Ingeniørhøjskolen Aarhus Universitet

Projektrapport

Autonomous Drone Control
Projektnummer: 14144

Forfattere

Navn: Jens Kuhr Jørgensen **Studienummer:** 11690
Dato 17-12-2014
Underskrift: _____

Navn: Thomas Fiil Lyngholm **Studienummer:** 11641
Dato 17-12-2014
Underskrift: _____

Navn: Rasmus Fredensborg Jensen **Studienummer:** 11471
Dato 17-12-2014
Underskrift: _____

Vejleder

Navn: Torben Gregersen
Dato 17-12-2014
Underskrift: _____

Revisionshistorik

Version	Dato	Ændring
1.0	24-11-2014	Dokument oprettet.
1.1	12-11-2014	Første korrekturlæsning på det samlede dokument.
2.0	15-12-2014	Korrekturrettelser.

Tabel 1. Revisionshistorik.

Ordforklaring

Følgende tabel indeholder en ordliste over de forkortelser og tekniske ord, der er benyttet i rapporten, og som kræver yderligere forklaring.

Forkortelse	**Betydning**	**Forklaring**
ADC	Autonomous Drone Control	Navnet på projektet.
O1	Option 1	Beskriver et alternativ til en handling i en use case.
Throttle	Throttle	Flyveterm: Beskriver, hvor meget gas dronens motorer får.
Yaw	Yaw	Flyveterm: Beskriver dronens rotation om dens egen z-akse.
Pitch	Pitch	Flyveterm: Beskriver dronens rotation om dens egen y-akse.
Roll	Roll	Flyveterm: Beskriver dronens rotation om dens egen x-akse.
UML	Unified Modeling Language	Hardware-struktureringsværktøj.
SysML	Systems Modeling Language	Software-struktureringsværktøj.
HTTP	HyperText Transfer Protocol	Applikationslagsprotokol.
TCP	Transmission Control Protocol	Transportlagsprotokol.
SQL	Structured Query Language	Programmeringssprog til databaser.
SQLite		Simpel udgave af SQL til embedded systemer og lignende.
RC	Radio Control	Anvendelse af radiosignaler til fjernstyring.
ASE/IHA	Aarhus School of Engineering/Ingeniørhøjskolen Aarhus	
GPS	Global Positioning System	Globalt satellit navigeringssystem.
3G	3. generation	3. generations kommunikationsnetværk.
Scrum		Agil udviklingsmetode.
RUP	Rational Unified Process	Iterativ udviklingsproces.

Tabel 2. Ordforklaringstabel (1/2).

Forkortelse	**Betydning**	**Forklaring**
HW	Hardware	
SW	Software	
UI	User Interface	Brugergrænseflade
GUI	Graphical User Interface	Grafisk brugergrænseflade
Bdd	Block definition diagram	
Ibd	Internal block diagram	
stm	State machine	Tilstandsmaskine
HTTP	Hypertext Transfer Protocol	Applikationsprotokol
PWM	Pulse-Width Modulation	Pulsbreddemodulation
TCP	Transmission Control Protocol	Kommunikationsprotokol
IDE	Integrated Development Environment	Integreret softwareudviklingsmiljø.
XML	Extensible Markup Language	Opmærkningssprog der anvendes til indpakning af data.

Tabel 3. Ordforklaringstabel (2/2).

Arbejdsfordeling

Nedenfor er specificeret arbejds- og ansvarsområder for udviklingsteamets medlemmer. Udover nedenstående arbejdsopgaver har hele teamet i fællesskab deltaget i udarbejdelsen af den overordnede systemarkitektur og -design samt accepttest og kravspecifikation.

Drone

Medlemmer:

- Jens Kuhr Jørgensen(11690)
- Rasmus Fredensborg Jensen(11471)
- Thomas Fiil Lyngholm(11641)

Arbejdsområder:

- Motordriver(Rasmus)
- GPS(Thomas)
- Kommunikation med server(Thomas)
- PID regulering(Jens)
- Styring af drone(Jens)

App

Medlemmer:

- Jens Kuhr Jørgensen(11690)
- Rasmus Fredensborg Jensen(11471)
- Thomas Fiil Lyngholm(11641)

Arbejdsområder:

- Kommunikation med server(Rasmus og Thomas)
- Valg af koordinater(Jens)
- SQL database(Jens)

Server

Medlemmer:

- Rasmus Fredensborg Jensen(11471)

Arbejdsområder:

- Implementering af server(Rasmus)

Abstract og resumé

Resumé

I dette dokument beskrives udviklingen og implementeringen af bachelorprojektet "Autonomous Drone Control"(ADC). Formålet med projektet er at udvikle et autonomt dronestyringssystem til anvendelse ved applikationer, hvor en visuel gengivelse af svært tilgængelige objekter eller områder er nødvendig. Prototypen består overordnet af tre dele: En Android applikation, en server med en database og en drone. Brugeren af systemet kan, vha. Android applikationen, kommunikere med dronen via serveren. Systemkomponenterne er implementeret på en Android smartphone, en Windows Server 2012, en MSSQL database og en "AeroQuad Cyclone ARF Kit" -drone med en Arduino Mega2560.
I projektets udviklingsprocess er der anvendt UML og SysML til at visualisere systemdesignet. Ydermere er der anvendt SCRUM og RUP som udviklingsværktøjer.

Rapporten beskriver projektforløbet og de værktøjer og metoder, der er anvendt for at designe og udvikle produktet. Samtidig gennemgår rapporten det realiserede system, og de overvejelser udviklingsteamet har gjort sig, samt de beslutninger der er taget, gennem projektforløbet.
Baseret på kravene til produktet er der specificeret og udført en accepttest. Ud fra denne kan det konkluderes, at en stor del af det kravspecificerede produkt er godkendt. Selve den autonome flyvning af dronen har imidlertid vist sig at være mere kompliceret end først antaget, hvorfor denne del kræver videre arbejde for at kunne godkendes. Overordnet set er arbejdet med projektet resulteret i en prototype, hvor al basal kommunikation mellem app, server og drone er færdigimplementeret, og autonom flyvning er implementeret, men ikke færdigoptimeret.

Abstract

This document describes the development and implementation of the bachelor project "Autonomous Drone Control"(ADC). The purpose of the project is to develop an autonomous drone control system to be used in applications where a visual representation of inaccessible objects or areas is needed. The project consists primarily of three main parts: an Android application, a server with a database and a drone. The user of the system can communicate with the drone via the server by using the Android application. The system components are implemented on an Android smartphone, a Windows Server 2012, a MSSQL database and an "AeroQuad Cyclone ARF Kit" drone with an Arduino Mega2560. In the project development process UML and SysML are used to visualize the system design. Furthermore, Scrum and RUP are used as development tools.

This report describes the progress of the project and the tools and methods used to design and develop the product. Furthermore the report describes the implemented system, along with considerations and decisions made by the development team throughout the project. Based on the product requirements an acceptance test has been specified and conducted. From the acceptance test it can be concluded that a large proportion of the specified product requirements are approved. The autonomous flight of the drone has proved to be more complicated than expected, which is why this part requires further work in order to be approved. Overall, the project has resulted in a prototype where all the basic communication between the app, server and drone is complete and autonomous flight is implemented but not fully optimized.

Indholdsfortegnelse

Indledning 1

Droner er de seneste år blevet mere tilgængelige, pålidelige og fleksible over for brugerdefinerede behov. Denne udvikling har gjort dronerne mere rentable inden for en lang række erhverv. Mange af disse erhverv har det til fælles, at de ønsker en visuel gengivelse af svært tilgængelige objekter, og netop denne opgave varetager dronerne til perfektion. En problematik ved dronerne er dog, at de kan være udfordrende at manøvrere afhængig af omgivelser og vejrforhold. Derfor vil det ofte være nødvendigt, at være i besiddelse af en erfaren dronepilot, for at kunne gennemføre en sikker flyvning.

Denne rapport er en del af et bachelorprojekt udarbejdet af tre elektrostuderende på Ingeniørhøjskolen, Aarhus Universitet. Projektet omhandler udviklingen af et autonomt dronestyringssystem. Brugeren af systemet skal via en smartphone applikation kunne bestemme dronens destination, samt få adgang til de billeder dronen fotograferer på destinationen, samt en log over dronens flyvning. For at opnå forbindelse til dronen under hele flyvningen, anvendes 3G til kommunikation med dronen, hvorfor der også implementeres en server til at varetage al kommunikation mellem smartphone app og drone.

Et eksempel på en brugssituation for systemet kunne være detaljeret fotografering af landområder, som led i kortlægning i forbindelse med byggeprojekter. Blandt andre eksempler vil landinspektører kunne drage nytte af systemet ved projekter, der vedrører anlæggelse af vej og jernbaner, da der her skal foretages detaljeret fotografering af store områder. Her vil systemet kunne erstatte brugen af en almindelig helikopter med pilot og fotograf, og samtidig kunne flyve lavere end en helikopter, og dermed være i stand til at tage mere detaljerede billeder.[1]

I denne rapport ønskes følgende behandlet:

1. Smartphone app implementeret på en Android-platform.
2. Server
3. Embedded software til autonom styring af drone.
4. Dataoverførsel mellem smartphone og server.
5. Dataoverførsel mellem drone og server.

Rapporten er opbygget i forhold til det overordnede projektforløb, og beskriver indledningsvis de ikke-tekniske aspekter af projektet, såsom problemanalyse, fremgangsmåde, krav og en overordnet systembeskrivelse. Herefter gennemgås detaljer for det udviklede produkt, og vigtige resultater fremhæves og diskuteres.

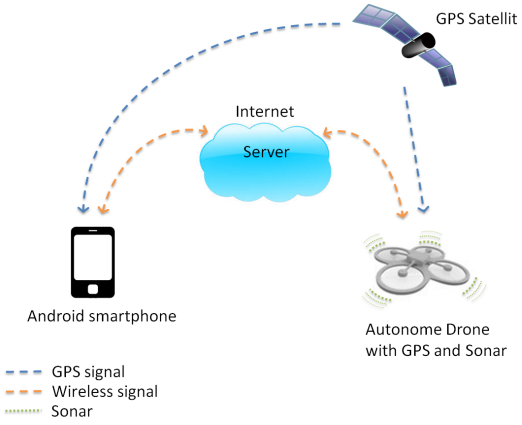
[1] http://www.byggeplads.dk/nyhed/2011/11/landinspektoer/droner-kortlaegning

Opgaveformulering og krav 2

I dette afsnit beskrives det problem der ønskes løst, samt de krav der stilles til den realiserede løsning.

Målet med dette projekt er at gøre det muligt for en uøvet operatør, via en smartphone applikation, at sende en drone til en ønsket destination. Dronen skal efter opstart flyve autonomt, og på destinationen skal dronen tage billeder og gøre disse tilgængelige for brugeren.

En "AeroQuad Cyclone ARF Kit"-drone er tilgængelig på ASE/IHA, og denne danner derfor udgangspunkt for dronevalget. Den software der styrer dronen, er tilgængelig som open source-kode[1].
På dronen skal der monteres en GPS-modtager, så den kan registrere sin nuværende position. Desuden skal der monteres et 3G-modul, så dronen kan få adgang til internettet. Brugerens interface til dronen skal dannes igennem en Android smartphone applikation. Kommunikation mellem bruger og drone skal foregå over internettet, og i projektet skal der derfor også implementeres en server, der kan tilgås fra både Android applikation og drone. Dronen skal ydermere være i stand til at fotografere og uploade billeder til serveren, mens den er i luften. Brugeren skal kunne få adgang til disse billeder i Android applikationen via internettet. Når de nødvendige billeder er taget, skal dronen selv flyve tilbage til dens udgangsposition. Udover at sende billederne til serveren, skal dronen lagre dem lokalt.
Dronen skal være udstyret med sonarsensorer for at muliggøre detektering af forhindringer og derved forhindre kollision. Desuden skal der monteres en lydgiver på dronen for at øge sikkerheden ved brug af dronen. På figur 2.1 ses en principskitse af systemet.



Figur 2.1. Principskitse af ADC.

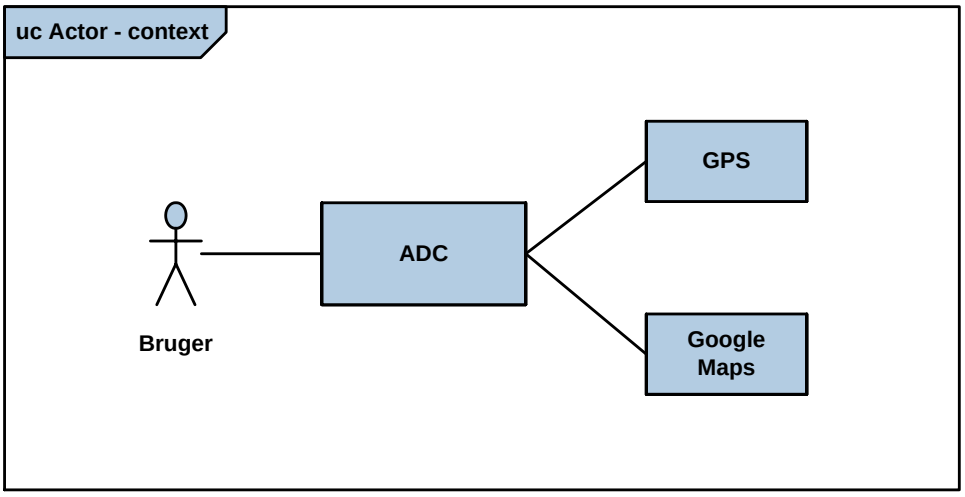
[1] https://github.com/AeroQuad/AeroQuad

Systembeskrivelse 3

I dette afsnit vil systemets funktionalitet blive uddybet. Dette gøres vha. use case diagrammer og aktørbeskrivelser.

