder de skal implementeres i alle afledte klasser. GPS klassen er lavet fordi 3G/GPS modulet kunne bruges i 3 forskellige GPS modes.

StandaloneGPS

Denne klasse er ansvarlig for al kommunikation med GPS'en når standalone mode er valgt.

GetAndPut

GetAndPut klassen er den klasse der er tættest på hardwaren. Klassen indeholder http metoder der bruges til kommunikation mellem dronen og serveren.

Communication

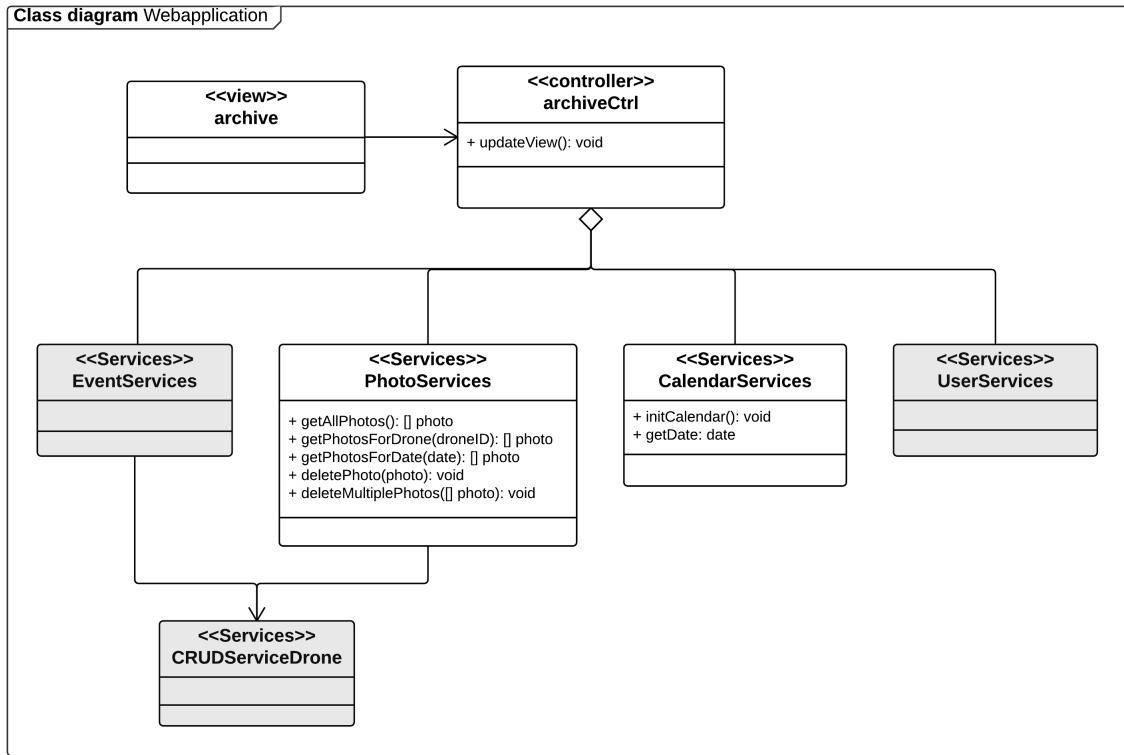
Communication klassen styrer alt der har med 3G kommunikation at gøre.

EventHandler

EventHandler klassen håndterer Events, og bruges som bindelede mellem communication- og GetAndPut klassen. EventHandleren sorterer eventID'et fra data der modtages og returnerer værdien af eventID til communication klassen.

Klassediagram webapplikation

På figur 3.30 vises klassediagrammet tilhørende iteration 3.



Figur 3.30: Klassediagram #iteration 3

archive

Denne klasse håndterer view'et tilhørende iteration tre. Sammen med archiveCtrl skaber archive klassen det bruger ser i sin browser.

archiveCtrl

ArchiveCtrl klassen er controller klassen til iteration tre. Det er den eneste klasse der har direkte forbindelse til view klassen. ArchiveCtrl klassen deler hukommelse med view'et igennem two-way-binding med scopes.

EventServices

Denne klasse indeholder logikken om event håndtering. Klassen bruges til at hente en event liste, et enkelt event for en givet drone og sende et nyoprettet event til serveren via CRUDServiceDrone klassen.

PhotoServices

Denne klasse indeholder logikken om foto håndtering. Klassen bruges til at hente fotos via CRUDServiceDrone klassen, derudover har klassen ansvaret for at slette uønskede billeder.

CalendarServices

Denne klasse indeholder logikken for kalenderen, klassen bruges til at oprette og navigere i kalenderen. Desuden bruges klassen til at fortælle hvilken dato der er valgt, når billeder fra tidligere flyvning ønskes vist.

UserServices

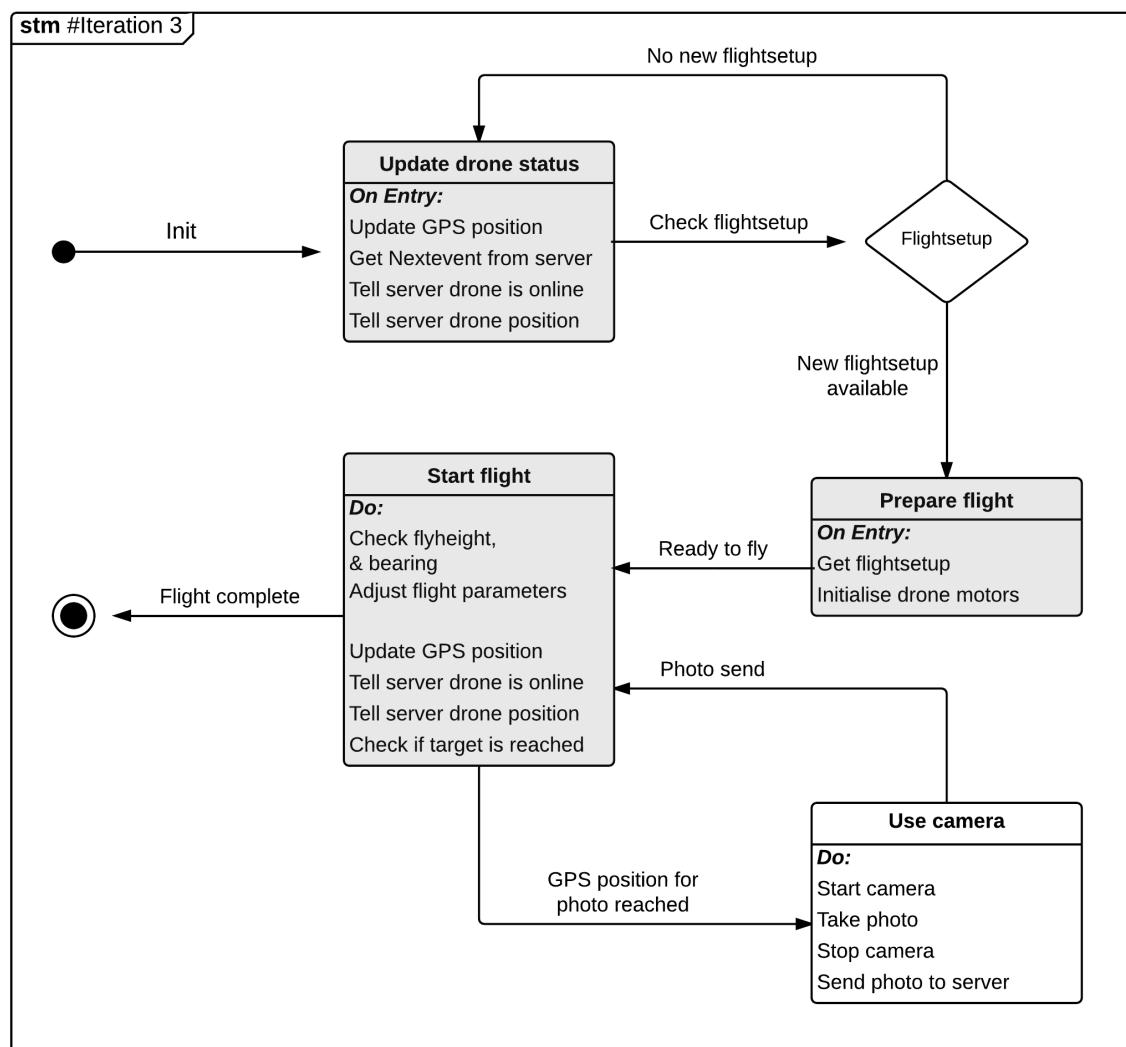
UserServices bruges til at gemme information om hvilken bruger der er logget ind i systemet.

CRUDServicesDrone

Denne klasse bruges som bindeledd imellem server og webapplikation, og indeholder logik til serveren. CRUDServicesDrone bruges når der sendes Get, Put og Post til serveren.

State machine diagram

I state machine diagrammet på figur 3.31, vises hvordan flowet i systemet er. Der er tilføjet en enkelt state, der håndterer alt det der har med billeder at gøre.



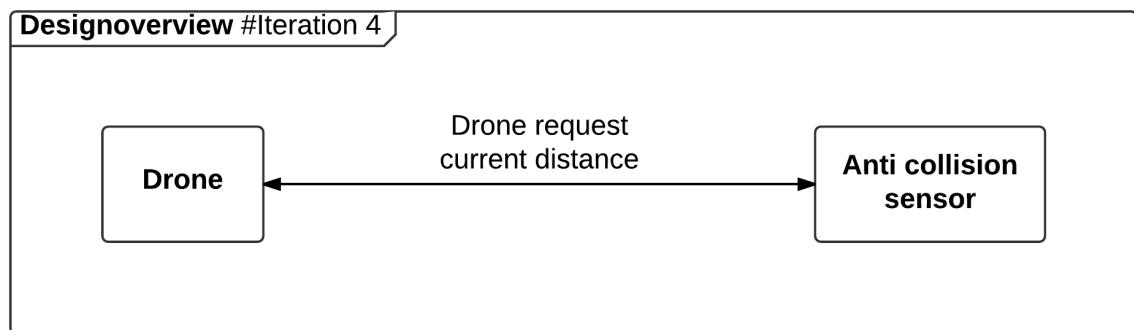
Figur 3.31: State machine #iteration 3

3.2.4 Iteration #4

I iteration 4 ønskes det at udvide systemet med anti kollision. Inden tilføjelsen af anti kollision kan dronen udelukkende flyve i områder uden forhindringer. Men med tilføjelsen af anti kollision vil det være muligt at flyve med dronen i normale områder med diverse forhindringer.

User story

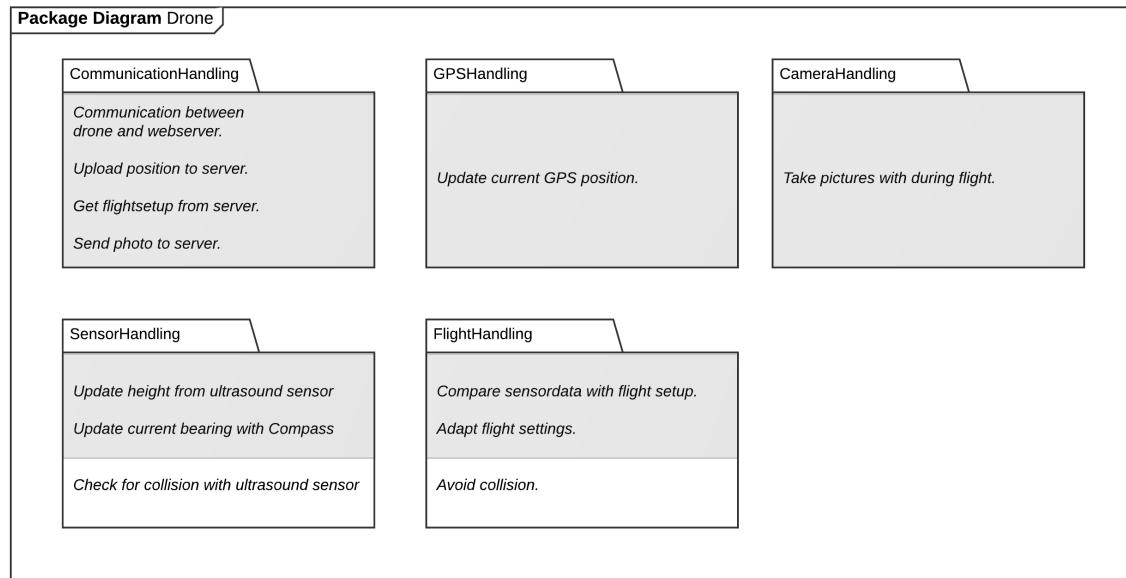
Under flyvning kontrolleres det løbende hvorvidt der er et objekt foran dronen. Hvis der detekteres et objekt foran dronen, skal dronen holde om med at flyve fremad. I stedet skal dronen dreje horisontalt indtil der ikke længere er et objekt foran den. Når der er frit foran dronen fortsættes den afbrudte flyvning. Bruger af systemet har ingen direkte kontakt med anti kollisionssystemet. For bruger ses anti kollision blot som en feature, som gør dronen i stand til at flyve uden at kolidere med objekter foran sig.



Figur 3.32: Design overview #iteration 4

Pakkediagram drone

I dette afsnit vises pakkediagram tilhørende drone. Hver pakke i pakkediagrammet består af en eller flere klasser, der med stort samspil udfører opgaver indenfor et fælles ansvarsområde. På hver pakke findes en lille beskrivelse, der tydeliggør pakkens ansvarsområde.



Figur 3.33: Pakkediagram drone

SensorHandling

Pakken er ansvarlig for indsamling af sensor data. I denne iteration skal pakken bruges til aflæsning af højdemåler, kompas fra flight control boardet og anti kollision sensor.

FlightHandling

Pakkens ansvar er kontrol og styring af drone under flyvning. Ved sammenligning af sensor data og data fra flyveopsætning tilpasses flyvehøjde, orientering mm. I denne iteration tilføjes funktionalitet som skal få dronen til at undgå kollision.

GPSHandling

Pakkens ansvar er håndtering af GPS. Dels er pakken ansvarlig for opstart og initiering af GPS. Pakken bruges hver gang dronens nuværende GPS position skal opdateres.

CommunicationHandling

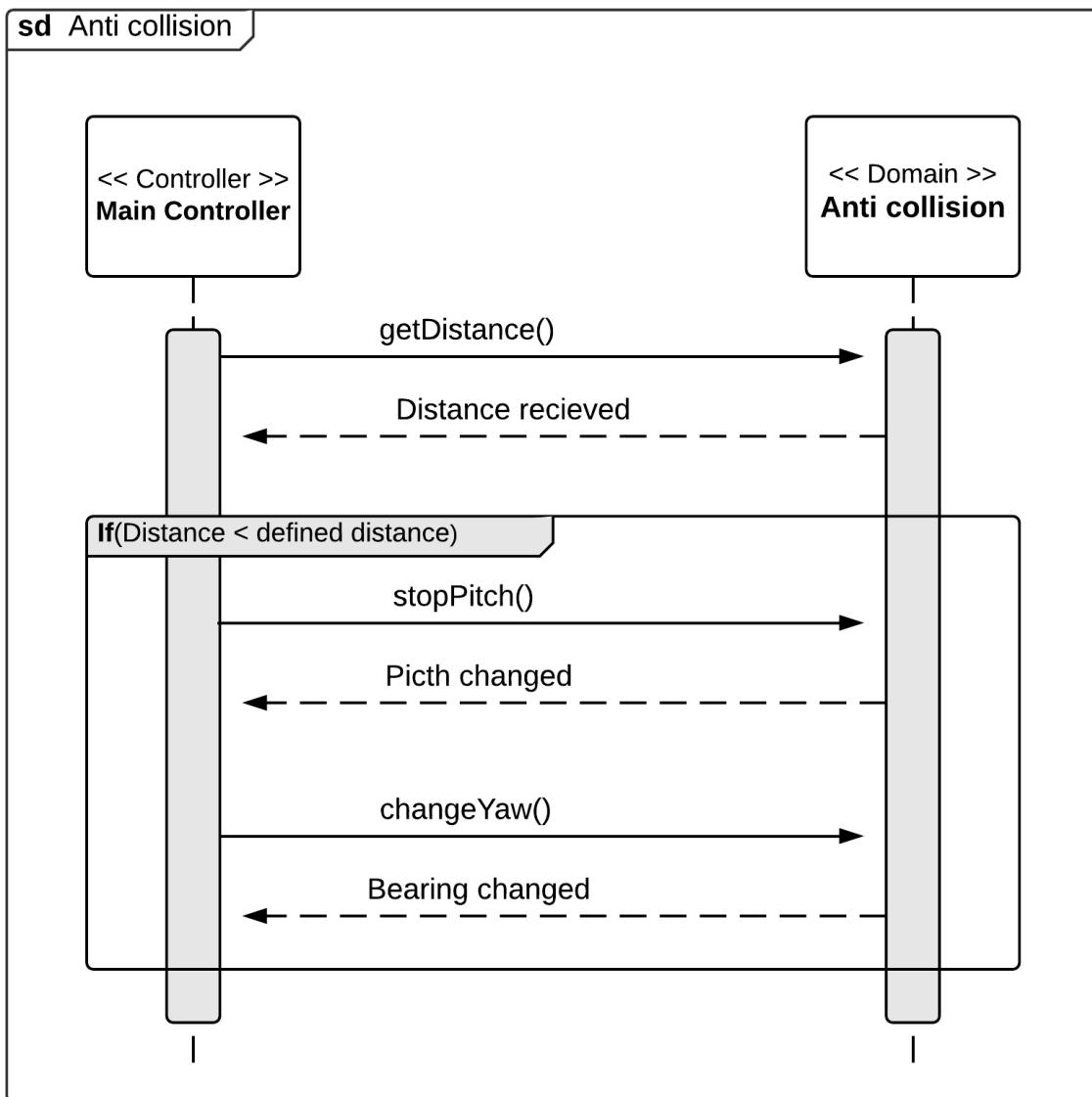
Pakkens ansvar er kommunikation imellem drone og server. Efter denne iteration skal dronen kunne hente flyveopsætninger fra server, sende sin nuværende GPS position til server og sende billeder til server.

CameraHandling

Pakkens ansvar er håndtering af kamera. Pakken bruges til at starte, initierer, bruge og slukke kameraet.

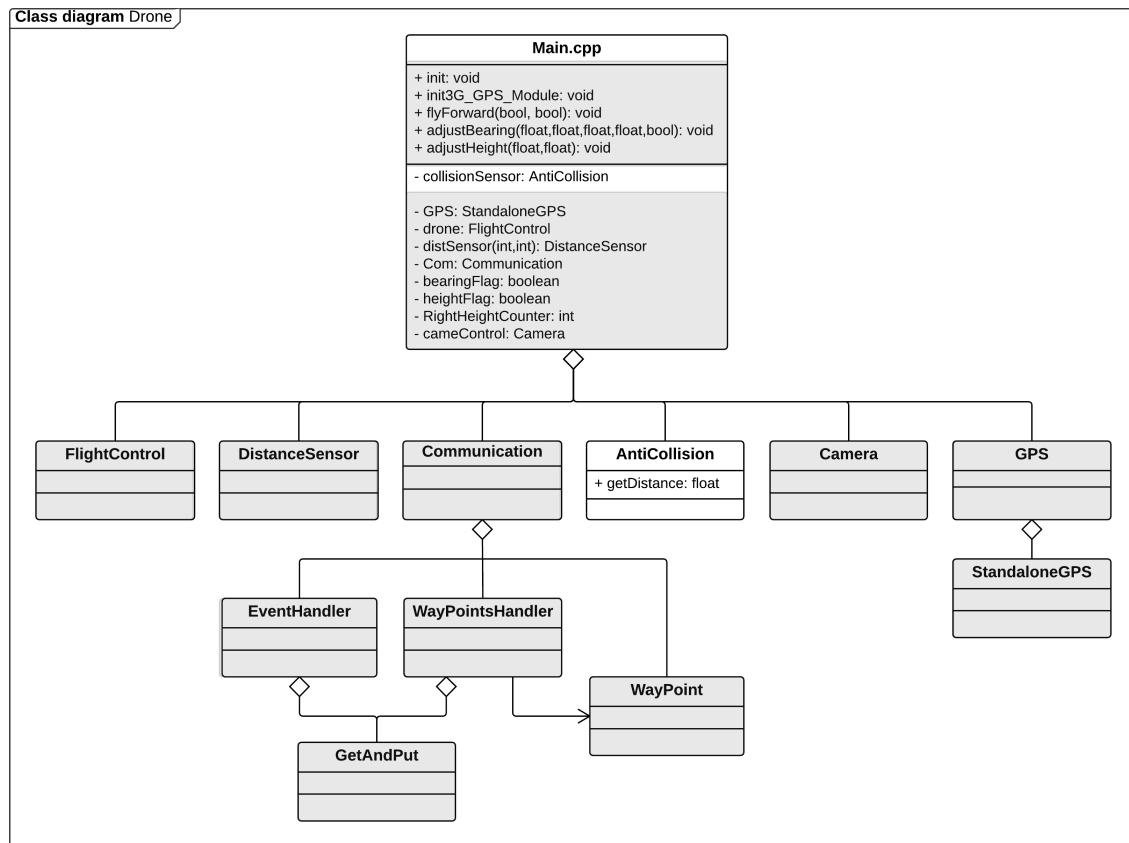
Sekvens diagram

På sekvensdiagrammet vises de klasser der indgår og bruges i fjerde iteration. På diagrammet vises det hvordan afstanden til eventuelle objekter foran dronen kontrolleres. Hvis afstanden til et objekt er under den definerede kollisionsgrænse stoppes dronens fremdrift og der ændres orientering.



Figur 3.34: Sekvens diagram #iteration 4

Klassediagram drone



Figur 3.35: Klasse diagram #iteration 4

AntiCollision

AntiCollision klassen fungerer ligesom DistanceSensor. Klassen bruges dog til at kontrollere anti kollisions sensorer der er monteret på dronen.

Camera

Camera klassen er ansvarlig for at tage billeder og sende dem videre i systemet.

FlightControl

FlightControl klassen står for alt styring af dronen. Bla. står FlightControl for kalibrering og styring af motorerne, samt aflæsning af højdesensor, GPS og kompas.

DistanceSensor

DistanceSensor klassen bruges til at kontrollere de sensorer der er monteret på dronen. DistanceSensor klassen bruges udelukkende til kontrol af sensore til højdemåling.

WayPointsHandler

WayPointsHandler klassen håndterer de waypoints der hentes ned fra server. Klassen tager de hentede waypoints og gør dem tilgængelige for main.cpp. WayPointsHandleren gør brug af set-metoder fra WayPoint klassen.

WayPoint

WayPoint klassen bruges til at hente waypoints.

Main.cpp

Main.cpp filen bruges til at sætte arduino board korrekt op, bla. sættes baudrate på de forskellige serielle forbindelser. Desuden bruges Main.cpp til at kalde og eksekverer forskellige klasse, objekter og funktioner.

GPS

GPS klassen er implementeret som en abstract klasse. Init og updateGPSPosition er lavet som virtuelle metoder, hvilket betyder de skal implementeres i alle afledte klasser. GPS klassen er lavet fordi 3G/GPS modulet kunne bruges i 3 forskellige GPS modes.

StandaloneGPS

Denne klasse er ansvarlig for al kommunikation med GPS'en når standalone mode er valgt.

GetAndPut

GetAndPut klassen er den klasse der er tættest på hardwaren. Klassen indeholder http metoder der bruges til kommunikation mellem dronen og serveren.