3.1 Aktører

På figur 3.1 ses aktør-kontekst diagrammet for systemet. Diagrammet viser relationerne mellem systemet og dets aktører. Til venstre på diagrammet ses den primære aktør/bruger af systemet, og til højre ses de sekundære aktører. De sekundære aktører beskriver aktører, der ikke direkte har interesse i systemet, men som yder en service for systemet. Diagrammet identificerer således systemets eksterne grænseflader. Af diagrammet fremgår det, at systemet har grænseflader til brugeren, GPS og Google Maps.

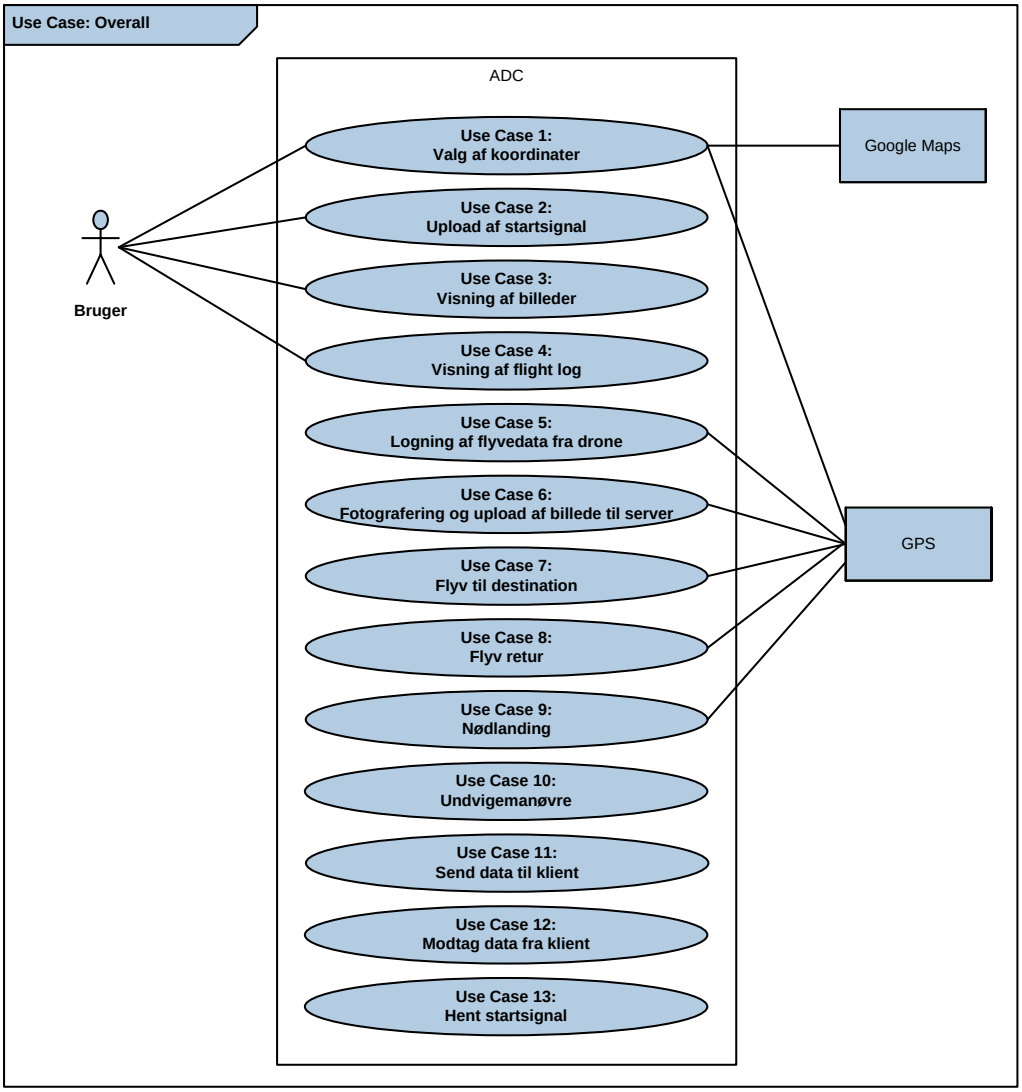


Figur 3.1. Aktør-kontekst diagram.

På figur 3.2 ses det overordnede use case diagram for systemet. Dette diagram kæder aktørerne sammen med de use cases, de er aktive i.

Systemets primære aktør er brugeren af ADC. For at anvende systemet skal brugeren være i besiddelse af et komplet ADC system bestående af en Android smartphone med systemets tilhørende applikation, en ADC server samt en ADC drone. Derudover skal de sekundære aktører være tilgængelige. De sekundære aktører er GPS og Google Maps. GPS står for positionsbestemmelse af smartphone og drone, mens Google Maps forsyner app'en med et

interaktivt kort.



Figur 3.2. Overordnet use case diagram.

Hver use case beskriver en funktionalitet i systemet. Et tænkt brugsscenarie kunne være:

- Use case 1: Valg af koordinater.
- Use case 2: Upload af startsignal.
- Use case 12: Modtag data fra klient.
- Use case 13: Hent startsignal.
- Use case 11: Send data til klient.
- Use case 7: Flyv til destination.
- Use case 6: Fotografering og upload af billede til server.
- Use case 12: Modtag data fra klient.
- Use case 3: Visning af billeder.
- Use case 11: Send data til klient.
- Use case 8: Flyv retur.

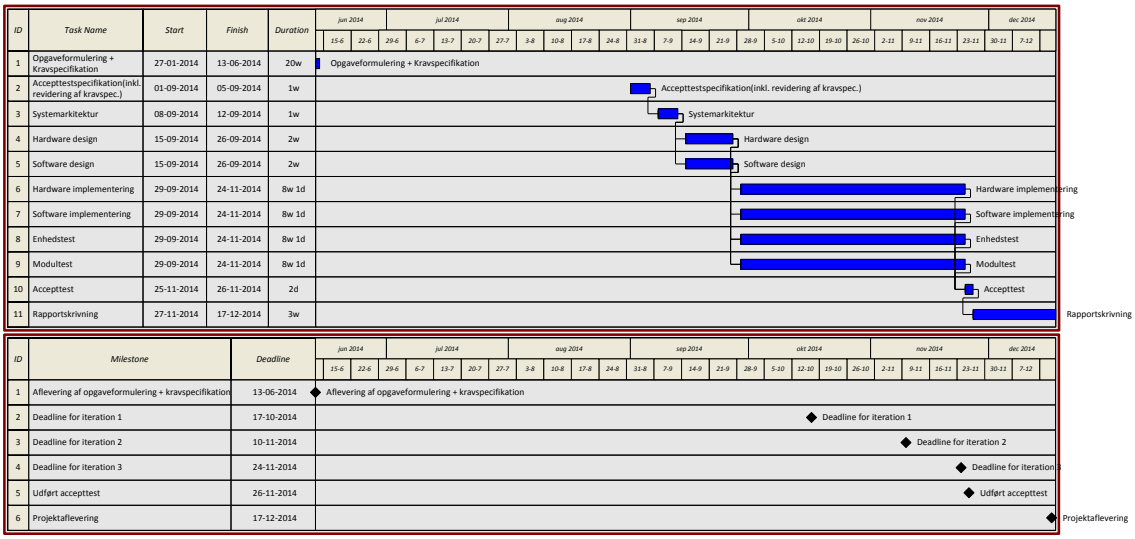
For en detaljeret beskrivelse af use case'ene, se kapitel 1 i dokumentet "Kravspecifikation".

Udviklingsproces 4

Dette afsnit omhandler udviklingsteamets planlægnings- og styringsværktøjer samt den anvendte fremgangsmåde i udviklingsforløbet. Derudover beskrives den prioritering, der er valgt i implementeringsrækkefølgen, samt de afgrænsninger der er gjort.

4.1 Styring og planlægning

I starten af projektet har teamet udarbejdet en overordnet tidsplan. Tidsplanen har sidenhen gennemgået mindre revideringer, og den sidste revision er vist på figur 4.1.



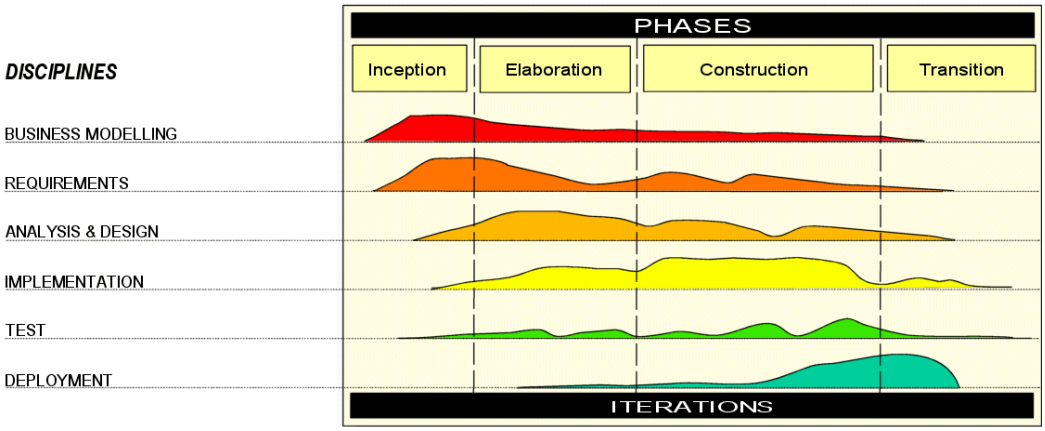
Figur 4.1. Overordnet tidsplan.

Tidsplanen indeholder tre væsentlige elementer, der udgør hvert deres formål for teamet:

- Datoer: Udviklingsteamets medlemmer ved på hvilke datoer de forskellige milestones skal være færdige.
- Milestones: Udviklingsteamets medlemmer ved hvilket indhold af projektet, der skal være færdigt.
- Overblik: Sikrer, at teamet har overblik og derved kan sikre fremskridt i projektet.

Disse tre elementer giver udviklingsteamet et godt overblik over projektets forløb, og er med til dels at sænke stressniveauet for teamets medlemmer og dels at højne muligheden for et tilfredsstillende produkt ved projektforløbets udgang.

Styringen af projektet har været inspireret af de agile arbejdsmetoder RUP[1] og Scrum[2]. RUP deler et projektforløb op i fire faser(se figur 4.2): Inception(start), elaboration(uddybning), construction(konstruering) og transition(overgang/ibrugtagning). I udarbejdelsen af dette projekt er inception-fasen først udført for hele projektet, dvs. der indledningsvis er blevet lavet en kravspecifikation for hele systemet. Derefter er projektet delt op i fire iterationer, der udgør en Scrum product backlog. For hver af de fire iterationer er de tre sidste RUP-faser gennemført for mindre sprints på få dage til få uger. Ved udgangen af hvert sprint leveres der således en gennemtestet softwarepakke.



Figur 4.2. RUP.

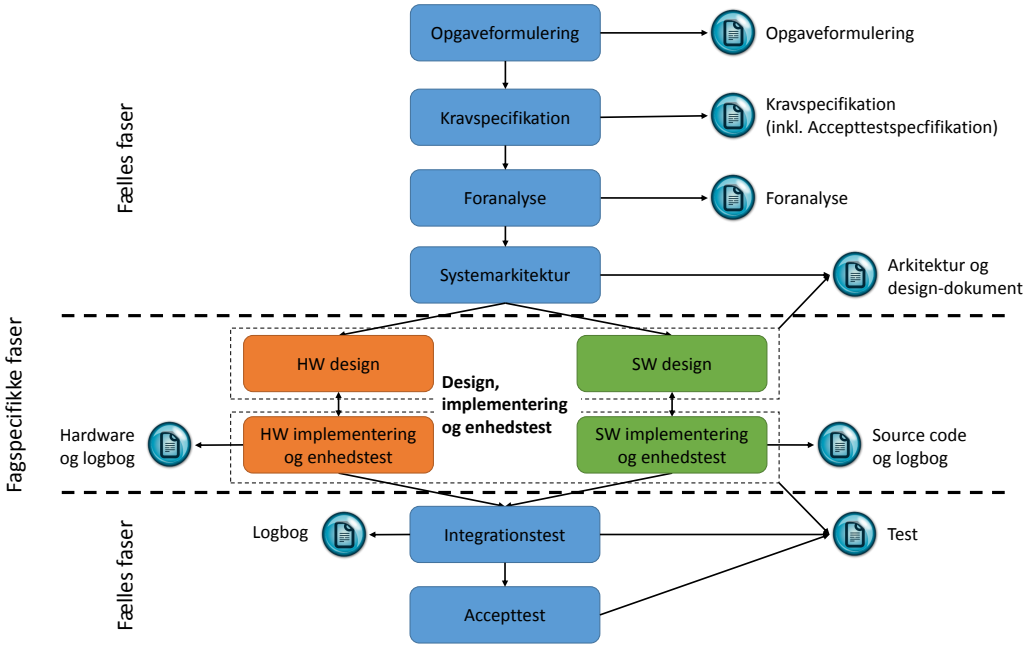
4.1.1 Møder

Der har i projektforløbet ikke været en dedikeret projekt-/Scrum-leder. Dette skyldes udviklingsteamets begrænsede størrelse samt homogeniteten blandt medlemmerne. Teamet har dagligt været i stand til at diskutere opståede problemstillinger, og dette har fjernet behovet for deciderede teammøder. Der er dog stadig blevet holdt et ugentligt møde med udviklingsteamets vejleder med henblik på rådgivning i projektarbejde. For at sikre fuldtalligt fremmøde til vejledermøderne, samt for at sikre generel forventningsafstemning i udviklingsteamet, er der ved projektets start indgået en samarbejdskontrakt[3] mellem teamets medlemmer.