Communication

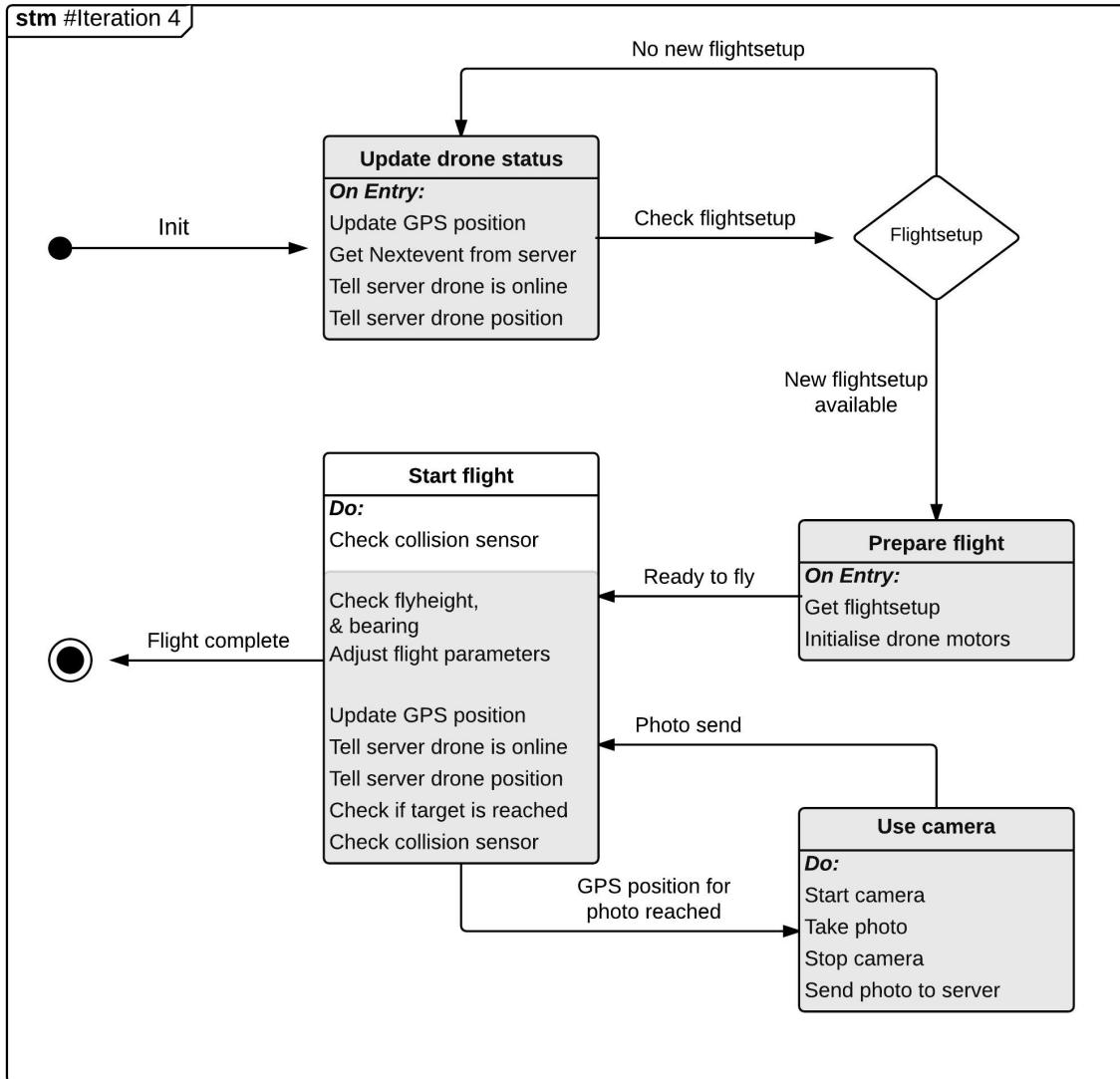
Communication klassen styrer alt der har med 3G kommunikation at gøre.

EventHandler

EventHandler klassen håndterer Events, og bruges som bindeled mellem communication- og GetAndPut klassen. EventHandleren sorterer eventID'et fra data der modtages og returnerer værdien af eventID til communication klassen.

State machine diagram

I state machine diagrammet på figur 3.36, vises de states der eksisterer i iteration 4. Desuden vises hvordan flowet imellem de to states ser ud.



Figur 3.36: Statemachine #iteration 4

3.3 Process view

Ingen af systemets centrale dele gør brug af tråde. Men systemet er opbygget således at afvikling af drone, webapplikation og server kan betragtes som 3 samtidige processer. Alle 3 dele bruges sideløbende i runtime og imellem dem flyder løbende data.

For systemets bruger ser det ud som om al information går direkte fra webapplikation til drone eller omvendt. Men reelt set er webapplikation og drone aldrig i direkte kontakt med hinanden. I stedet går al kommunikation til og fra serveren, som fungerer som et kommunikationslag.



Figur 3.37: Process view

Server

Serveren er passiv, hvilket betyder den aldrig tager initiativ til udveksling af data. Serveren foretager ingenting på egen hånd, den står i stedet og venter på at blive igangsat af drone eller webapplikation.

Drone

I drone processen håndteres al styring af drone og kommunikation mellem drone og server. Når dronen er tændt og 3G/GPS modulet initialiseret, opdaterer dronen med få minutters interval egen GPS position for efterfølgende at sende serveren information om positionen. Desuden kontrollerer dronen løbende om der er en ny flyveopsætning tilgængelig på serveren. Hvis der er en ny flyveopsætning tilgængelig hentes den, og en ny flyvning påbegyndes.

Webapplikation

Mellem server og webapplikation er der lavet en socket connection. Det betyder at indholdet af webapplikationen opdateres hver gang der kommer nyt indhold på serveren. Desuden bruges webapplikationen når systemets bruger ønsker at lave nye flyveopsætninger eller opdatere de nuværende.

Der er lagt megen energi i at designe og bygge systemet på en vis der tillader udvidelser og tilføjelser. Dels kan server nemt tilgås af flere forskellige webapplikationer og desuden kan webapplikationen håndtere flere droner og klienter samtidig.

3.3.1 Kommunikations timing

For at kommunikationen mellem drone og server kan fungere optimalt, er der udformet kommunikations timings tabeller. Timings tabellerne foreskrive hvilke request drone skal sende til server og hvilke svar der forventes fra server.

Nedenfor forklares kort hvordan drone kommunikerer med webapplikationen i forbindelse med en autonom flyvning.

1. Dronen tændes.
2. Handling i tabel 3.1 udføres og gentages efterfølgende hvert 5. minut.
3. Handling i tabel 3.2 udføres og gentages hvert 45. sekund indtil der returneres en anden værdi end 0 på NextEvent GET requestet.
4. Handling i tabel 3.3 udføres
5. Drone påbegynder flyvning

Enhed	HTTP Request	Data	Handling
Drone	PUT	Online og Location	Opdatere sin position og sætter sig online
Server	Response	200 OK	Giver ok svar

Tabel 3.1: Kommunikation drone og server - step 1

Enhed	HTTP Request	Data	Handling
Drone	GET	NextEvent	Efterspørg nextevent værdig
Server	Response	NextEvent	Retunere drone data med nextevent

Tabel 3.2: Kommunikation imellem drone og server - step 2

Enhed	HTTP Request	Data	Handling
Drone	GET	Waypoint	Dronen henter waypoints tilhørende event
Server	Response	Waypoints	Retunere waypoints tilhørende event
Drone	PUT	NextEvent = 0	Dronen ændre sin NextEvent værdig til 0
Server	Response	200 OK	Giver ok svar

Tabel 3.3: Kommunikation imellem drone og server - step 3

3.4 Data view

I data viewet beskrives layoutet for databasen. Viewet har til formål at skabe overblik over databasen, hvordan den er designet og hvordan den tilgåes. Databasen spiller en væsentlig rolle da dronen og clienten bruger databasen til at udveksle data med hinanden. Først i afsnittet vil det overordnet design blive beskrevet og efter det vil hver enkelt tabel blive beskrevet mere i detaljer.

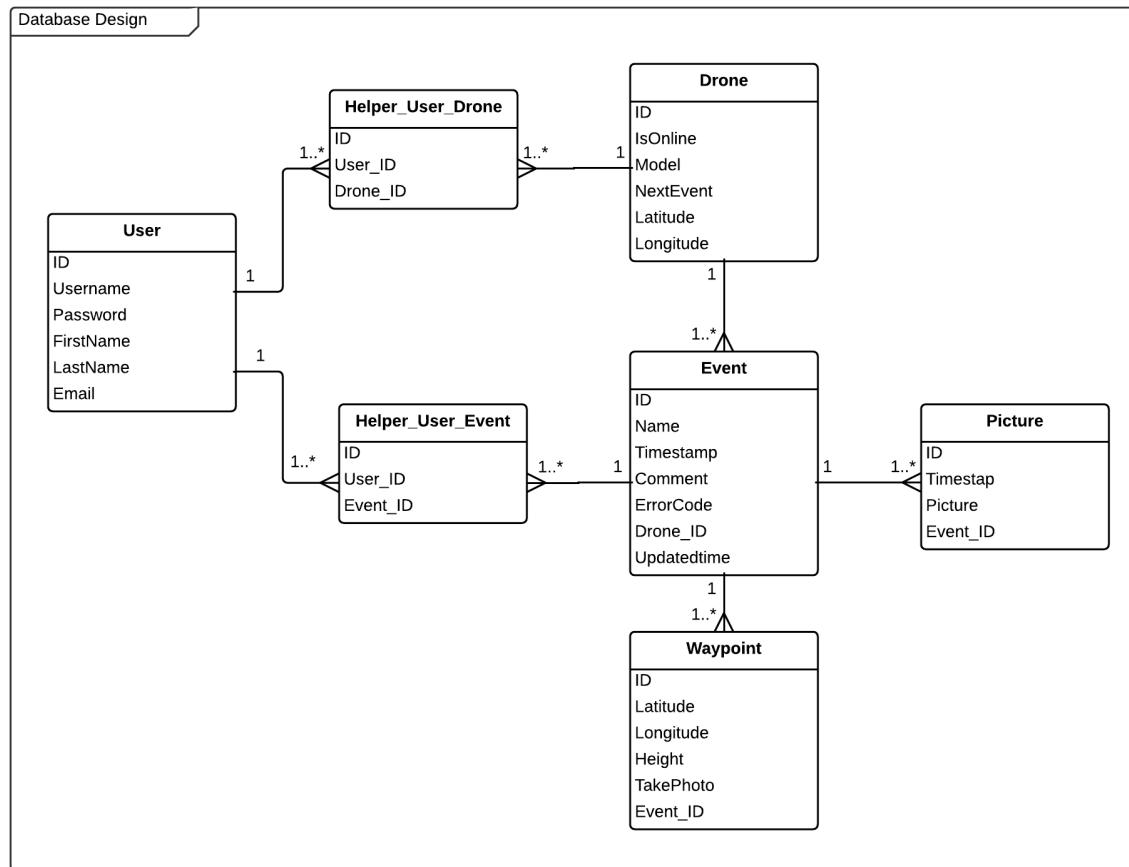
3.4.1 Database design

Mission

Databasen har til formål at vise systemets brugere hvilke droner der er til rådighed. Blandt andet skal det være muligt for bruger at se hvilke droner der er ude og flyve og hvilke der ikke er aktive. Derudover tracker databasen hvilke events der foregår med hver drone og danner historik over hvert event. Eksempler på events kan være information om flyveruten og om der tages billeder på flyveturen.

Design

På figur 3.38 ses det overordnede designet af databasen. Som beskrevet i foranalysen er det valgt at gøre brug af en SQLite database. Alt kommunikation til og fra databasen vil fungere igennem et database API. Database API'et er et REST API som er udviklet ved brug af Django og Django REST frameworket. Ved kombination af disse Django og Django REST fås en databasen som er "indkapslet" og kun kan tilgås gennem API'et. Kommunikationen til og fra API'et foregår ved sende eller modtage JSON filer til API'ets endpoints. Hjælpe tabellerne som ses på figur 3.38 bliver automatisk oprettet af django frameworket, hvilket betyder de senere ikke er til at finde i koden for databasen. Hjælpe tabellerne er dog tegnet med på diagrammet, da de eksistere i systemet kan være med til at forstærke forståelsen af databasen.