4.2 Fremgangsmåde

Teamets fremgangsmåde i udarbejdningen af dokumentationen til projektet er inspireret af ASE-modellen[4]. På figur 4.3 ses den modificerede model, der viser, hvilke dokumenter/artefakter der opstår ud fra hvilke faser.

[1] http://en.wikipedia.org/wiki/Rational_Unified_Process
[2] http://en.wikipedia.org/wiki/Scrum_(software_development)
[3] Samarbejdskontrakten kan ses på den vedlagte CD med filnavnet "Samarbejdskontrakt.pdf".
[4] Se bilag "Vejledning til gennemførelse af projekt 1 - v.1.1.pdf" på CD.



Figur 4.3. Modificeret udgave af ASE-modellen.

Projektet er således mundet ud i følgende dokumenter: "Opgaveformulering", "Kravspecifikation", "Foranalyse", "Arkitektur og design"og "Test".
I det følgende beskrives indholdet af projektfaserne.

Opgaveformulering

I denne fase formuleres overordnet, hvad projektet går ud på. Opgaveformuleringen afspejler de første samtaler med kunden, og beskriver kort, hvad kunden har af krav til systemet. I dette projekt agerer udviklingsteamet også kunde.

Kravspecifikation

I denne fase defineres en specifik kravspecifikation til produktet. Kravspecifikationen består primært af funktionelle og ikke-funktionelle krav samt en accepttestspecifikation til systemet. De funktionelle krav beskriver kravene til systemets funktionalitet, og opstilles på baggrund af use cases, der beskriver del-funktionaliteter i systemet. De ikke-funktionelle krav beskriver implementeringsspecifikke krav, der ikke har direkte indflydelse på, hvordan systemet fungerer. Til at teste og verificere kravspecifikationen opstilles i samme fase en accepttest med definerede testprocedurer.

Foranalyse

Sideløbende med udarbejdelsen af kravspecifikationen udfærdiges en foranalyse, hvori forskellige løsningsmuligheder undersøges. Analysen har til formål at udføre "proof of concept"ved i et tidligt stadie at finde ud af, hvad der kan lade sig gøre, og hvad der ikke kan lade sig gøre, og derved minimere eventuelle risici ved projektet. Ved udgangen af denne fase ved udviklingsteamet mere om, hvilke teknologier der skal anvendes for at løse den givne opgave.

Systemarkitektur

I denne fase defineres den overordnede arkitektur for projektet. Projektets hardware og software deles op i mindre blokke for at overskueliggøre projektet. Overordnede strukturelle diagrammer for hard- og softwareblokkene opstilles, og grænseflader mellem blokkene defineres.

Design

I denne fase specificeres indhold og funktionalitet af de i systemarkitekturen opstillede moduler og grænseflader.

Implementering og enhedstest

I denne fase implementeres projektets blokke, så disse opfylder de i designfasen definerede specifikationer. Dette indbefatter blandt andet realisering af det designede hardware og implementering af softwareklasser. I forbindelse med implementeringen udføres der løbende enhedstests. Der er igennem hele projektet ført logbog over vigtige resultater og eventuelle komplikationer i forbindelse med de udførte enhedstests.

Integrationstest

I denne fase samles enhederne fra implementeringsfasen, og der udføres integrationstests for at verificere enhedernes samspil. Der er ligeledes ført logbog over de udførte integrationstests.

Accepttest

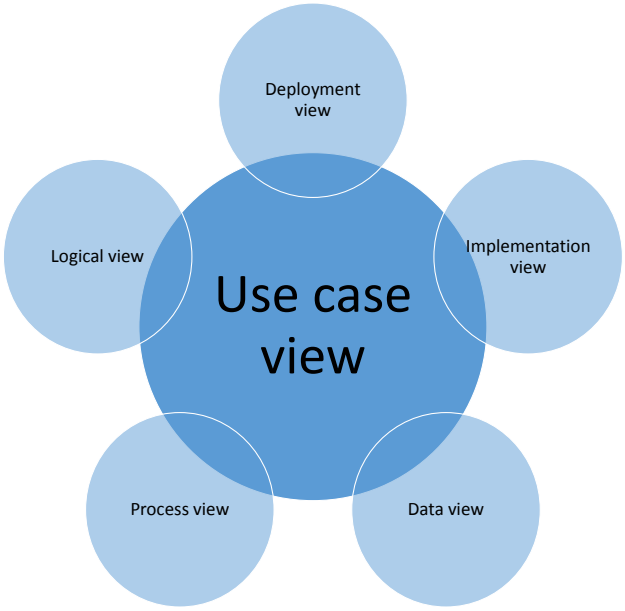
I den sidste fase af projektet gennemføres den tidligere specificerede accepttest.

4.3 Iterationer

Systemets funktionalitet er delt op i 13 use cases, og disse use cases er udviklet over fire iterationer[5]. Som det fremgår af tidsplanen på figur 4.1, er tre af de fire iterationer tildelt hver deres deadline. Inddelingen i iterationer med deadlines betyder, at der er mulighed for at opdage og reagere på eventuelle forsinkelser i udviklingen på et tidligere stadie end ellers. Projektets funktionalitet er afgrænset til de første tre iterationer, og den fjerde iteration er derfor ikke tildelt nogen deadline, da use casene i denne iteration ikke har været forventet implementeret inden for det tilgængelige tidsrum.

4.4 N + 1 view model

N + 1 view model'en[6] bruges til at beskrive et softwaresystem for forskellige interessenter med forskellige syn og interesser i systemet. Modellen tager udgangspunkt i et use case view, der repræsenteres ved "+ 1" i modellen. Derefter er det op til systemudviklerne at bestemme, hvor mange views der skal til, for at systemet er tilstrækkeligt belyst. Til dette system anvendes der en 5 + 1 view model. De anvendte views ses på figur 4.4.



Figur 4.4. 5 + 1 view model.

4.4.1 Use case view

Use case view'et består af use cases, og beskriver systemets brugsscenarier. Udover et use case view er der anvendt et til fem views til at belyse hver use case. Use case diagrammerne kan findes i dokumentet "Kravspecifikation".

[5] Use cases og iterationer kan ses i afsnit 1.2 og 1.6 i dokumentet "Kravspecifikation".
[6] http://en.wikipedia.org/wiki/4%2B1_architectural_view_model

4.4.2 Logical view

Logical view'et beskriver funktionaliteten i systemet, og hvordan systemet internt opererer under de forskellige brugsscenarier. I logical view'et konstrueres applikationsmodeller for hver use case, hvor tre typer diagrammer indgår: Sekvensdiagrammer, tilstandsdiagrammer/-maskiner og klassediagrammer. I dette projekt benyttes ydermere tre-lags modellen[7] til at beskrive app'en, hvor klassediagrammet inddeles i tre lag:

- Presentation tier
- Logic tier
- Data tier

Præsentationslaget indeholder de UI-elementer, som brugeren bliver præsenteret for. Det logiske lag indeholder alle de klasser, der står for funktionalitet, beregninger og logiske beslutninger, mens datalaget indeholder interfaces til databaser, filsystemer o.l., hvor information gemmes og hentes.

4.4.3 Process view

Process view'et beskriver de forskellige processer/tråde i systemet, hvordan samspillet imellem trådene er, hvilke processer der kører sideløbende med hinanden og hvornår.

4.4.4 Deployment view

Deployment view'et(også kendt som physical view), beskriver på hvilke fysiske enheder softwarekomponenterne er implementeret. Derudover beskriver view'et de anvendte kommunikationsprotokoller.

4.4.5 Data view

Data view'et beskriver, hvordan persistent data lagres i systemet. I dette view beskrives også de anvendte databasers struktur.

4.4.6 Implementation view

Implementation view'et(også kendt som development view) beskriver bl.a. filstrukturen i udviklingssystemet, samt hvilken compiler der er anvendt til at compilere koden og en "howto compile"-guide. Formålet med view'et er, at muliggøre en reproduktion af softwaren. View'et indeholder desuden interessante elementer om implementeringen af projektet, der ikke hører til i nogle af de andre views, herunder omstændige source-kode kommentarer.

[7] http://en.wikipedia.org/wiki/Multitier_architecture

4.5 Softwareudviklingsprincip

Softwareudviklingen i projektet er drevet af det agile udviklingsprincip *Make it work, make it right, make it fast*[8], der dækker over tre faser i et iterativt udviklingsforløb. Ved tilføjelse af ny funktionalitet kan man, ved at følge denne tankegang, opnå gode resultater. De tre faser indeholder følgende:

- *Make it work*: Implementér den basale funktionalitet så enhver test vil lykkes. Der lægges ikke fokus på det æstetiske, kun funktionelle.
- *Make it right*: Fokus ændres nu til kodeopbygningen. Koden omstruktureres, så unødvendig kode fjernes. Navne og opbygning rettes, så koden bliver nemmere at overskue og vedligeholde.
- *Make it fast*: I den sidste fase optimeres kodens ydeevne.

4.6 Udviklingsværktøjer

I dette afsnit beskrives de væsentligste udviklingsværktøjer, der er gjort brug af i udviklingen af dette system.

UML

UML(Unified Modeling Language) er et grafisk sprog til strukturering og dokumentation af softwareintensive systemer. Sproget dækker over en standard for udseende af diagrammer til beskrivelse af objekt-orienterede softwaresystemer. UML er i dette projekt anvendt til at designe og dokumentere systemets software vha. bl.a. domænemodeller, sekvensdiagrammer, klassediagrammer og tilstandsdiagrammer.

SysML

SysML(Systems Modeling Language) er et grafisk sprog til strukturering og dokumentation af systemudvikling. SysML udgør en undergruppe og en udvidelse af UML, og de to sprog har derfor flere fælles træk i deres grafiske udtryk. SysML er i dette projekt anvendt til at beskrive systemets hardware vha. block definition diagrams(bdd'er) og internal block diagrams(ibd'er)[9].

Microsoft Visio 2010

Visio er anvendt med skabelonen "UML2.2-Visio2010"[10] til at udvikle SysML- og UML-diagrammer for systemet.

[8] http://agileinaflash.blogspot.dk/2009/03/make-it-work-make-it-right-make-it-fast.html
[9] http://www.sysmlforum.com/
[10] Skabelonen kan findes på den vedlagte CD.

LaTeX

LaTeX er anvendt til dokumentering og rapportering. LaTeX er et opmærkningssprog til tekstformatering, der har den fordel ift. f.eks. Word, at skribenten ikke skal tænke over design og layout, når først preamble'en er sat op. Dette gør, at vedligeholdelse af mellemstore/store dokumenter bliver lettere.

TortoiseSVN v1.8.8 og SmartSVN

TortoiseSVN og SmartSVN er versionsstyringsværktøjer, der er blevet anvendt til versionsstyring af software og LaTeX dokumenter.

Microsoft Power Point 2013

Power Point er anvendt til at designe diagrammer og figurer til dokumentationen af projektet.

Microsoft OneNote 2013

OneNote er anvendt til at føre logbog i.

Microsoft Visual Studio 2012/13

Visual Studio er anvendt til at udvikle software til systemets server.

AeroQuad Configurator v3.2

AeroQuad Configurator er anvendt til at kompilere og uploade bootloader og firmware til AeroQuad flight control board'et, samt kalibrere og monitorere dronens sensorer.

Atmel Studio 6

Atmel Studio er anvendt til at udvikle software til systemets embeddede platform(Arduino'en).

Eclipse Android Developer Tool v22.6.2

Eclipse ADT er anvendt til at udvikle software til systemets Android applikation.

PangoScrum

PangoScrum er et online Scrum-værktøj[11]. Dette værktøj har udgjort for den traditionelle anvendelse af Scrum med tavler og gule stickers.

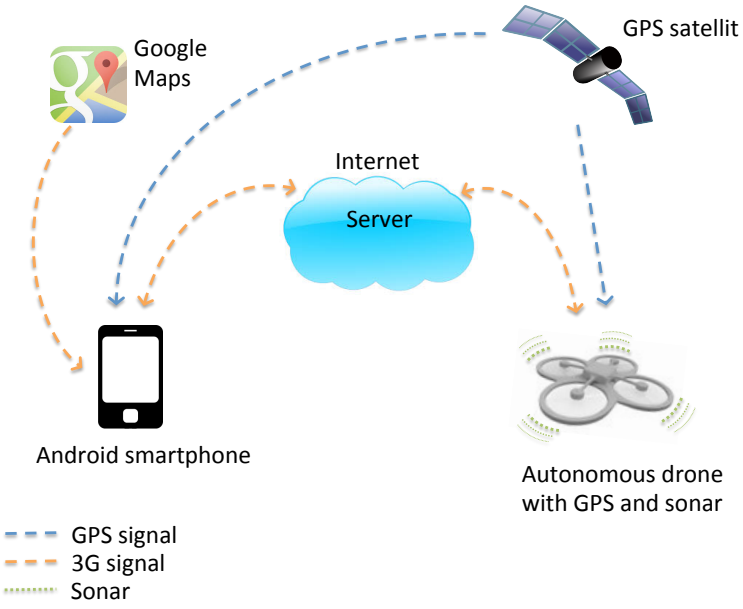
[11] http://pangoscrum.com/

Realisering af system 5

I dette kapitel beskrives realiseringen af ADC-systemet. Indledningsvis vil den overordnede systemarkitektur blive beskrevet. Herefter beskrives selve dronen og dens softwareopbygning. Efterfølgende uddybes realiseringen af Android applikationen, og endelig vil serverens opbygning og implementering blive forklaret. Det vil igennem hele kapitlet blive beskrevet, hvilke tanker og valg der er gjort under udviklingsprocessen.

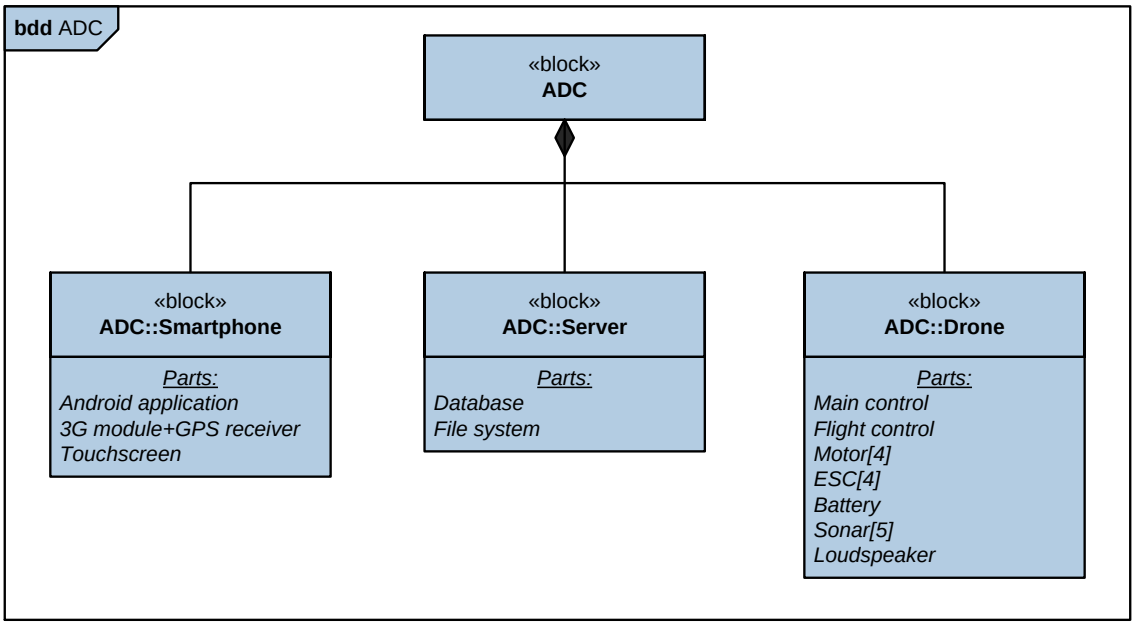
5.1 Systemarkitektur

På figur 5.1 ses en illustration af det system, der er udviklet under projektforløbet. Billedet viser systemets grundelementer, og hvordan de kommunikerer med hinanden. Figuren illustrerer, hvordan smartphone applikationen via 3G henter et interaktivt kort fra Google Maps. Derudover bruger app'en 3G til tovejskommunikation med serveren over internettet. Serveren kommunikerer på samme måde med dronen over 3G. Dronen er desuden forsynet med sonarsensorer for at undgå kollisioner. Både drone og smartphone anvender GPS til positionsbestemmelse.



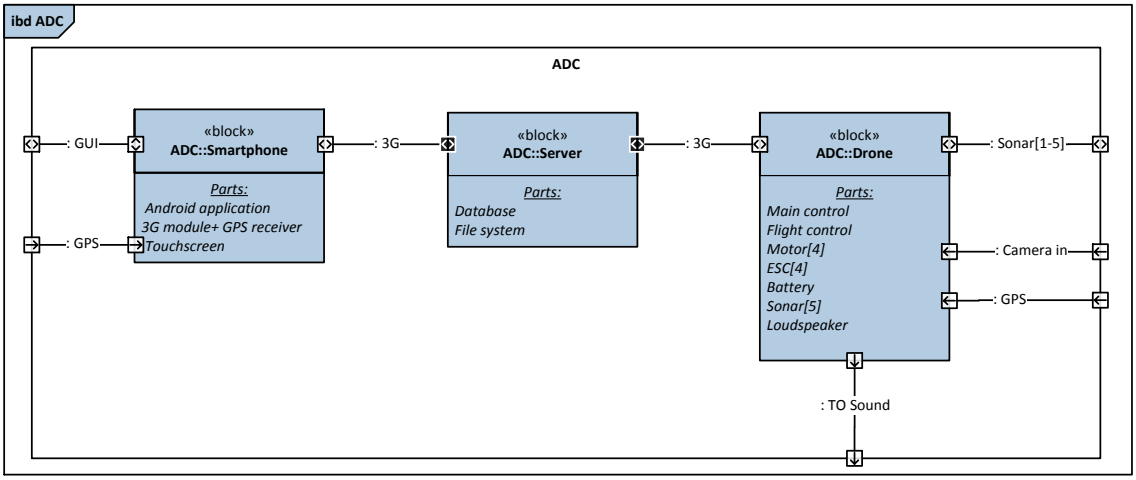
Figur 5.1. Systemoverblik.

På figur 5.2 ses det overordnede bdd for systemet. Figuren viser systemets overordnede blokke og bestanddele. Dette diagram danner grundlaget for et detaljeret hardwaredesign, der er beskrevet i kapitel 2 på side 5 i dokumentet "Arkitektur og design".



Figur 5.2. Block definition diagram for systemet.

På figur 5.3 ses det overordnede ibd for systemet. Figuren viser, hvordan systemets blokke fra figur 5.2 kommunikerer internt i systemet og eksternt til omverdenen[1]. De konjugerede in/out porte betyder, at det data der sendes ud fra den ene type in/out port, er input for den anden type in/out port og omvendt.

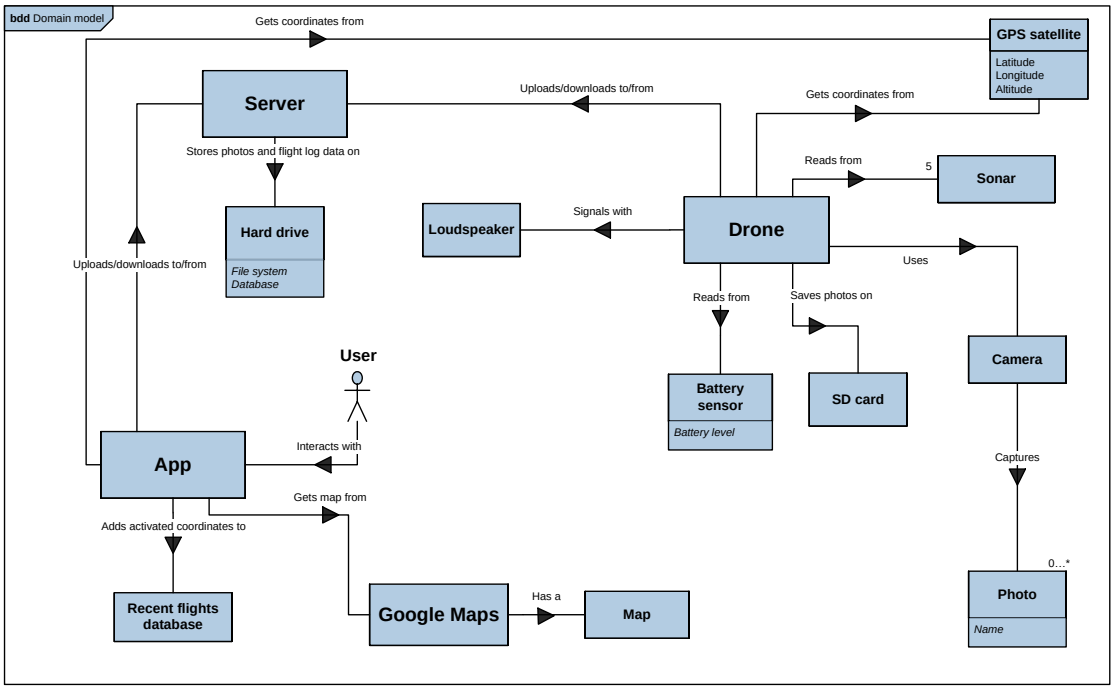


Figur 5.3. Internal block diagram for systemet.

[1]Se afsnit 2.3 på side 9 i dokumentet "Arkitektur og design" for et mere detaljeret ibd for systemet.

5.2 Domænemodel

Domænemodellen er et værktøj, der bruges til at gå fra kravspecifikation til softwarearkitektur/-design. Modellen er udarbejdet sammen med kunden, og faciliterer overgangen fra krav til strukturering og design. Domænemodellen identificerer objekter fra use casene beskrevet som softwareobjekter, og beskriver softwareobjekternes interne relationer. Domænemodellen for systemet ses på figur 5.4.



Figur 5.4. Domænemodel over det samlede system.

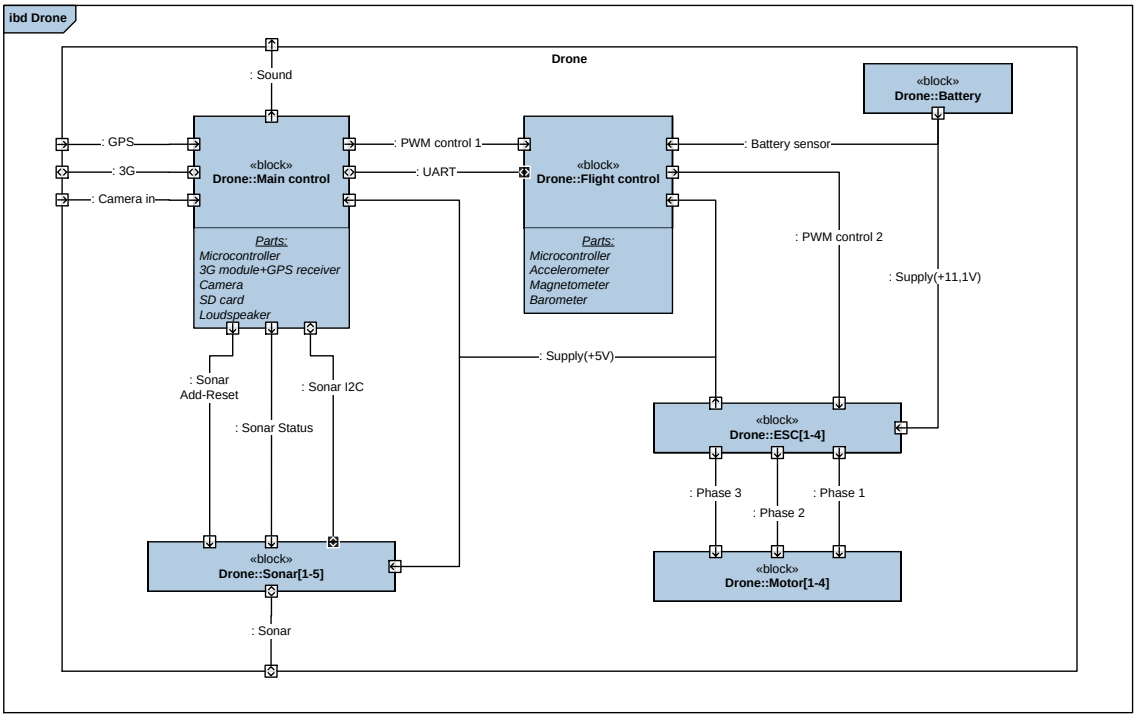
5.3 Design og implementering af drone

I dette afsnit beskrives design og implementering af dronen. Dronen er implementeret med et AeroQuad Cyclone ARF kit[2]. På figur 5.5 ses det overordnede ibd for dronen. Blokkene *Main control* og *Flight control* er bygget op omkring hhv. en Arduino Mega2560[3] og et AeroQuad 32 v2 flight control board[4].