**Figur 3.38:** Database design

Tilgængelige endpoints

De tilgængelige endpoint i API'et er følgende:

Endpoint:	<i>api/waypoints/</i>
Allows:	POST, OPTIONS, GET
Content-Type:	application/json
Indehold:	Giver adgang til waypoint tabellen, med mulighed for at tilføje nye waypoints
Kommentar:	Det er muligt at tilføje "?format=json" til URL'en for at få dataen i ren json format

Tabel 3.4: Waypoint endpoint

Endpoint:	<i>api/users/</i>
Allows:	OPTIONS, GET
Content-Type:	application/json
Indehold:	Giver adgang til user tabellen
Kommentar:	Det er muligt at tilføje "?format=json" til URL'en for at få dataen i ren json format

Tabel 3.5: User endpoint

Endpoint:	<i>api/pictures/</i>
Allows:	POST, OPTIONS, GET
Content-Type:	application/json
Indehold:	Giver adgang til picture tabellen, med mulighed for at tilføje nye billeder
Kommentar:	Det er muligt at tilføje "?format=json" til URL'en for at få dataen i ren json format

Tabel 3.6: Picture endpoint

Endpoint:	<i>api/events/</i>
Allows:	POST, PUT, OPTIONS, GET
Content-Type:	application/json
Indehold:	Giver adgang til event tabellen, med mulighed for at oprette nye events og opdatere dem
Kommentar:	Det er muligt at tilføje "?format=json" til URL'en for at få dataen i ren json format

Tabel 3.7: Event endpoint

Endpoint:	<i>api/drones/</i>
Allows:	POST, OPTIONS, GET
Content-Type:	application/json
Indehold:	Giver adgang til drone tabellen, med mulighed for at tilføje nye droner
Kommentar:	Det er muligt at tilføje "?format=json" til URL'en for at få dataen i ren json format

Tabel 3.8: Drones endpoint

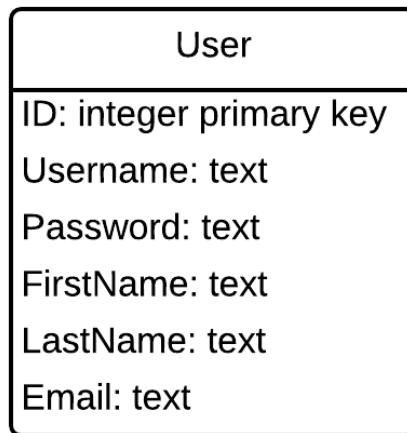
Endpoint:	<i>api/drones/#/</i>
Allows:	PUT, OPTIONS, GET
Content-Type:	application/json
Indehold:	Giver adgang til en record i drone tabellen, med mulighed for at opdatere droner
Kommentar:	Det er muligt at tilføje "?format=json" til URL'en for at få dataen i ren json format

Tabel 3.9: Single drone endpoint

3.4.2 Detaljeret database beskrivelse

User table

Tabellen indeholder data om systemets brugere. Tabellen gør det via foreing keys muligt for systemets brugere at tilgå droner, events og de gemte flyruter i systemet.



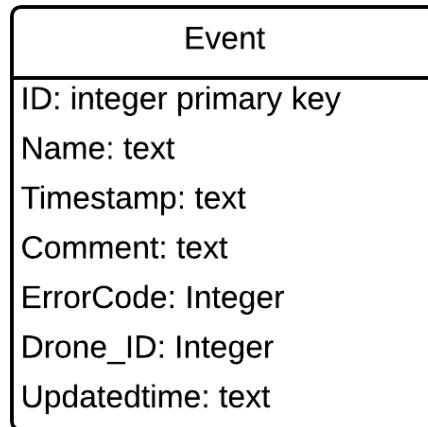
Figur 3.39: User table

Formål	At holde data om brugere i systemet, samt tjekke om brugere eksisterer, når de forsøger at logge på webapplikation.
Forbindelser	Tabellen har en foreing key til at hjælpe tabellerne: Helper User Drone og Helper User Event.
Attributter	<ul style="list-style-type: none"> • ID: Primary key. • Username: Brugernavnet til systemet. Max length: 50 char • Password: Brugerens kode. Max length: 50 char • FirstName: Brugers fornavn. Max length: 50 char • LastName: Brugerens efternavn. Max length: 50 char • Email: Brugerens email adresse.

Tabel 3.10: User table

Event table

Tabellen indeholder data om events i systemet. Tabellen fungere som omdrejningspunkt i databasen. Droner, billeder og waypoints kan tilføjes til et event.



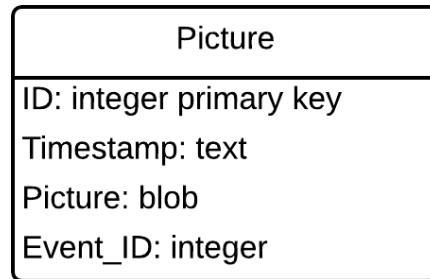
Figur 3.40: Event table

Formål	Holde data om events i systemet.
Forbindelser	Tabellen har en foreing key til user tabellen med en mange til mange relation. Drone-, waypoint- og picture-tabellen har foreing keys til event tabellen.
Attributter	<ul style="list-style-type: none"> • ID: Primary key. • Name: Navnet på det givet event. Max length: 100 char • Timestamp: Timestamp for oprettelse: Datefield • Updated: Timestamp for opdaterede: Datefield • Comment: Kommentar til givet event. Max length: 100 char • ErrorCode: Fejlkode hvis fejl opstår under flyvning: Integer • UserId: Mange til mange relation til user tabellen: Foreingkey

Tabel 3.11: Event table

Picture table

Tabellen indeholder billeder og informationer om de billeder taget af dronen. Billeder gemmes i en blob type, som er binær-data i databasen.



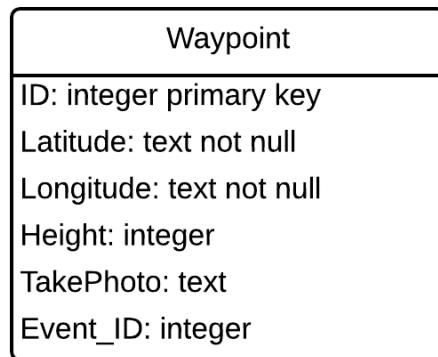
Figur 3.41: Picture table

Formål	Holde data om billeder i systemet og selve billederne.
Forbindelser	Picture tabellen har en foreing key til event tabellen.
Attributter	<ul style="list-style-type: none"> • ID: Primary key. • Name: Navnet på det givet event. Max length: 100 char • Timestamp: Timestamp for oprettelse: Datefield • Picture: blob: Udefinerbar størrelse binær data • Event ID: Relation til event tabellen: Foreing key

Tabel 3.12: Picture table

Waypoint table

Tabellen indeholder alle waypoints som brugeren har oprettet i systemet. Hvert waypoint i tabellen indeholder information om gps-koordinatet, højden og om der skal tages et billede ved lokationen eller ej.



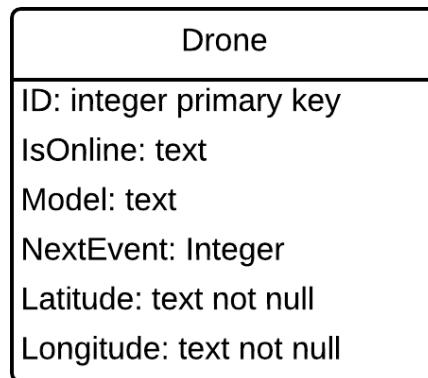
Figur 3.42: Waypoint table

Formål	Holde data om waypoints i systemet.
Forbindelser	Waypoint tabellen har en foreing key til event tabellen.
Attributter	<ul style="list-style-type: none"> • ID: Primary key. • Latitude: Required text felt. Max length: 100 char • Longitude: Required text felt. Max length: 100 char • Height: Integer • Event ID: Relation til event tabellen: Foreing key

Tabel 3.13: Waypoint table

Drone table

Tabellen indeholder alle droner som er oprettet i systemet. Bruger har mulighed for at tilføje flere droner.



Figur 3.43: Drone table

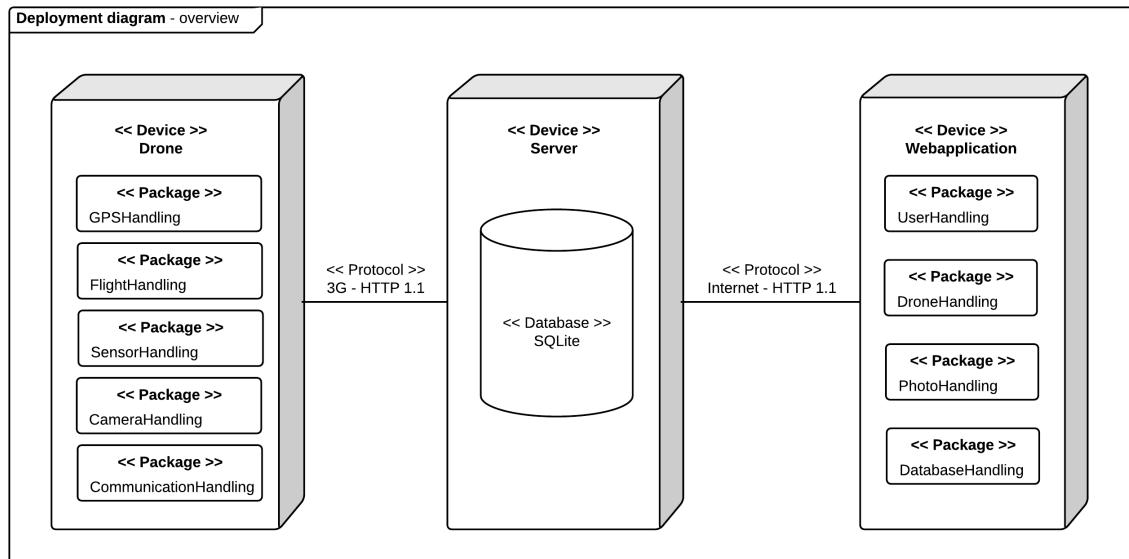
Formål	Holde information om hver drone i systemet.
Forbindelser	Drone tabellen har en foreing key til event tabellen.
Attributter	<ul style="list-style-type: none"> • ID: Primary key. • IsOnline: Status for dronen: Boolean værdig • Model: Modeltype: Text field: Max length 50 char • NextEvent: Events for dronen: Integer field • Latitude: Dronens GPS lokation: Max length 100 char • Longitude: Dronens GPS lokation: Max length 100 char

Tabel 3.14: Drone table

3.5 Deployment view

I dette afsnit beskrives hvilke softwareklasser der bruges i systemet og hvor de bruges. Desuden beskrives hvilke protokoller der er anvendt i systemet, fx. beskrives layout af meddelelser der sendes mellem drone og server.

På illustrationen figur 3.44 vises systemet opdelt i de 3 mest grundlæggende hardware dele. Ydermere vises det hvilke software pakker der bruger i hver af de 3 hardware dele.



Figur 3.44: Overordnet deployment diagram

På den følgende side forklares der om HTTP 1.1 protokollen. Indledningsvis beskrives det hvorfor HTTP protokollen bruges, dernæst beskrives det hvordan protokollen benyttes.

3.5.1 HTTP protokol

HTTP protokollen er valgt som kommunikations led.

Der er begrænsede muligheder med 3G/GPS modulet når det kommer til kommunikation. Da 3G/GPS modulet ikke understøtter alle kommunikations lag, men understøtter HTTP protokollen er denne protokol valgt som kommunikations lag. Derudover er HTTP protokollen den mest anvendte protokol til kommunikation mellem en webapplikation og server. HTTP protokol version 1.1 er valgt fremfor 1.0, da version 1.1 indeholder metoden PUT, som er en vigtig metode for systemet.

Metoderne der anvendes er GET, PUT og POST. Når en af disse metoder anvendes, kommer der et svar retur, som indeholder en række informationer. Svaret på figur 3.45 inderholder en header og en body fra GET requesten til events.

Headeren indeholder informationer om HTTP forbindelsen. Den første linje indeholder en status besked, som viser hvorvidt beskeden er blevet modtaget korrekt eller om der er sket en fejl. Ydermere inderholder headeren tidspunktet for modtagelsen, hvilket format body'en er i, tilladte metoder til kommunikation og hvilken type server der er hentet fra. Selve body'en starter på figur 3.45 efter 7e og slutter ved 0, hvor 7e er antal karakterer i body'en vist i hexadecimal. Når PUT eller POST anvendes, er det nødvendigt at sende information for headeren med. Derudover er det vigtigt at sende antal karakterer der er i body'en med. Ved PUT og POST modtages der også et svar i form af en header og en body, med de samme informationer som ved GET metoden.

```
Data received: 384
data:
http/1.1 200 ok
server: nginx/1.4.6 (ubuntu)
date: tue, 02 dec 2014 09:49:46 gmt
content-type: application/json
transfer-encoding: chunked
connection: keep-alive
vary: accept, cookie
x-frame-options: sameorigin
allow: put, options, get

7e
{"id": 1, "is_online": true, "model": "aeroquad", "next_event": 1,
"latitude": "56.172171", "longitude": "10.191837999999962"}
0
```

Figur 3.45: Header og Body for GET metoden

Yderligere information om parametrene i body'en kan findes i database viewet, kapitel 3.4.

3.6 Implementation view

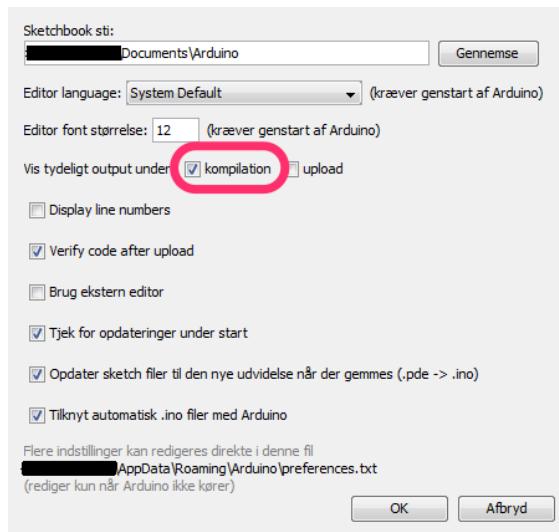
I implementation view'et beskrives elementer som uden forklaring ville være svære at forstå og bruge for udefrakommende. Indledningsvis beskrives de værktøjer der er benyttet til opsætning af dronens main controller, et Arduino 2560 board. Dernæst beskrives hvordan interne timerne på Arduino boardet er sat op til at genererer PWM signaler. Til sidst i afsnittet beskrives værktøjer der er brugt til udvikling af server og webapplikation.

3.6.1 Opsætning af udviklings IDE til Arduino

Dette afsnit har til formål at beskrive opsætningen af udviklings IDE'et til Arduino. For at kunne gennemføre guiden skal AVR's Atmel Studio 6 og Arduino 1.0 eller højere være installeret på computeren.

For at kunne compilere projekter til Arduino, kræves Arduino Core biblioteket i Atmel. Dette bibliotek bliver kun oprettet i forbindelse med kompileringen af projekter igennem Arduino IDE'et. Biblioteket skal inkluderes i Atmel Studio, derfor skal der først oprettes et Arduino projekt i Arduino IDE'et og kompileres. Dette skal dog kun gøres en gang, for at få oprettet Arduino Core biblioteket så det kan inkluderes.

Åben Arduino IDE'et, gå ind i indstillinger og sæt flueben ved: Vis tydeligt output under kompilation.



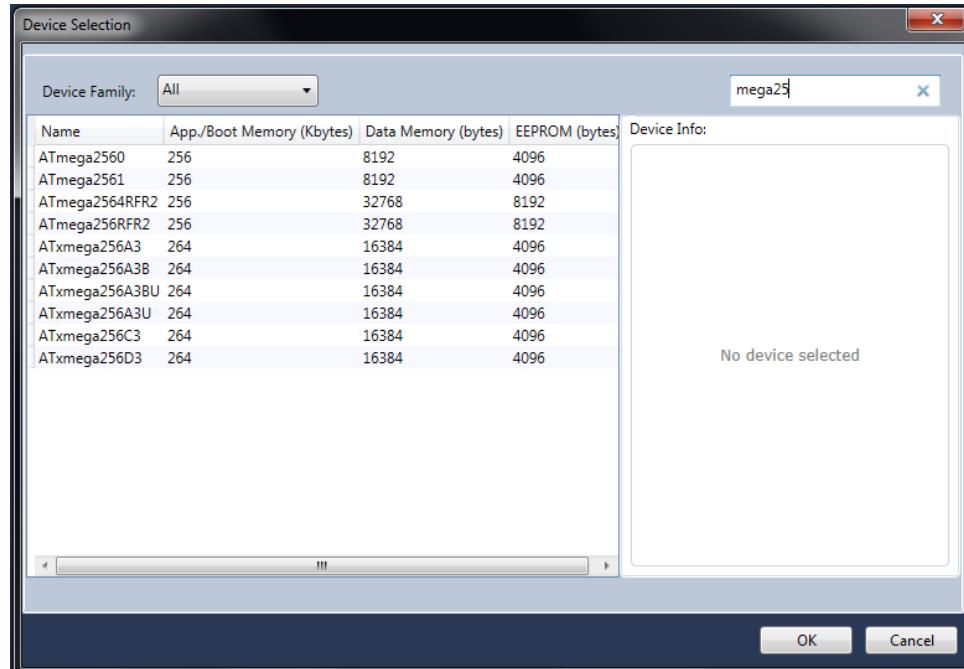
Figur 3.46: Indstillings vindue Arduino

I Arduino vælges et tilfældigt projekt, den rigtige arduino vælges under Værktøjer -> kort. Hvorefter skal koden kompileres. I vinduet i bunden vises en masse information, bla. stien for hvor core filen ligger. På figur 3.47 ses et eksempel af stien til core filen, i det røde skal udviklerens computer navn stå.

```
C:\Users\...\AppData\Local\Temp\build2556367648181810842
```

Figur 3.47: Stien til arduino core fil

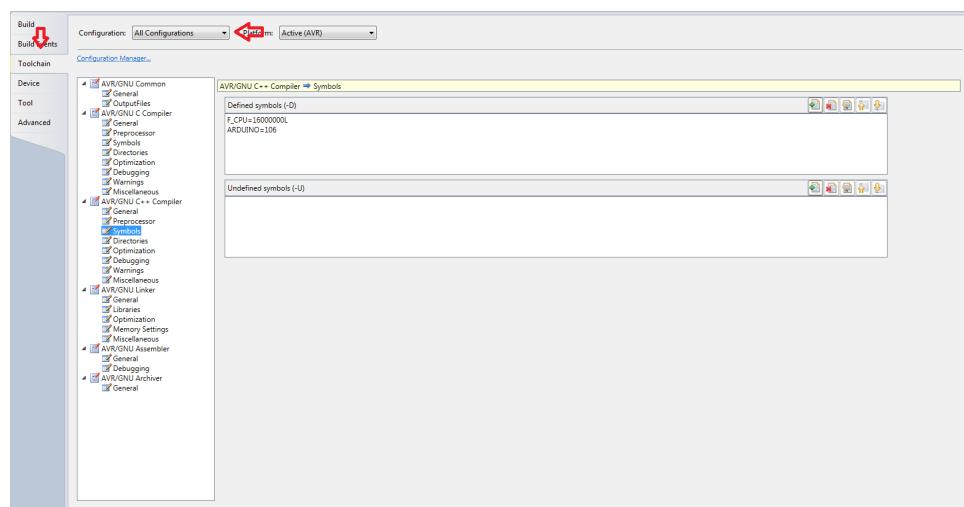
Inde i mappen ligger en core fil, denne fil skal kopieres over i Atmel Studio's workspace. Som standard ligger workspacet i "C:\Users\computer navn\Documents\Atmel Studio". Inde i Atmels workspace oprettes en mappe: "ArduinoCore"eller lignende. Core filen kopieres til mappen og navnet ændres til libcore. Ved opstart af Atmel, vælges hvilken type chip arduino bruger. Til projektet er der anvendt en Mega2560.



Figur 3.48: Atmel chip

Compiler opsætning

For at kunne kompilere koden, skal de rette indstillinger til Atmel Studio vælges. Inde i Atmel vælges: Project -> Projektnavn properties: Toolchain menuen vælges. I configuration vælges All Configurations.



Figur 3.49: Atmel toolchain menu

I menuen for GNU C++ compiler vælges Symbols. Her tilføjes clockfrekvens for arduino chippen og version af Arduino IDE. For at tilføje clockfrekvens skrives F_CPU

= 16000000L, hvis ens clockfrekvens er 16MHz. Arduino IDE version skrives: ARDUINO=106, 106 for version 1.06.

Næste trin er at opsætte Directories, det er stierne til de eksterne biblioteker til Atmel. Arduino har nogle biblioteker, som Atmel skal bruge, så dem skal der linkes til. Bibliotekerne er henholdsvis Arduino.h og pins_arduino.h Stien til Arduino IDE'et er installations stien og resten af stien er som vist på billedet.



Figur 3.50: Atmel directories menu

Gå til Optimization menuen under GNU C++ compiler og sæt flueben ved: Prepare functions for garbage collection (-ffunction-sections).

I menuen for GNU Linker vælges Libraries. Her tilføjes core filen fra tidligere samt den sti til filen. En vigtig detalje er at core filen skal ligge over libm filen under Libraries i Atmel, som kan ses på figur 3.51.



Figur 3.51: Atmel linker menu

Atmel Studio er nu sat op til at fungere sammen med Arduino biblioteker. For at gøre brug af Arduino biblioteket tilføjes Arduino.h til projektet.

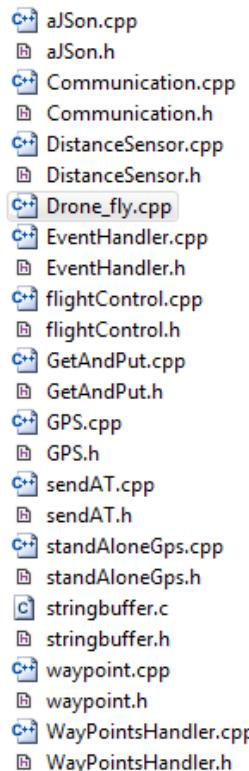
Eksterne bibliotekter til Atmel

I Atmel og arduino findes der ikke standard JSON biblioteker, der kan håndtere JSON objekter. Det har derfor været nødvendigt at bruge et eksternt bibliotek, der kan håndtere disse JSON objekter. Det anvendte bibliotek hedder aJson¹ og er et frit tilgængeligt JSON bibliotek som er lavet til Arduino.

¹<https://github.com/interactive-matter/aJson>

Filoversigt Drone

Dette afsnit omhandler de forskellige filer og biblioteker der er anvendt i Atmel Studio.



Figur 3.52: Atmel mappestruktur

aJSon

JSON biblioteket håndterer sammen med stringBuffer biblioteket de JSON objekter der bruges af dronen. Der kan læses mere om JSON biblioteket her²

Communication

Communication klassen er den klasse der håndterer al kommunikation til dronen. Communication klassen er den øverste klasse i forhold til kommunikationen over 3G.

DistanceSensor

Denne klasses funktionalitet er aflæsningen af sensorværdierne.

Drone_fly

Dette er selve mainen til systemet. Det er denne fil der håndterer alle andre klasser der anvendes i drone systemet.

EventHandler

EventHandler håndterer alle GET og PUT metoder der har med events at gøre.

²<https://github.com/interactive-matter/aJson>

FlightControl

FlightControl klassen styrer selve dronen. Den får input fra både Flight Control boardet og 3G/GPS shieldet og udregner bl.a. orientering.

GetAndPut

GetAndPut klassen er den klasse der er mest hardware nært i forhold til kommunikationen over 3G. Det er denne klasse der kontrollerer hvilken GET og PUT requests der skal sendes og til hvilken server. Til metoderne er der taget udgangspunkt i Cooking Hacks' kode³.