[2] http://www.aeroquadstore.com/AeroQuad_Cyclone_ARF_Kit_p/aqarf-001.htm

[3] Databladet kan ses på den vedlagte CD med filnavnet "ATmega2560 (doc2549).pdf"

[4] Databladet kan ses på den vedlagte CD med filnavnet "Flight_control_board_µC_STM32F405VGT6-datasheet.pdf"

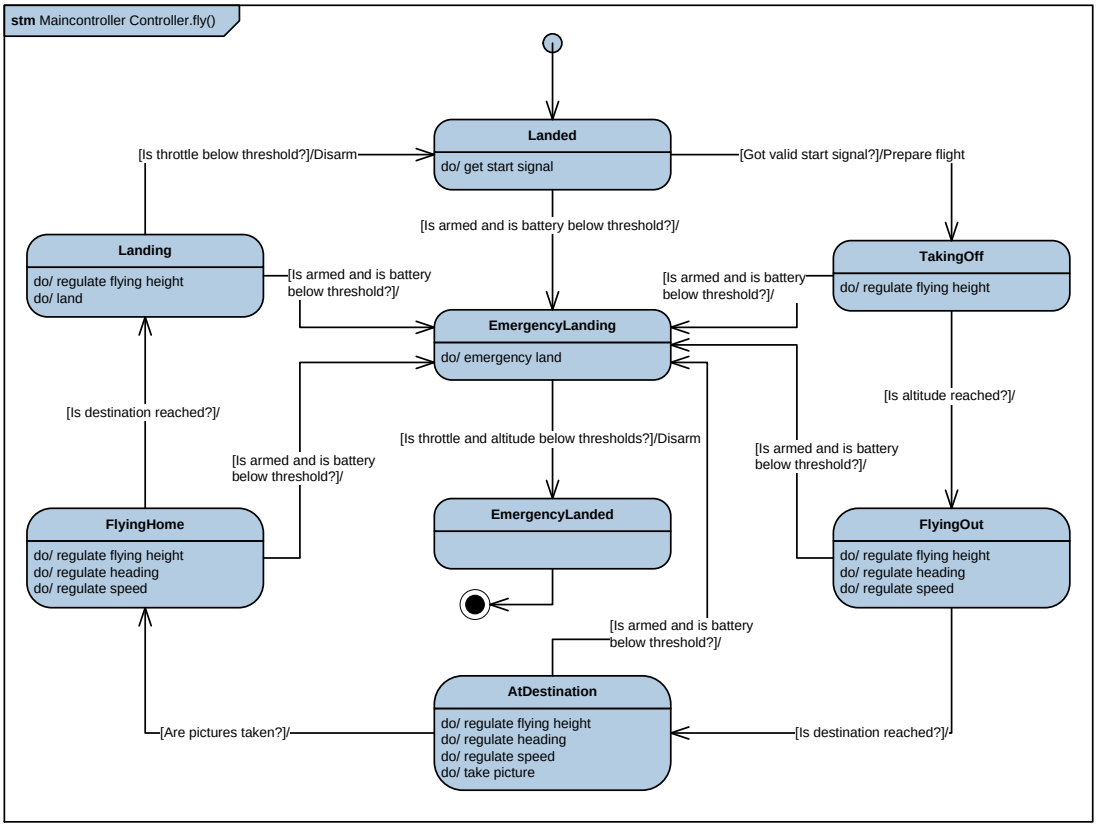


Figur 5.5. Internal block diagram for dronen.

5.3.1 Opbygning af dronesoftware

Dronens *ADCDroneController*-klasse står for at binde samtlige droneklasser sammen, og er den eneste klasse, der anvendes fra dronens main. Klassen har tre public metoder, hvoraf kun to anvendes ved almindelig drift af dronen. Disse to metoder er *sendLog* og *fly*. Metoderne kaldes fra dronens main loop, der styres af et $205ms$ timer interrupt. *sendLog*-flaget sættes cirka hvert 30. sekund, hvorefter *sendLog*-metoden sender en log til serveren. *fly*-flaget sættes cirka hvert 205. millisekund, hvorefter *fly*-metoden styrer dronens bevægelse. *fly*-metoden er bygget op om en state machine, som det ses på figur 5.6[5]. Afhængig af dronens aktuelle status, vil dronen befinde sig i en af de otte states. For hver state er der defineret en specifik adfærd for dronen.

[5]For kodespecifik beskrivelse af state machine'en samt beskrivelse af de enkelte states se afsnit 3.7.2.3 på side 109 i dokumentet "Arkitektur og design".



Figur 5.6. Tilstandsdiagram for *ADCDroneController*-klassens fly-metode.

5.3.2 PID-regulering

Til at styre den autonome flyvning er der designet fire PID-regulatorer. De fire regulatorer har forskellige PID-parametre, sætpunkter, inputs og formål, og de vil derfor blive beskrevet hver for sig i de følgende underafsnit. Til alle reguleringerne er den samme PID-klasse anvendt, hvorfor denne beskrives først.

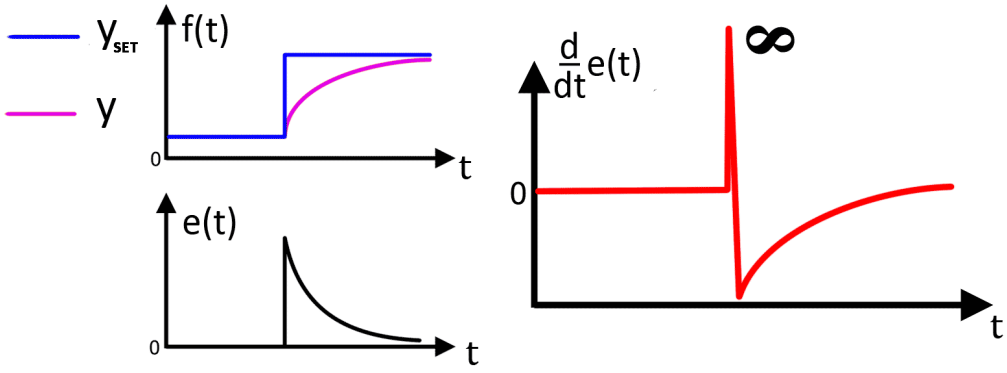
5.3.2.1 PID-klasse

Klassen *PIDRegulator* står for at styre motorernes hastighed ud fra de målte sensorværdier. Klassens primære funktionalitet ligger i metoden:

```
double updatePID(double error, double position)
```

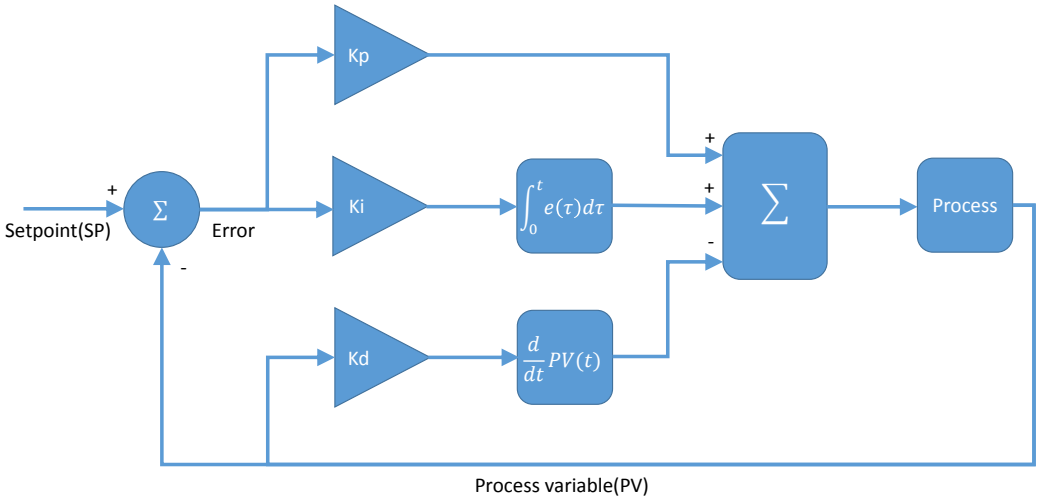
Metoden tager fejlen(*error)* og positionen(*position)* for den aktuelle regulator som input-parametre, og returnerer et output, der senere bruges som hastighedsparameter i en af *ADCDroneController*-klassens flyvemetoder.
Fejlen i regulatorerne føres ind i regulatorens P- og I-led. D-leddet kan implementeres på to forskellige måder: Error feedback eller rate feedback. Ved implementering af error feedback anvendes fejlen som input til D-leddet. Ved implementering af rate feedback anvendes

procesvariablen som input til D-leddet.
Ulempen ved implementeringen af error feedback er et fænomen kaldet "derivative kick". Derivative kick opstår, når sætpunktet pludseligt ændres betydeligt, da en ændring i sætpunktet betyder en direkte ændring i fejlen. D-leddet beregner hældningen af fejlen, og vil derfor ved en øjeblikkelig ændring i sætpunktet give et potentielt uendeligt stort kick til processen. Dette er illustreret på figur 5.7.



Figur 5.7. Derivative kick.

For at undgå derivative kicks er alle PID-regulatorer i dette projekt implementeret med rate feedback, som vist på figur 5.8.



Figur 5.8. Rate feedback.

Regulering af throttle

Throttle-reguleringen[6] bestemmer dronens højde ved at flytte dronen langs z-aksen. Denne regulator anvender dronens barometermålinger, der giver dronens højde i meter, ift. det

[6]For beskrivelse af de fire flyvetermer throttle, yaw, pitch og roll se http://uav-society.blogspot.dk/2014/06/quadcopter-mechanics.html.

barometriske tryk dronen sidst er blevet tændt under.

Regulering af yaw

Yaw-reguleringen bestemmer dronens kurs ved at rotere dronen om z-aksen[7]. Denne regulator anvender magnetometer- og accelerometermålinger.

Regulering af pitch

Pitch-reguleringen bestemmer dronens fart i fremadretningen ved at rotere dronen om y-aksen. Denne regulator anvender dronens afstand til destinationen målt i meter[8].

Regulering af roll

Roll-reguleringen roterer dronen om dens x-akse. Denne regulator anvender dronens afstand til destinationen målt i meter[8]. Roll-regulatoren er ikke implementeret i denne prototype.

5.3.3 Dataoverførsel

Til at overføre data mellem server og drone er der anvendt et 3G+GPS shield fra Cooking-Hacks[9]. En af de udslagsgivende grunde til valget af netop dette shield er implementering af applikationslagsprotokollen HTTP. Ved at anvende AT-kommandoer er det muligt at sende HTTP-telegrammer til en webserver. Selve HTTP-protokollen er request-response baseret, hvilket betyder, at en klient sender en forespørgsel til serveren, hvorefter serveren sender et svar tilbage. Klienten kan enten forespørge data fra, eller sende data til, serveren. Både forespørgsel og svar er bygget op af en header og en body. Header'en indeholder oplysninger om afsenderen og om forespørgslens karakteristika, og body'en indeholder selve den data, der forespørges fra, eller sendes til, serveren.

Til at facilitere dataoverførslen mellem server og drone er *ServerIF*-klassen designet. Som udgangspunkt for funktionaliteten i *ServerIF*-klassen er der anvendt tutorial sketches fra Cooking-Hacks' hjemmeside[10]. Her ligger simple Arduino sketches til at demonstrere shield'ets funktionaliteter, herunder HTTP. For at spare tid er det valgt at tage udgangspunkt i dette og portere koden til *ServerIF*-klassen. Derudover er der implementeret hjælpemetoder til at serialisere og deserialisere XML. I dokumentet "Arkitektur og design" afsnit 3.6.4.1 på side 84 findes yderligere information om implementering af serialisering og deserialisering.

ServerIF-klassen har tre public metoder, hvori klassens hovedfunktionalitet ligger:

[7] For beskrivelse af x-, y- og z-akser for dronen se http://uav-society.blogspot.dk/2014/06/quadcopter-mechanics.html.

[8] For detaljer om beregning af afstand til destination se kapitel 3.7.2.6 på side 116 i dokumentet "Arkitektur og design".

[9] Online elektronikforhandler for hobbyingeniører

[10] http://www.cooking-hacks.com/documentation/tutorials/arduino-3g-gprs-gsm-gps

```
StartSignalStruct getStartSignal()
void initiateServerConnection()
void sendLogPackage(GPSDataStruct gpsData, int heading, int droneStatus)
```

Til at hente startsignalet er den første metode, *getStartSignal*, implementeret. Denne metode kalder den private metode *signalGETRequest*. Heri oprettes forbindelsen til serveren og HTTP-forespørgslen på startsignalet afsendes. Herefter blokeres applikationen på dronen, og venter på svar fra serveren. Når svaret på forespørgslen er modtaget, isoleres body'en. Ved hjælp af metoden *deserializeHttpContent* deserialiseres indholdet, og returneres i en *StartSignalStruct* indeholdende koordinatsæt, tid og dato.