GPS

Denne klasse er en abstrakt klasse, som kun indeholder virtuelle metoder. Det gør at andre klasser der bruger denne abstrakte klasse, som minimum skal implementere de virtuelle metoder.

sendAT

sendAT klassens ansvar er at sende AT kommandoer til 3G/GPS modulet. Denne klasse sender disse AT kommandoer og venter på svar fra modulet hvorefter der returneres en værdi afhængigt af svaret. Koden fra cooking hacks⁴ er brugt som eksempel

standAloneGps

standAloneGps håndterer systemets GPS og de metoder der skal bruges for at hente GPS koordinater. Til GPS er der taget udgangspunkt i Cooking hacks' kode⁵

waypoint

Waypoint klassen bruges til at pakke alt information omkring et givet waypoint ind i et objekt.

WayPointsHandler

WayPointsHandler håndterer alle GET og PUT metoder der har med waypoints at gøre.

⁵<http://www.cooking-hacks.com/documentation/tutorials/arduino-3g-gprs-gsm-gps#step15>

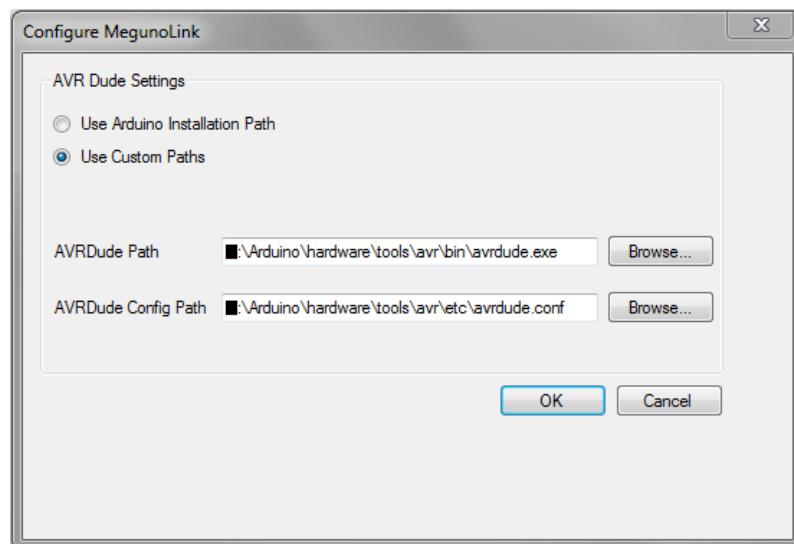
Opsætning af serial programmerings link

For at kunne uploadé kode til en Arduino, udenom Arduino IDE'et, skal der anvendes et program til programmering igennem den serielle forbindelse. MegunoLink Lite er et program med en brugergrænseflade der kan anvendes til at sende og modtage seriell data fra en arduino.

MegunoLink Lite hentes her⁶.

MegunoLink skal sættes op til at uploadé til den korrekte Arduino.

I configuration benyttes "use custom path" og ved AVRDUDE stien linkes til stien med AVRDUDE til arduino og configurationens stien.



Figur 3.53: Konfigurer MegunoLink

For at kunne uploadé, skal MegunoLink vide hvor projektets hex fil ligger. Denne vælges i edit project properties, markeret med en rød cirkel. I dropdown menuen, markeret med en rød pil, vælges den arduino der skal bruges til projektet.



Figur 3.54: MegunoLink edit menu

⁶<http://www.megunolink.com/megunolink-lite/megunolink-lite-plotting-tool/>

3.6.2 Opsætning af timerne til PWM

Da dronen skal flyve autonomt er det påkrævet at dronen skal styres via main controlleren (arduino boardet). Main controlleren styrer dronen ved at sende 6 forskellige pwm signaler til flight control boardet. De 6 pwm signaler bruges til at kontrollere og regulere henholdsvis: Throttle, yaw, pitch, roll, alttitudehold og flightmode.

For at undgå at lave busywait når de forskellige pwm signaler skal ændres (skift i periodetid eller duty cycle) benyttes tre 16-bit timerne som befinner sig på main controlleren. Arduino 2560 boardet har fire forskellige 16-bit timerne, som hver kan styre pwm signal (output signal) på to digitale pins, hvis de bruges korrekt.

Fordelen ved at bruge timerne til at kontrollere de 6 pwm signaler er primært, at pwm signalerne uafhængig og lynhurtigt kan ændres. For at ændre periodetid eller duty cycle på et pwm signal, skal værdien af et eller flere register blot skiftes.

PWM signalets periodetiden kan forandres ved at ændre værdien af registeret der bruges til at bestemme hvor højt timeren skal ”tælle”, dette register kaldes typisk TOP eller ICRn. PWM signalets duty cycle kan forandres ved at ændre værdien af det register der bestemmer hvornår pwm signalet toggles. Dette register kaldes som regel *Output compare*.

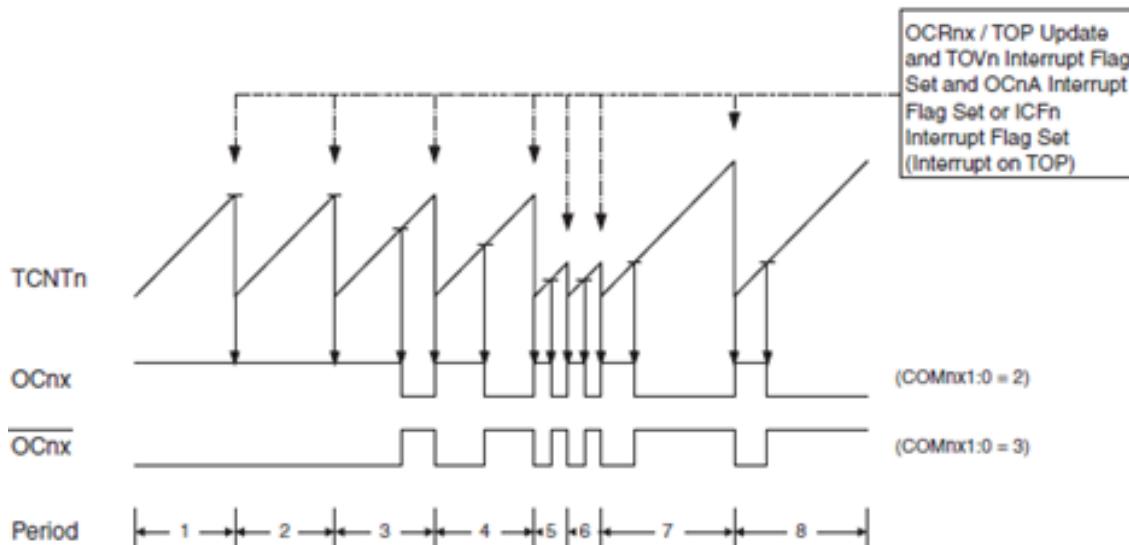
Det blev besluttet at bruge følgende tre timer i Fast PWM mode:

Timer1 - til kontrol af de to digitale pins 11 og 12.

Timer3 - til kontrol af de to digitale pins 2 og 5.

Timer4 - til kontrol af de to digitale pins 6 og 7.

Figur 3.61 viser hvordan output kan se ud ved brug af en timer i Fast PWM mode.



Figur 3.55: Fast PWM Mode, Timing Diagram

I det følgende forklares hvordan de tre 16-bit timerne er sat op. Bla. forklarer valg af mode, udregning af frekvens, samt udregning af hvilke værdier der skal bruges i registerne ICRn og Output compare.

Mode 16-bit timer

Når main controllerens 16-bits timerne benyttes er det muligt at sætte timerne i 16 forskellige modes. Valget af mode styrer dels opløsningen af timeren, men styrer også hvordan timeren skal fungere. Timerne kan bruges i Normal mode, i CTC (Clear Timer on Compare), i Phase Correct PWM Mode eller i Fast PWM.

For at få den størst mulige opløsning benyttes de tre timer i mode 14 - "Fast PWM". I mode 14 bestemmer ICRn registeret værdien af TOP, som er et 16-bit stort. Dette betyder at ICRn maksimalt kan sættes til værdien 0xFFFF (65535).

Waveform Generation Mode Bit Description

Mode	WGMr3	WGMr2 (CTCn)	WGMr1 (PWMr1)	WGMr0 (PWMr0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	-	-	-
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Figur 3.56: Fast PWM Mode, Timing Diagram

Compare Output Mode

Når det overordnede mode er valgt for timerne, skal compare output mode vælges. Compare output mode bestemmer hvornår PMW signalet på de forskellige pin sættes helholdvis højt og lavt.

Det besluttes at sætte Compare output mode til: COMnA1 = 1 og COMnA0 = 0. Med denne indstilling sættes PWM signalet lavt når timerens counter er lig OCnA eller OCnB og sættes højt igen hver gang timerens counter er lig 0. Indstillerne sættes på samme vis for COMnB og COMnC.

Compare Output Mode, Fast PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at BOTTOM (non-inverting mode)
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at BOTTOM (inverting mode)

Figur 3.57: Fast PWM Mode, Timing Diagram

TCCR3A

Da både mode til 16-bit timeren og compare output er valgt kan registeret TCCR3A indstilles. Fra indstilling af mode til 16-bit timeren vides det at WGMn1 = 1, mens WGMn0 = 0. Yderlige vides det at COMnA1 = 1 og COMnA0 = 0 og at COMnB og COMnC skal se ligeledes ud.

Binært skal TCCR3A registeret se således ud: 0b10101010, hvilket i hex svarer til: 0xAA.

Bemærk: Dette delafsnit tager udgangspunkt i TCCR3A registeret tilhørende timer 3. TCCR1A og TCCR4A indstilles på samme vis.

TCCR3A – Timer/Counter 3 Control Register A

Bit	7	6	5	4	3	2	1	0	TCCR3A
(0x90)	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	
ReadWrite	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 3.58: Fast PWM Mode, Timing Diagram

Clock select

Hvilken prescale der vælges har betydning for hvor hurtig en clock der bruges til timeren. I udgangspunkt er det en 16MHz clock der bruges, men ved at bruge prescale er det muligt at mindske clocken. Når der gøres brug af en 16-bit timer, har counteren maksimalt 65535 steps.

Ud fra viden om clockens hastighed og antal steps i counteren kan frekvensen på timerne PWM signal udregnes således: Clock / antal steps = 244,14 Hz.

Hvilket giver følgende periodetid: 1 / 244,14 Hz = 4,096 ms.

Sættes der prescale på clock'en vil frekvensen falde og periodetiden stige. Men da der ønskes periodetid så tæt på 2ms vælges det ikke at bruge prescale.

Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	clk _{I/O} /1 (No prescaling)
0	1	0	clk _{I/O} /8 (From prescaler)
0	1	1	clk _{I/O} /64 (From prescaler)
1	0	0	clk _{I/O} /256 (From prescaler)
1	0	1	clk _{I/O} /1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Figur 3.59: Fast PWM Mode, Timing Diagram

TCCR3B

Da både mode til 16-bit timeren og clock select er valgt kan registeret TCCR3B indstilles. Fra indstilling af mode til 16-bit timeren vides det at WGMn3 = 1, mens WGMn2 = 1. Yderligere vides det at CSn2 = 0, CSn1 = 0 og CSn0 = 1.

Både ICNC3 (Input Capture Noise Canceler) og ICES3 (Input Capture Edge Select) sættes lig 0. Binært skal TCCR3B registeret se således ud: 0b00011001, hvilket i hex svarer til: 0x19.

Bemærk: Dette delafsnit tager udgangspunkt i TCCR3B registeret tilhørende timer 3. Registrerne TCCR1B og TCCR4B tilhørende timer 1 og 4 indstilles på samme vis.

TCCR3B – Timer/Counter 3 Control Register B

Bit (0x91)	7	6	5	4	3	2	1	0	TCCR3B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 3.60: Fast PWM Mode, Timing Diagram

TCCR3C

Force Output Compare A, B og C bruges kun når timeren er sat til et ikke PWM mode. Derfor sættes Force Output Compare A, B og C = 0. Binært skal TCCR3B registeret se således ud: 0b00000000, hvilket i hex svarer til: 0x00.

Bemærk: Dette delafsnit tager udgangspunkt i TCCR3C registeret tilhørende timer 3. Registrerne TCCR1C og TCCR4C tilhørende timer 1 og 4 indstilles på samme vis.

TCCR1C – Timer/Counter 1 Control Register C								
Bit	7	6	5	4	3	2	1	0
(0x82)	FOC1A	FOC1B	FOC1C	-	-	-	-	-
Read/Write	W	W	W	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

Figur 3.61: Fast PWM Mode, Timing Diagram

Indstilling Top og Compare match

Reelt set er opsætningen af duty cycle og periodetid fuldstændig ligegyldigt. Det eneste der betyder noget, er at de PWM signaler der sendes til flight control boardet har pulsbredde mellem 1-2 ms.

Det vides at periodetiden på pwm signalerne vil være 4,096 ms når der arbejdes med clock på 16MHz, der ikke benyttes prescale og Top = 65535.

Ved opstilling af to simple ligninger, kan det beregnes hvilken værdi der skal skrives ind i compare match registeret for at skabe et pwm signal med pulsbredde på 1 ms.

$$4,096 \text{ ms} * x = 1 \text{ ms}$$

Udtrykket omformes til: $x = 1 \text{ ms} / 4,096 \text{ ms} = 0,244$.

$$\text{Værdi af Compare match for at opnå pulsbredde på 1ms: } \text{Top} * x = 65535 * 0,244 = 16000.$$

På samme vis kan det udregnes hvilken værdi der skal skrives i compare match for at få pulsbredde på 2 ms.

$$4,096 \text{ ms} * x = 2 \text{ ms}$$

Udtrykket omformes til: $x = 2 \text{ ms} / 4,096 \text{ ms} = 0,488$

$$\text{Værdi af Compare match for at opnå pulsbredde på 1ms: } \text{Top} * x = 65535 * 0,488 = 32000.$$

Det er nu beregnet: Pulsbredde på 1ms fås ved compare match = 16000, mens pulsbredde på 2ms fås ved compare match = 32000. For at styre flight control board skal værdien i compare match altså veksle op og ned i størrelse mellem 16000 og 32000.

3.6.3 Opsætning af server

Dette afsnit har til formål at beskrive hvordan et udviklingsmiljø sættes op på en givet maskine, så vider udvikling af systemet kan finde sted. Afsnittet forklarer også hvilke tredjeparts teknologier der er anvendt og eventuelle afhængigheder.

OS

Det anbefales at udviklingen finder sted på en OS X eller Linux maskine, da UNIX-terminalen er et meget brugt værktøj og som standart er windows cmd meget "begrænset". Det kan dog lade sig gøre at bruge windows's cmd med nogle tweaks. Dette er ikke noget der vil blive forklaret yderligere og derfor er det kun UNIX-terminalen der er beskrevet i afsnittet.

IDE/Text Editor

Da alt server opsætningen og logikken er skrevet i python, som er et højniveau script sprog, er der ikke nogle specielle krav til et IDE. Et simpelt tekst redigerings program er at fortrække. Det anbefales enten at bruge Sublime⁷ Text Editoren eller Atom⁸ Editoren. Begge text editors tilbyder god code highlightning og mulighed for at tilføje tredje parts pakker. Et udkast af de pakker der bruges er "PyLinter", "pep8"eller "Linter", som tjekker koden for evt. fejl i indentering, manglende kommentar eller kode standart.

Installation af software

Dette underafsnit vil forklare hvordan diverse software pakker installeres på udviklingernes maskine og hvordan udviklerne starter den lokale server op.

Det første program der skal installeres er git. Skriv følgende i terminalen.

Linux:

```
1 $ sudo apt-get install Git
```

OS X: Kommer som standart med git installeret.

Efter git er installeret på maskinen skal source koden klones fra github. Skriv følgende i terminalen.

```
1 $ git clone https://github.com/Opstrup/drone_backend
```

⁷<http://www.sublimetext.com/2>

⁸<https://atom.io/>

Efter source koden er hentet skal package manager programmet pip⁹ installeres. Skriv følgende i terminalen.

Linux:

```
1 $ sudo apt-get install python-pip
```

OS X:

```
1 $ sudo easy_install pip
```

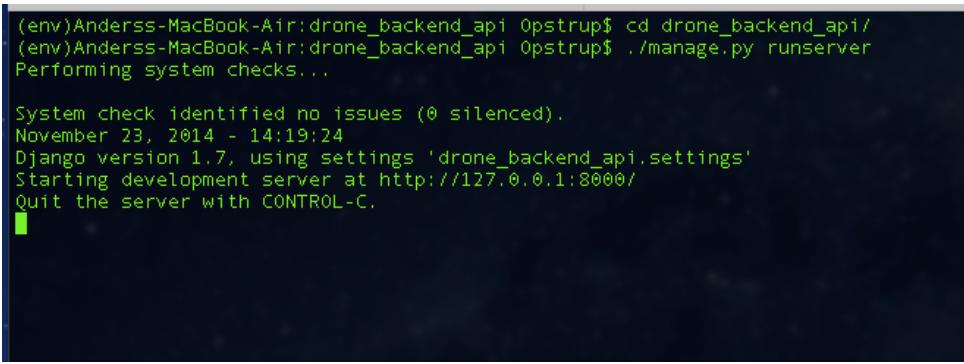
Når pip er installeret skal scriptet "requirements.txt" køres, dette script ligger i roden af server projektet som blev hentet fra github. Skriv følgende i terminalen (da pip bliver brugt som package manager er kommandoens på Linux og OS X).

```
1 $ pip install -r requirements.txt
```

Efter scriptet er kørt og pip har installeret alle afhængigheder, er serveren klar til blive startet. For at starte serveren, skriv følgende i terminalen.

```
1 $ cd drone_backend_api/
$ ./manage.py runserver
```

På figur 3.62 ses et udklip af terminal vinduet ved server startup. Det ses at serveren er startet på "http://127.0.0.1:8000/", bemærk at denne url ikke bruges og vil derfor vise en: 404 error page not found fejl, hvis den tilgås. For at tilgå servere kan api endpoints, som beskrevet i Data view afsnittet, benyttes, http://127.0.0.1:8000/api/drones/.



```
(env)Anderss-MacBook-Air:drone_backend_api Opstrup$ cd drone_backend_api/
(env)Anderss-MacBook-Air:drone_backend_api Opstrup$ ./manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
November 23, 2014 - 14:19:24
Django version 1.7, using settings 'drone_backend_api.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Figur 3.62: Server startup msg

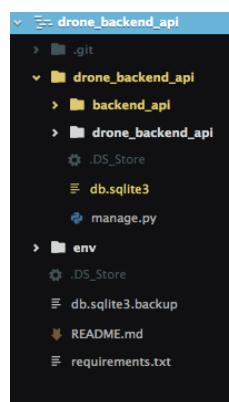
⁹<https://pypi.python.org/pypi/pip>

Mappestruktur

Dette underafsnit vil forklare om den overordnede mappestruktur i projektet og giver en overordnet forståelse af django udvikling.

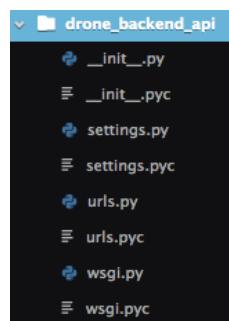
På figur 3.63 ses den overordnete mappestruktur. I roden af mappen findes to væsentlige filer: db.sqlite3 hvilket er database filen til serveren, denne fil vil aldrig blive brugt direkte, men kun indirekte igennem django-frameworket. Den anden fil er manage.py, denne fil er main filen i hele projektet. Filen redigeres aldrig, men bruges kun via terminalen, til forskellige kommandoer på django-applikationen.

I roden findes der også to mapper "backend_api" og "drone_backend_api". "drone_backend_api" mappen er selve projekt mappen og "backend_api" er applikations mappen.



Figur 3.63: Mappestruktur rod

På figur 3.64 ses projekt mappen, de væsentlige filer i denne mappe er settings.py og urls.py. Settings.py er filen hvor alle tredje parts applikationer registeres i "INSTALLED_APPS", filens sikkerhedsindstillinger sættes op i "MIDDLEWARE_CLASSES".Urls.py filen bruges til at registrere hvilke url's serveren kender og hvilke funktioner og views der skal præsenteres. Yderligere dokumentation omkring settings.py og urls.py kan findes under djangos egen dokumentation omkring settings¹⁰ og urls¹¹.

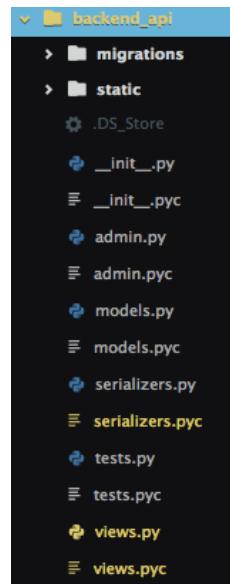


Figur 3.64: Mapestruktur projekt mappe

¹⁰<https://docs.djangoproject.com/en/dev/ref/settings/>

¹¹<https://docs.djangoproject.com/en/dev/ref/urls/>

På figur 3.65 ses applikations mappen. Denne mappe indeholder alt logikken til serveren.



Figur 3.65: Mappestruktur applikations mappe

admin.py

Denne fil indeholder registrering af diverse tables i projektet. De kan tilgås via en admin page, hvor en administrator kan redigere i det data der ligger i databasen. Yderligere dokumentation omkring admin.py kan findes på¹².

models.py

Denne fil indeholder alt logikken til hvilke tabeller og attributter der eksistere i databasen. Via denne fil bliver db.sqlite3 filen som findes i roden, manipuleret. Yderligere dokumentation omkring models.py kan findes på¹³.

serializers.py

Denne fil indeholder serializers kaldes når et api-endpoint bliver kaldt. Via disse serializers finder serveren ud af hvilke data der skal hentes fra databasen og repræsenteres. Disse serializers bliver hovedsageligt brugt til at formater dataen fra databasen til et JSON format. Yderligere dokumentation omkring serializers.py kan findes på¹⁴.

views.py

Denne fil håndtere de requests som kaldes og afgør ud fra dem hvilke serializer der skal generes og præsenteres. Yderligere dokumentation omkring views.py kan findes på¹⁵.

¹²<https://docs.djangoproject.com/en/dev/ref/contrib/admin/>

¹³<https://docs.djangoproject.com/en/dev/topics/db/models/>

¹⁴<https://docs.djangoproject.com/en/dev/topics/serialization/>

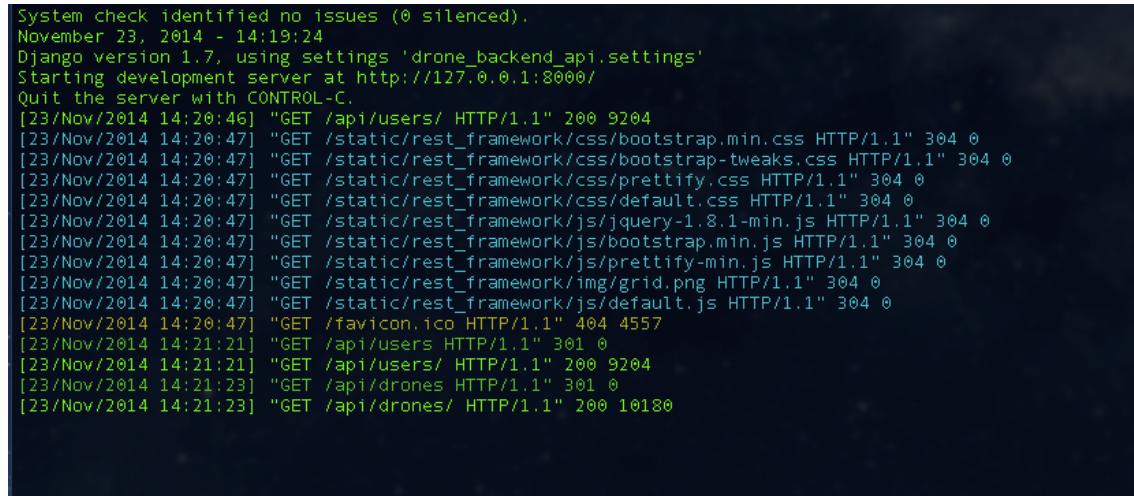
¹⁵<https://docs.djangoproject.com/en/dev/topics/class-based-views/>

Debug

Dette underafsnit vil beskrive hvordan server software kan debugges og hvordan de forskellige værktøjer kan bruges.

Terminal

På figur 3.66 ses et udklip af terminal vinduet, hvor to GET requests har fundet sted. Som det ses på billedet, er det muligt at debugge serveren og tjekke hvilke data sendes frem og tilbage ved hvert request.

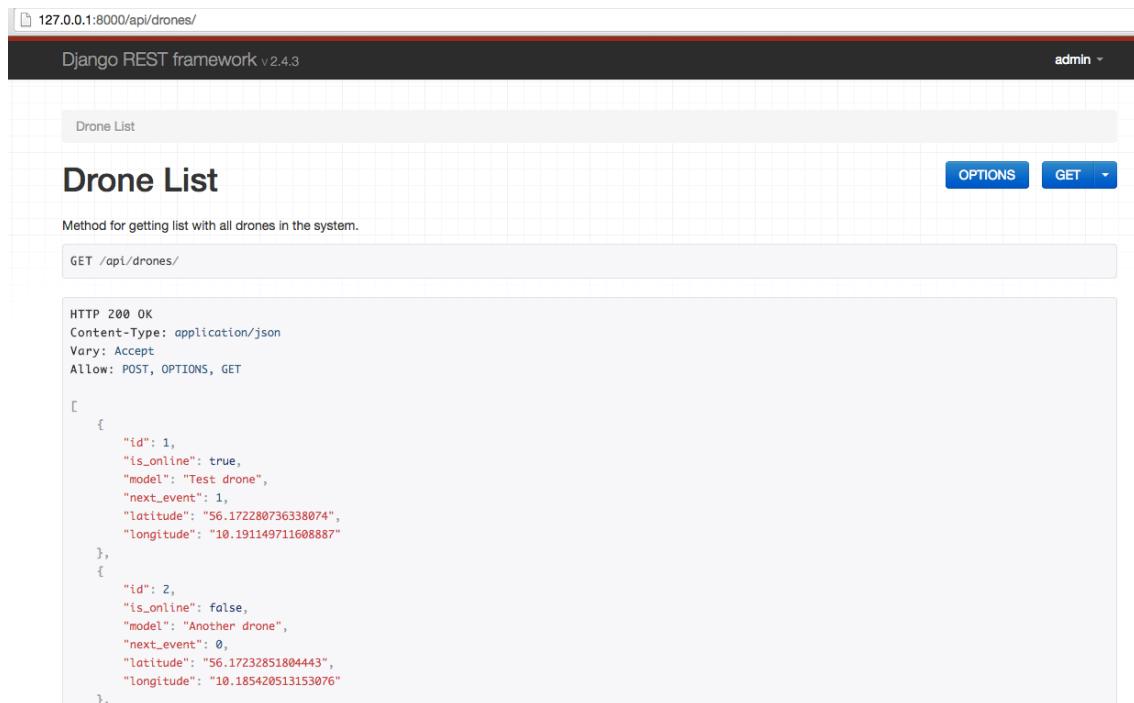