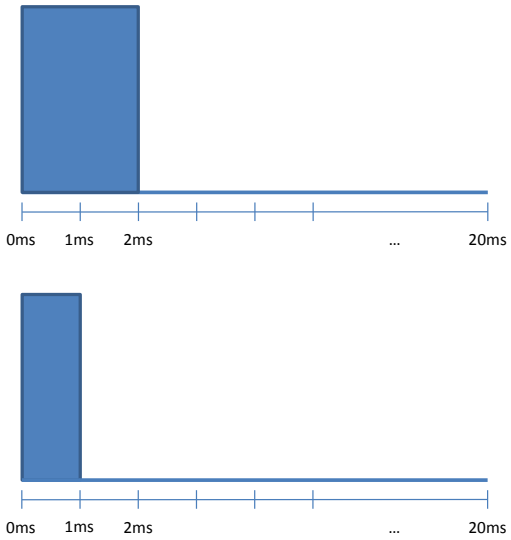
Til at uploade logposter anvendes de to sidste metoder. I første iteration af denne funktionalitet var upload af logposter designet på samme måde som download af startsignal og dermed blokerende. Da styringen af dronen er implementeret ved brug af regulering, var det nødvendigt at anvende et andet design. Den valgte løsning er en interrupt-styret state machine.

Ved at *sendLog*-flaget sættes højt, initieres forbindelsen til serveren med metoden *initiateServerConnection*. Efter forbindelsen er etableret, uploades der en logpost med den private metode *sendLogPackage*. Denne metode danner en logpost ved brug af de medsendte parametre for straks at uploade den til serveren. I dokumentet "Arkitektur og design" underafsnit 3.7.2.4 forefindes yderligere dokumentation for denne implementering.

5.3.4 Motor driver

For at kunne styre dronen med Arduino'en er det nødvendigt at vide, hvordan en transmitter/receiver styrer dronen, da Arduino'en essentielt skal overtage dennes rolle. Når dronen styres med en transmitter/receiver, forbindes receiver'en til AeroQuad (AQ) board'ets fire styrepins. Styrepinsene kontrollerer hastigheden af dronens tre rotationsakser: Pitch, yaw og roll, samt motorhastigheden: Throttle. Det er signalet fra receiver'en til disse pins, Arduino'en skal imitere. Signalet følger PWM RC-standarden[11], hvor selve styringen sker med et PWM-signal med en on-tid på 1-2ms, en periodetid på 20ms og dermed en frekvens på 50Hz, som det ses på figur 5.9:

[11] http://en.wikipedia.org/wiki/Servo_control



Figur 5.9. RC PWM-protokol.

Denne type signal sendes til hver af de fire styrepins på AQ board'et, hvor on-tiden mellem 1 og 2ms omsættes til en værdi, der bestemmer hastigheden. En on-tid på 1ms svarer til den laveste hastighed, og en on-tid på 2ms svarer til den højeste.
Til at konstruere dette PWM-signal på Arduino'en, benyttes to af Arduino'ens 16-bit timere. Se i dokumentet "Arkitektur og design"afsnit 3.7.2.1 på side 105 hvorledes timerne opsættes, for at kunne generere dette signal.
For at kontrollere hastigheden på dronen, er kun tilbage at styre duty cyclen på det genererede PWM signal, så on-tiden ligger mellem 1 og 2ms, alt efter hvilken hastighed der ønskes på de tre rotationsakser og motorhastigheden. Opsætningen af Arduino'ens timere og styringen af duty cyclen på de forskellige styrepins, er alt sammen pakket ind i *MotorDriver*-klassen, der også kan ses gennemgået i ovennævnte afsnit i dokumentationen.

5.4 Design og implementering af smartphone app

I dette afsnit vil udviklingen af Android applikationen blive beskrevet. Under implementeringen af applikationen er der arbejdet efter det agile udviklingsprincip *Make it work, make it right, make it fast*, der er beskrevet i afsnit 4.5.

Indledningsvis er der ud fra system- og opgavebeskrivelsen forsøgt at danne et overblik over app'ens funktionalitet og udseende. I dokumentet "Kravspecifikation" er der i afsnit 1.5("Grænsefladekrav") defineret en række krav til den grafiske brugergrænseflade på

app'en. App'en udgør det eneste egentlige interface til systemet. Ud fra de funktionelle og ikke-funktionelle krav samt grænsefladekravene er mock-ups'ene på figur 5.10 designet.

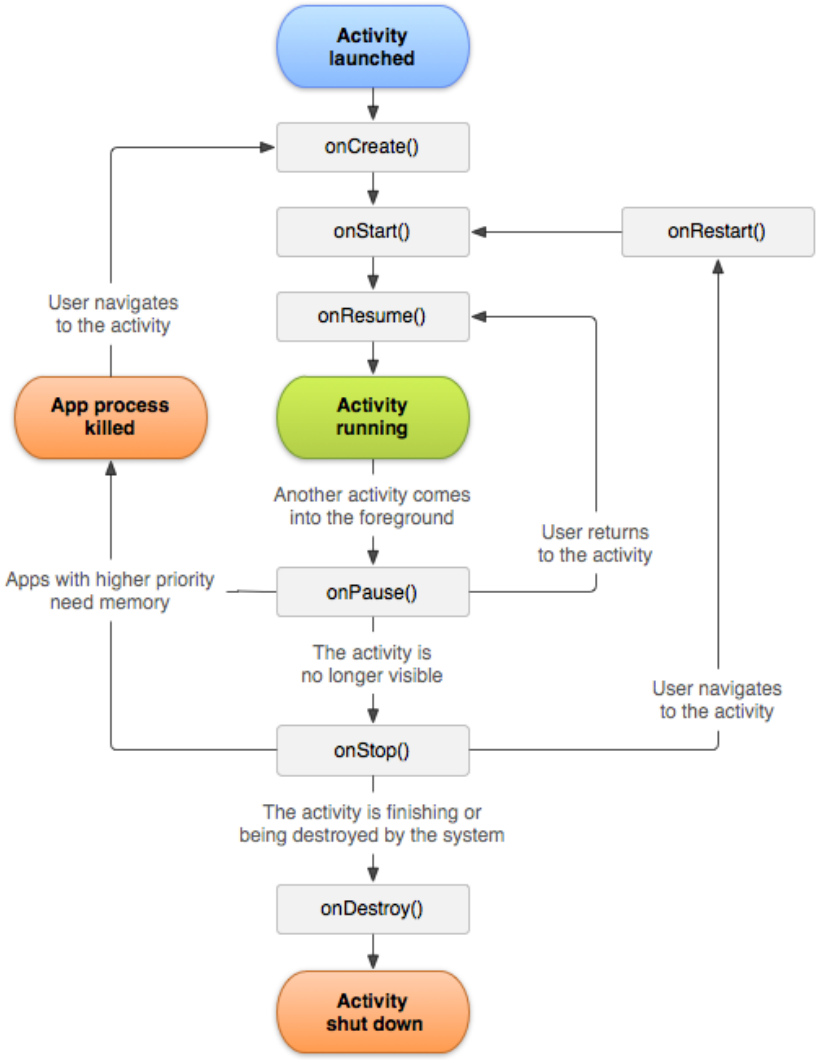
Figur 5.10. Udvalgte mock-ups af smartphone applikationen.

Ved at anvende mock-ups er den basale funktionalitet blevet hurtigt implementeret. For brugerenvenlighedens skyld er det forsøgt at minimere antallet af menulag.
Desuden er domæne- og applikationsmodellen anvendt til udviklingen af Android softwaren.

5.4.1 Persistent hukommelse

Det er valgt at gemme det sidste valgte koordinatsæt, så dette altid vil blive vist i koordinattekstfeltet i menuen "Sæt Koordinater". Til at implementere denne funktionalitet er *SharedPreferences* valgt. *SharedPreferences* er en klasse til at gemme simpel data i key-value pairs. *SharedPreferences*-funktionaliteten er i dette projekt pakket ind i en klasse med navnet *SimpleStorage*.

På figur 5.11 er Android aktiviteters livscyklus illustreret.



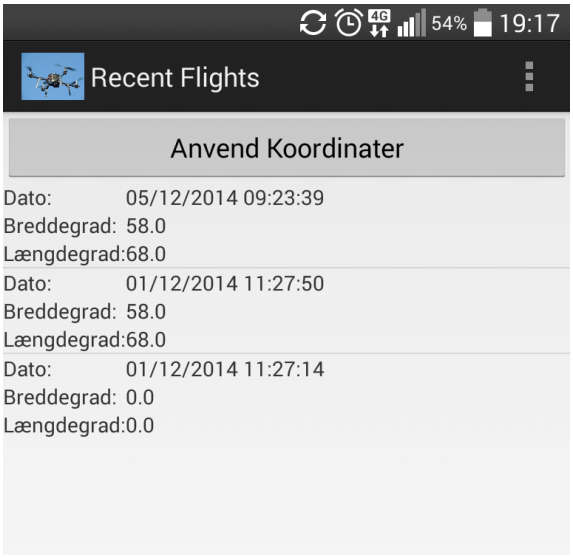
Figur 5.11. Illustration af en Android aktivitets livscyklus.

Denne livscyklus viser de forskellige tilstande en Android aktivitet kan befinde sig i, fra den oprettes og vises på skærmen, til den stoppes, og i sidste ende nedlægges. Mellem hver tilstand kaldes specifikke callback-metoder, hvor udvikleren kan afvikle vigtig funktionalitet i forhold til den kommende tilstand. I *SetCoordinatesActivity* indlæses værdierne i funktionen *onResume*. Når der navigeres væk fra denne aktivitet, gemmes værdierne fra koordinattekstfeltet i funktionen *onPause*. Dermed vil de sidst indtastede koordinater altid blive vist selvom app'en lukkes.

Til at gemme informationer om de seneste aktiverede flyvninger er det valgt at anvende en database. *SharedPreferences* blev overvejet, men valgt fra, idet det ikke ville være hensigtsmæssigt at gemme en så omfattende datamængde på denne måde. Valget er derfor faldet på *SQLite*, der er standarddatabasen i Android. *SQLite* er et naturligt valg, idet det er fuldt integreret i framework'et, og kan gemme flere værdier per id.

Databasen opdateres, når brugeren vælger at bekræfte aktivering af de valgte koordinater. Herved gemmes dato og tid, samt valgt længde- og breddegrad i databasen. Når brugeren

senere ønsker at vælge destinationskoordinat ved brug af seneste flyvninger, indlæses indholdet af databasen. På figur 5.12 ses et screenshot af denne liste.



Figur 5.12. Screenshot af "Recent Flights" i applikationen.

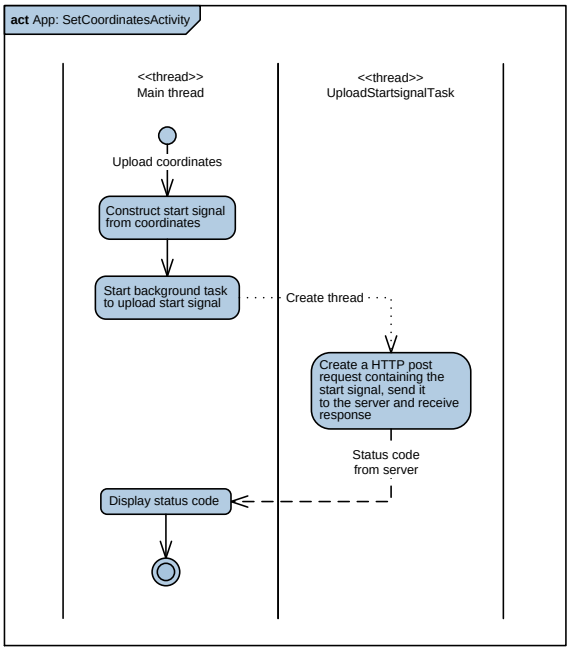
5.4.2 Kommunikation med server

Idet brugeren bekræfter aktivering af de valgte koordinater, uploades disse til serveren. Når denne form for netværkskommunikation foretages, er det yderst vigtigt, at det ikke foregår på hovedtråden, og bør derfor udføres ved anvendelse af en baggrundstråd. Årsagen til denne nødvendighed ligger i tråd- og procesmodellen for Android framework'et[12].

Ved opstart af en applikation oprettes en proces med én hovedtråd. Denne tråd er som udgangspunkt ansvarlig for afvikling af al kode. Herunder ligger ligeledes ansvaret for at opdatere den grafiske brugergrænseflade. Ved at afvikle beregningstung kode i hovedtråden, kan applikationen virke langsom og i sidste ende stoppe med at reagere. Derfor er der opstillet to simple regler for tråde i Android: "Blokér aldrig tråden der håndterer den grafiske brugergrænseflade" samt "Opdater kun den grafiske brugergrænseflade i hovedtråden"[12].

I projektet er det forsøgt at overholde begge regler ved at anvende baggrundstråde til al netværkskommunikation. Til at hente billeder fra serveren er der implementeret en *DownloadAsyncTask*-klasse i server-interfacet, der, som navnet antyder, arver fra *AsyncTask*-klassen. Derudover er der implementeret bundne services til at facilitere upload af startsignal samt download af flight log. I hver service startes en *AsyncTask* til at udføre netværksaktiviteten i baggrunden. Ved at anvende designet med services fortsættes download eller upload, selvom aktiviteten ikke er synlig. Hvis netværkskommunikationen færdiggøres, mens aktiviteten ikke er synlig, orienteres aktiviteten herom af servicen, når der vendes tilbage til aktiviteten. Et eksempel på den flertrådede kommunikation er illustreret på figur 5.13.

[12] http://developer.android.com/guide/components/processes-and-threads.html

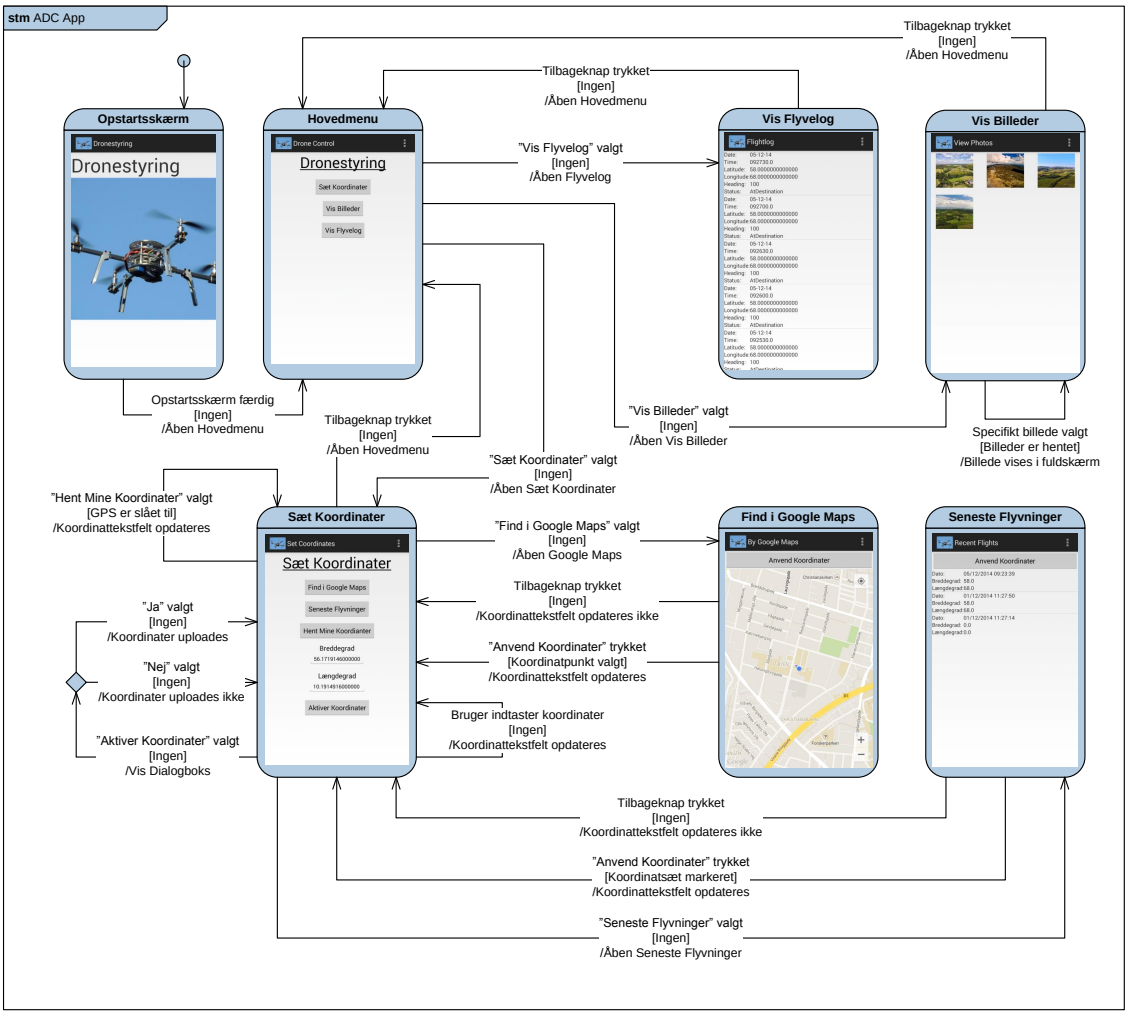


Figur 5.13. Aktivitetsdiagram for upload af startsignal.

5.4.3 Den færdige app

Det er forsøgt at give et overblik over den færdige app på figur 5.14[13]. Figuren viser et tilstandsdiagram for den implementerede app, og illustrerer, hvordan brugeren kan navigere rundt i app'en.

[13]For udgave, se dokumentet "Statemaskine_App_total.pdf" på den vedlagte CD.

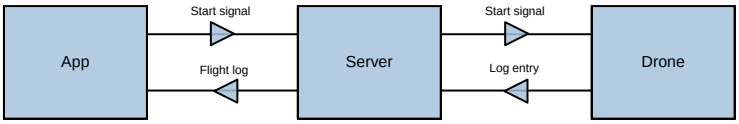


Figur 5.14. State machine for anvendelse af app'en.

5.5 Design og implementering af server

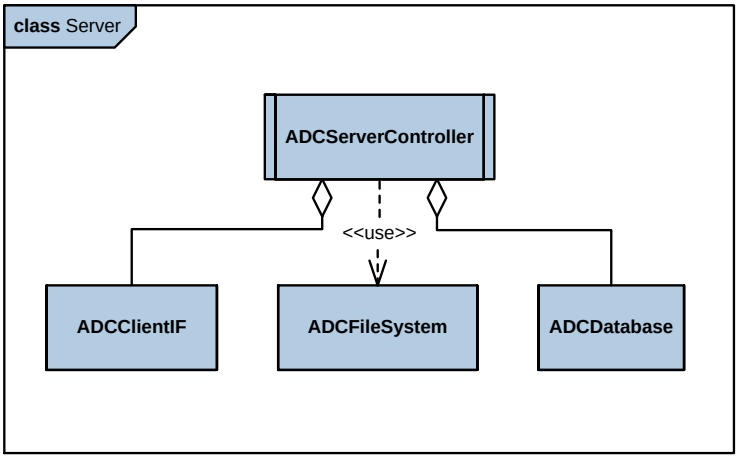
I dette afsnit gennemgås designet og implementeringen af serveren. Serveren er designet og implementeret som én blok bestående af selve serveren og en database på en SQL-server. Formålet med serveren er at agere bindeled imellem app'en og dronen ved at gøre forskellige typer data tilgængelig for begge klienter. Serveren benytter TCP som transportlag, og kommunikerer med klienter gennem HTTP-protokollen[14], hvilket vil sige, at serverens interface til de to klienter er ens. App og drone ses dog stadig som forskellige typer klienter fra serverens side, og serveren modtager data fra den ene type klient, der afsendes til den anden type klient og omvendt. På figur 5.15 ses et simpelt diagram over data flow'et mellem de tre systemdele.

[14] http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol



Figur 5.15. Data flow'et mellem systemets tre overordnede dele.

Serveren modtager et startsignal, når brugeren igangsætter en flyvning fra app'en. Dette startsignal sendes til dronen, når den tændes. Startsignalet indeholder oplysninger om dronens destination. Under en flyvning modtager serveren logposter fra dronen. Brugeren kan hente logposterne i form af en samlet flight log fra app'en, så dronens aktivitet kan følges under en flyvning.
På figur 5.16 ses et simplificeret klassediagram for serveren[15].



Figur 5.16. Klassediagram for serveren.

Serveren benytter både filsystemet og en database til at gemme data i. Startsignaler fra app'en gemmes i serverens filsystem, mens logposter, der modtages fra dronen under en flyvning, gemmes i en database på SQL-serveren.

5.5.1 Håndtering af forespørgsler

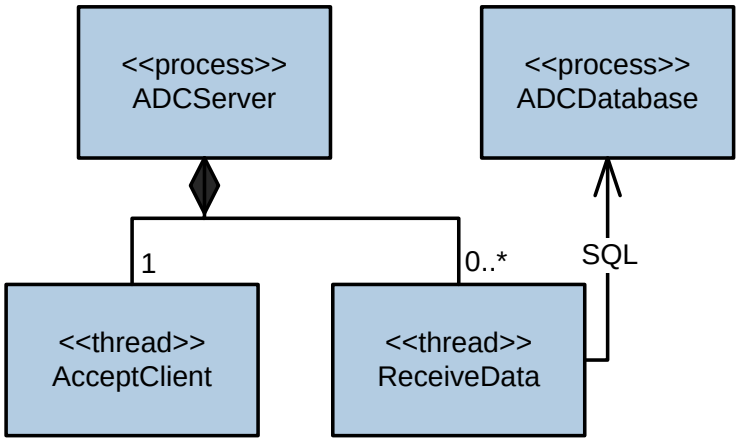
Som tidligere beskrevet, benytter serveren HTTP-protokollen til at kommunikere med klienterne i systemet. For at håndtere HTTP-forespørgsler fra klienter, er der implementeret deserialisering af HTTP header'en. Hermed kan oplysninger om HTTP-forespørgslen udtrækkes, og håndteringen kan gennemføres ved at hente eller gemme det forespurgte data, alt efter forespørgslens natur.
Der er i realiseringen af serveren lagt vægt på, at modtaget data, og data der skal sendes, bliver formatvalideret, så begge klienter oplever samme dataformater. Dette sker igennem

[15]Se afsnit 3.3.2.4 på side 45 i dokumentet "Arkitektur og design" for et mere detaljeret klassediagram for serveren.

XML-serialisering og -deserialisering, hvor der benyttes serialisérbare C#-klasser til at indeholde XML-information modtaget fra klienter. Klasseserialiseringen er en verificering i sig selv, og medfører, at dataformater er konsistente og valide, når en forbindende klient skal bruge data, eller data fra en klient skal gemmes[16].

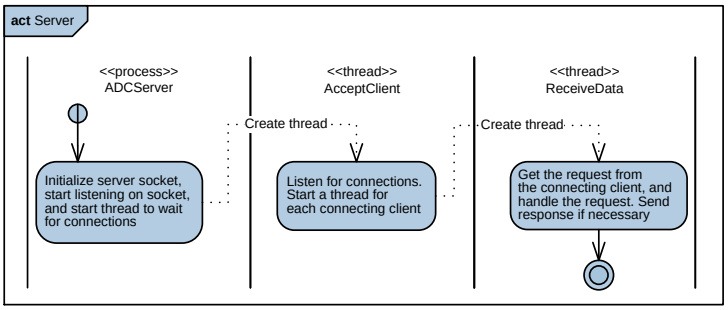
5.5.2 Trådhåndtering af klienter

Det er i flere situationer nødvendigt, at begge klienter kan tilgå serveren samtidigt, hvorfor der benyttes tråde til at håndtere forbindende klienter på serveren. På figur 5.17 ses et overblik over de tråde og processer, der kører på serveren.



Figur 5.17. Processer og tråde på serveren.

Serveren benytter én tråd til at lytte efter klienter(*AcceptClient*) og én tråd for hver forbindende klient(*ReceiveData*) til at modtage og håndtere forespørgsler fra klienten. Håndteringen består i at deserialisere HTTP-headeren, og gemme/hente data i/fra filsystemet eller databasen alt efter forespørgslens indhold. På figur 5.18 ses et forsimplet aktivitetsdiagram over serverens runtime-adfærd mht. håndtering af klientforbindelser i tråde[17].



Figur 5.18. Serverens runtime-adfærd.

[16]Se mere om serialisering/deserialisering af data i afsnit 3.6 Deployment View, underafsnit 3.6.4.2 på side 85 i dokumentationen

[17]Se afsnit 3.4.1.3 på side 67 i dokumentationen, for et mere detaljeret aktivitetsdiagram for serverens klienthåndtering.

Der kan altså være flere klienter forbundet til serveren på samme tid, da de bliver håndteret i hver deres tråd. Dette er en væsentlig funktionalitet i systemet, da klienter ellers kunne være nødsaget til at vente på andre klienthåndteringer, før de selv kunne blive betjent.

Test 6

6.1 Procedure

For hver del i projektet er der udført tests for at garantere korrekt funktionalitet, men ligeledes for at sikre, at den gradvise integration af de enkelte moduler bliver så gnidningsfri som mulig. Ved at lave omfattende tests tidligt i projektet opdages eventuelle fejl og mangler. Dette kan i sidste ende betyde, at man undgår et eventuelt tidsspild på at udbedre disse. I projektet er der udført følgende typer af tests:

- **Enhedstest:** Her bliver hver enhed/modul testet for sig. Der testes på den basale funktionalitet, og eventuelle krav fra kravspecifikationen verificeres. I dette projekt har det blandt andet resulteret i følgende enhedstests:
  - Test af serialisering og deserialisering i forbindelse med modtaget/sendt data via HTTP-metoder på dronen.
  - Test af PID-regulatorer på drone ved simulering af sensorværdier.
  - Arduino'ens egenskaber til at generere korrekte PWM-signaler til flight control board'et er blevet testet.
- **Integrationstest:** Her bliver systemets enheder sat sammen for at teste deres indbyrdes grænseflader. Det har resulteret i følgende tests:
  - Test af kommunikation mellem server og drone samt server og smartphone applikation.
  - Test af PID-regulatorernes evne til at ændre motorernes hastighed.
- **Accepttest:** I denne del testes det endelige produkt i henhold til accepttestspecifikationen. Dette er en sluttest, der ofte foretages med kunden tilstede, således begge parter er enige om resultatet.