```
System check identified no issues (0 silenced).
November 23, 2014 - 14:19:24
Django version 1.7, using settings 'drone_backend_api.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
[23/Nov/2014 14:20:46] "GET /api/users/ HTTP/1.1" 200 9204
[23/Nov/2014 14:20:47] "GET /static/rest_framework/css/bootstrap.min.css HTTP/1.1" 304 0
[23/Nov/2014 14:20:47] "GET /static/rest_framework/css/bootstrap-tweaks.css HTTP/1.1" 304 0
[23/Nov/2014 14:20:47] "GET /static/rest_framework/css/prettyify.css HTTP/1.1" 304 0
[23/Nov/2014 14:20:47] "GET /static/rest_framework/css/default.css HTTP/1.1" 304 0
[23/Nov/2014 14:20:47] "GET /static/rest_framework/js/jquery-1.8.1-min.js HTTP/1.1" 304 0
[23/Nov/2014 14:20:47] "GET /static/rest_framework/js/bootstrap.min.js HTTP/1.1" 304 0
[23/Nov/2014 14:20:47] "GET /static/rest_framework/js/prettyify-min.js HTTP/1.1" 304 0
[23/Nov/2014 14:20:47] "GET /static/rest_framework/img/grid.png HTTP/1.1" 304 0
[23/Nov/2014 14:20:47] "GET /static/rest_framework/js/default.js HTTP/1.1" 304 0
[23/Nov/2014 14:20:47] "GET /favicon.ico HTTP/1.1" 404 4557
[23/Nov/2014 14:21:21] "GET /api/users HTTP/1.1" 301 0
[23/Nov/2014 14:21:21] "GET /api/users/ HTTP/1.1" 200 9204
[23/Nov/2014 14:21:23] "GET /api/drones HTTP/1.1" 301 0
[23/Nov/2014 14:21:23] "GET /api/drones/ HTTP/1.1" 200 10180
```

Figur 3.66: GET eksempel

Browser

På figur 3.67 ses et andet eksempel på hvordan browseren kan bruges til at debugge serveren. Her ses det hvilke data der kan hentes ved at gå til api-endpoint "<http://127.0.0.1:8000/api/drones/>". Yderligere information kan findes under Data View afsnittet.

*Figur 3.67:* Browser eksempel

Tredje parts teknologier

Til udviklingen af server softwaren er der benyttet nogle tredje parts teknologier, dette underafsnit beskriver hvilke og hvordan de er blevet brugt.

Djangorestframework

Django frameworket er et godt framework til udvikling af webapplikationer. Til projektet ønskes et RESTful api oven på django frameworket, derfor er der brugt et django-rest-frameworket¹⁶.

django-cors-headers

Cors headers¹⁷ er et django applikation som tilføjer CORS (Cross-Origin Resource Sharing) til applikationer. Dette gør det muligt at tilgå applikationen fra andre servere, som fx. client softwaren som kan ligge på en anden server eller dronen som tilgår serveren udefra.

¹⁶<http://www.django-rest-framework.org/>

¹⁷<https://github.com/ottoyiu/django-cors-headers>

3.6.4 Opsætning webapplikation

Dette afsnit har til formål at beskrive hvordan et udviklingsmiljø sættes op på en givet maskine så videre udvikling af systemet kan finde sted. Afsnittet forklarer også hvilke tredjeparts teknologier der er anvendt og eventuelle afhængigheder der er brugt.

For at kunne opsætte et udviklings miljø til client applikationen, skal source koden først hente. Dette kan ske via git, skriv følgende kommando i terminalen.

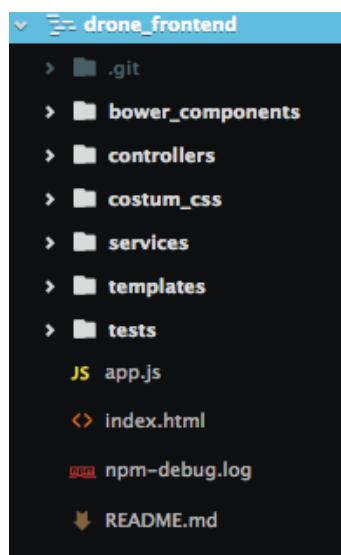
```
$ git clone https://github.com/Opstrup/drone_frontend
```

Mappestruktur

Dette underafsnit vil forklare om den overordnede mappe struktur i webapplikation og give en overordnet forståelse af angular client udvikling.

På figur 3.68 ses mappestrukturen for client applikationen. Da Angular frameworket er opbygget over MVC¹⁸ modellen, er mappestrukturen opdelt så det afspejler denne model. I roden af projekt mappen findes to væsentlige filer, app.js og index.html. app.js er settings filen for projektet, i denne fil bliver alle afhængigheder af projektet hentet ind og initialiseret. States i applikationen og dertilhørende controllere bliver også registeret i app.js¹⁹.

index.html er som kendt fra webudvikling, den side der hentes først ved et request. Da Angular applikationer er opbygget med kun en side, fungerer det lidt anderledes. Når en side opbygget i Angular besøges, vil index siden blive hentet men der uddover vil javascript- og css-filer også blive hentet. Via Angular frameworket bliver alle andre sider loaded ind i index siden, hvilket gøre applikationen hurtigt, idet filerne ikke skal indlæses igen.



Figur 3.68: Mapestruktur rod

¹⁸MVC model i forbindelse med Angular: <https://docs.angularjs.org/guide/databinding>

¹⁹Information omkring app.js indstillingerne: <https://docs.angularjs.org/guide/introduction>

bower_components

I denne mappe findes alle tredje parts afhængigheder, i form af css og java script filer. Til håndtering af tredje parts teknologier er der benyttet bower²⁰, som er et package manager program til webudvikling.

controllers

I denne mappe findes alle controller²¹ i systemet, som fungerer på samme måde som kendt fra normal MVC.

costum_css

Ud over bootstrap til håndtering af css'en, er der også blevet lavet små filer med css kode. Disse filer findes i denne mappe.

services

I denne mappe findes alle services²² i systemet. Disse services indeholder alt buisness logikken i systemet og derved fungere som models i MVC modellen.

templates

I denne mappe findes alle html filerne for systemet, disser filer danner sammen med controllerne de views som bliver præsenteret i browseren²³.

Udviklingsmiljø

Opsætning af et udviklingsmiljø til client applikationen er lige til. Source koden indeholder alt hvad der skal bruges for at kunne compile, der skal kun bruges en browser til at starte applikationen. Til videreudvikling af applikationen anbefales det at udvide med andre applikationer, for at gøre udviklingsarbejdet lettere.

Text editor

Som alt andet web udvikling foregår det hele i scripts som køres i browseren, derfor er der ingen specielle krav til et IDE. Nogle gode text editors til web udvikling er Sublime eller Atom²⁴.

Lokal web server

For at applikationen fungerer optimalt, er det at foretrakke at der installeres en web server på udviklerens maskine. Der er ikke nogle specielle krav til hvilke slags server. En mulighed ville være at sætte en Apache server op. For at simplificere installationen og opsætningen kan XAMPP²⁵ benyttes. Et andet eksempel på en web server kunne være en node.js server²⁶.

²⁰<http://bower.io/>

²¹<https://docs.angularjs.org/guide/controller>

²²<https://docs.angularjs.org/guide/services>

²³<https://docs.angularjs.org/guide/templates>

²⁴Yderligere beskrevet i afsnittet om Server Opsætning

²⁵<https://www.apachefriends.org/index.html>

²⁶<http://nodejs.org/>

Package management

Som før nævnt er bower brugt til package management og det anbefales at der fortættes med dette da bower²⁷ selv downloader og installerer de påkrævede filer, så udvikleren ikke skal tænke yderligere over dette.

Tredje parts teknologier

Til udviklingen af web applikationen er der blevet brugt nogle tredje parts teknologier, som er beskrevet yderligere i dette under afsnit.

bootstrap

Det populære bootstrap²⁸ css framework er benyttet til at designe applikationen.

ui-router

Ui router²⁹ er en applikation som er lavet til Angular frameworket som gør routing imellem diverse views meget simpel.

google-maps

Google maps er blevet brugt til at vise kortet i webapplikationen. Via google maps bliver der hentet GPS-koordinater som dronen flyver efter. Standard Google maps API'et er ikke blevet benyttet, men et angular applikation³⁰ er anvendt i stedet, da det fungerer bedre med det resterende kode.

²⁷<http://bower.io/>

²⁸<http://getbootstrap.com/>

²⁹<https://github.com/angular-ui/ui-router>

³⁰<https://angular-ui.github.io/angular-google-maps/>