6.2 Resultater

Ved implementeringsfasens afslutning er følgende resultater opnået som led i udførelsen af accepttesten.

Der er udviklet en Android applikation, der kan anvendes til at vælge koordinater på de fire specificerede måder. Det er ligeledes muligt at uploade det valgte destinationskoordinat som startsignal. Derudover er der implementeret funktionalitet til visning af flight log samt billeder.

Den autonome dronestyring er implementeret i det omfang, at dronen kan hente et startsignal og omsætte dette til en destination. Herefter kan dronen lette autonomt fra jorden, beregne

kursen mod destinationen, og derefter begynde flyvning mod destinationen. Selve flyvningen mod destinationen er implementeret, men er ikke færdigoptimeret. Det har derfor ikke været muligt at godkende test case 7 og 8, der tester flyvning til og fra destination. Funktionaliteten er implementeret, men med det nuværende design og de anvendte sensorer, er det ikke muligt at udføre en godkendt flyvning fra et startpunkt til en destination.
Det er ydermere testet, at dronen er i stand til at uploade logposter med et fast interval under en simuleret flyvning, dvs. en flyvning hvor dronens motorer ikke er forsynede. Jf. kravspecifikationen er funktionerne til fotografering og upload af billeder, nødlanding samt undvigemanøvre ikke forsøgt implementeret.

Samtlige test cases for de implementerede use cases vedrørende serveren er godkendte. Det er muligt at modtage startsignal fra app, sende startsignal til drone, og sende billeder og flight log til app. Derudover er det muligt at håndtere forespørgsler fra app'en samtidig med at logposter modtages fra dronen. De use cases der ikke forventedes implementeret, er der ikke udført test cases for.

Det samlede system blev testet på et åbent område, med mulighed for at overtage styringen over dronen til hver en tid. Dronen blev tændt uden et startsignal eksisterende på serveren, hvorefter et startsignal sendtes til serveren fra app'en. Ved modtaget startsignal signalerede dronen en forestående flyvning vha. dronens lydgivere. Herefter påbegyndtes flyvning, hvor dronen først lettede til den specificerede højde for derefter at indstille kursen til destination. Efter indstillet kurs påbegyndtes flyvning mod destination. Her var det nødvendigt at overtage styringen af dronen, da selve flyvningen ikke var færdigoptimeret.

6.3 Diskussion

I dette afsnit diskuteres resultatet af de implementerede systemdele: Drone, app og server, og der gives forslag til eventuelle forbedringer.

6.3.1 Diskussion af drone

6.3.1.1 Reguleringsflag

For at opnå konstant reguleringsinterval og dermed konstant tidsforsinkelse i systemet, er der implementeret et reguleringsflag i dronens main. Flaget sættes i et timer interrupt, og reguleringen foregår således hvert $205.ms$. Flaget giver dog ikke nogen garanti for et konstant reguleringsinterval, da der ikke nødvendigvis tjekkes på flaget, så snart det er sat. Derfor bestod første implementering af reguleringen i at kalde den fra selve interrupt'et. Dette resulterede imidlertid i, at micro controlleren stoppede. Årsagen til dette er endnu ukendt, men en mulig grund kan være, at reguleringen læser fra den serielle port, der også anvender interrupts. Når et interrupt kaldes, blokeres alle andre interrupts, og dette har muligvis forårsaget problemer. Ifølge tests har den implementerede løsning givet fornuftige resultater[1], og implementeringen er derfor midlertidig godkendt.

[1] Se afsnit 2.2 på side 11 i dokumentet "Test" for detaljer om testresultaterne.

6.3.1.2 PID-tuning

Til at tune de anvendte PID-regulatorers parametre findes der forskellige fremgangsmåder. I udviklingen af dette produkt er der anvendt en brute force metode, hvor parametrene ændres, hvorefter dronens adfærd testes. Med et system af denne kaliber er dette en tidskrævende proces, da parametrene af sikkerhedsmæssige årsager kun kan ændres lidt af gangen. Andre mere gængse metoder er simulering af systemet og Ziegler-Nichols[2].
Ved simulering af systemet laves der en overføringsfunktion for dronen ved at åbne feedback loop'et og derefter analysere på systemets respons på et step input. Overføringsfunktionen danner dermed grundlag for en simuleringsmodel, hvorpå PID-parametrene kan tunes. Det ville være muligt at lave modeller til dronens yaw-, pitch- og roll-reguleringer, da systemet for disse bevægelser har samme forhold uanset bevægelsesretningen. For højde-/throttle-reguleringen er der forskel på bevægelse med eller mod tyngdekraften. Udviklingsteamet har ikke verificeret en måde at løse denne problemstilling på. Da højderegulerингen er den mest kritiske regulering, er der ikke brugt kræfter på fintuning af de andre tre regulatorer, før denne regulering virker.
Ved brug af Ziegler-Nichols-metoden anvendes der først en ren P-regulator. Forstærkningen i P-leddet skrues op, indtil systemet begynder at oscillere, og resten af tuningen tager herefter udgangspunkt i denne forstærkning og oscilleringsperiode. Af sikkerhedsmæssige årsager har udviklingsteamet endnu ikke testet denne metode.

6.3.1.3 Højderegulering

I de udførte tests anvender dronen udelukkende barometermålinger til at regulere flyvehøjden. Efter flere testflyvninger har det vist sig, at dette formentlig ikke er en gangbar løsning pga. støjfyldte sensorværdier[3]. Barometeret er følsomt over for luftstrømninger, lydbølger, lys og temperatur, men føres sensorværdierne igennem et moving average filter[4] på nogle sekunder, kan værdierne midles til et mere stabilt udtryk for den faktiske højde. Implementeringen af et sådant filter ville dog betyde, at systemet ville blive for langsomt til at reagere hurtigt nok på højdeændringer.
En løsning på dette problem kunne være at anvende dronens accelerometer. Dobbeltintegreres accelerometerværdierne, fås dronens position, $r(t)$:

$$a(t) \tag{6.1}$$

$$v(t) = v_0 + \int_0^t \mathrm{a}\,\mathrm{d}t \tag{6.2}$$

$$r(t) = r_0 + \int_0^t \mathrm{v}\,\mathrm{d}t \tag{6.3}$$

Anvendes denne position til den kortsigtede højderegulering sammen med en midtvejskorrektion fra de midlede barometerværdier, kunne der muligvis opnås en bedre højderegulering.

[2] http://en.wikipedia.org/wiki/Ziegler-Nichols_method
[3] Se afsnit 1.2.5 på side 5 i dokumentet "Test" for test af barometer.
[4] http://en.wikipedia.org/wiki/Moving_average

6.3.2 Diskussion af smartphone applikation

Den færdige udgave af ADC-app'en svarer i høj grad til det ønskede resultat jf. dokumentet "Kravspecifikation". Alle funktionaliteter er implementeret, og er godkendt i accepttesten. Dog er der stadig områder, hvor der er plads til forbedringer, hvis app'en skal kunne fungere som et kommercielt produkt. Her er der f.eks. tale om app'ens grafiske interface, som ikke er prioriteret i implementeringen, da funktionaliteten er vægtet højest.

Til at kommunikere med serveren er der for upload og download af hhv. startsignal og logposter implementeret bundne *Service*'s med baggrundstråde i form af *AsyncTask*'s. Til download af billeder er *AsyncTask*-klassen anvendt uden *Service*'s. Hvis brugeren vælger at minimere applikationen under download af billeder, er der med det implementerede design risiko for, at hentningen fejler. Download af billeder skal implementeres ligesom upload og download af hhv. startsignal og logposter, med bundne *Service*'s, for at løsningen er optimal. Der er imidlertid ikke lagt vægt på dette, da fotografering og upload ikke er implementeret på dronen, hvilket medfører, at download af billeder i app'en ikke er væsentlig for det implementerede system. En anden optimering vedrørende billeder i app'en ville være, at gemme de hentede billeder lokalt på telefonen og i øvrigt kun hente de nyeste billeder fra serveren. Dette ville mindske datatrafikken for app'en betydeligt.

Selvom der ikke er lagt stor vægt på det grafiske udseende for app'en, er der alligevel foretaget valg ifm. den grafiske implementering. Det er valgt hovedsageligt at implementere GUI'et med et statisk XML layout. Ved at anvende et statisk grafikdesign er det simpelt at definere "Presentation"-laget i "Three tier"- arkitekturen[5], der er benyttet under designet/implementeringen af app'en. I stedet for at programmere den grafiske brugerflade runtime, defineres den i et XML-dokument. I det anvendte IDE er det på denne måde muligt både at skrive og drag-n-drop'e GUI'et til app'en.

6.3.3 Diskussion af server

Realiseringen af serveren er resulteret i en funktionalitet, der langt hen af vejen er som ønsket for et færdigt system. Der er imidlertid elementer, der skal optimeres, hvis alle use cases, der vedrører serveren, skal implementeres. Håndtering af billeder fra dronen til serveren vil eksempelvis kræve optimering i klienthåndteringen, da der her er tale om betydeligt større datamængder i overførslerne. Store filer såsom billedfiler skal overføres i små dele over klient-socket'en. Dette kompliceres, når klienthåndteringen foregår asynkront, som den gør på serveren i kraft af trådhåndteringen af klienter. Der er i implementeringen af serveren påbegyndt et udkast til modtagelse af større filer, men dette er ikke færdigimplementeret, hvorfor det ikke er beskrevet andre steder i rapport og dokumentation. Der er ikke lagt vægt på dette aspekt af klienthåndteringen, da fotografering af billeder på dronen ikke forventedes implementeret.

Der er imidlertid lavet et fungerende udkast til overførsel af billeder fra serveren til app'en, da dette har været mindre kompliceret at implementere med den nuværende klienthåndtering.

[5] http://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture

Dette er også testet succesfyldt sammen med app'en, hvor visning af billeder også er implementeret, hvorfor denne test case[6] også er godkendt.

På serveren har udviklingsteamet selv implementeret al serialisering og deserialisering af HTTP headers i HTTP-svar og -forespørgsler. Dette har ikke været vitalt for kommunikation mellem server og klienter, da der i forvejen findes adskillige biblioteker til at håndtere netop dette aspekt af HTTP-kommunikation. Dog har det medført, at der på serveren er fuld kontrol over indholdet af forespørgsler fra klienter og svar til klienter. Dette har gjort det muligt selv at definere header- og body-indhold uden at inkludere oplysninger, der ikke bliver brugt af klienter/server. Serialiseringen/deserialiseringen er samtidig implementeret, fordi udviklingsteamet har villet tilegne sig viden om dette aspekt af server-klient kommunikation via HTTP. Dette har dog været på bekostning af tid og ressourcer, der kunne have været anvendt andre steder i projektet ved at benytte et eksternt bibliotek.

I forbindelse med kommunikation med dronen er der tidligt i udviklingsfasen truffet en beslutning om ikke at sende svar tilbage til dronen, når den sender data til serveren. Denne beslutning er taget, fordi al netværkskommunikation er tidskrævende for dronen. Jo flere netværksrelaterede operationer dronen skal foretage i hovedprogrammet, jo længere tid er der mellem hver gang dronen kan regulere sin flyvning. Ved ikke at sende svar tilbage til en klient, følges "request-response"[7]-modellen i HTTP-protokollen ikke. Dette har imidlertid været nødvendigt for at minimere netværksrelaterede operationer på dronen, og dermed optimere flyvningen. Der sendes stadig svar tilbage til app'en, således at serveren her følger HTTP-protokollens "request-response"-model.

[6] Se dokumentet "Test" afsnit "3.5.7 Test case 3: Visning af billeder".
[7] http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

7 Konklusion

Formålet med projektet har været, at levere en samlet løsning til et autonomt dronestyringssystem bestående af en Android applikation, en server samt en drone. Systemet skulle være brugervenligt og nemt at anvende for brugere uden erfaring med droneflyvning. Den implementerede prototype er tæt på det ønskede system, men kræver optimering, for at kunne kaldes fuldt ud funktionel.

Det er lykkedes at illustrere grundprincipperne i en prototype fungerende efter grundtankerne bag projektet. Der er således udviklet en fuldt ud funktionel Android applikation, hvorpå brugeren kan vælge koordinater på de fire specificerede måder og uploade disse som et startsignal til den implementerede server. På app'en har brugeren desuden mulighed for at downloade en flyvelog fra serveren, der viser dato og klokkeslæt, lokation, kurs og status for dronen. Endelig kan brugeren downloade og gennemgå billeder fra serveren.
Serveren kan modtage og håndtere samtlige implementerede forespørgsler fra både app og drone. På serveren er der implementeret en database og et interface til filsystemet til at lagre hhv. flyvelog, startsignal og billeder modtaget fra dronen. I denne prototype er kameraet ikke implementeret på dronen, hvorfor billedeoverførslen mellem drone og server heller ikke er implementeret.
Dronen er i stand til at downloade et startsignal fra serveren og ud fra dette startsignal at påbegynde en autonom flyvning vha. diverse sensoraflæsninger, heriblandt GPS og barometer. Sensorværdierne bruges til regulering af dronens bevægelser, og mere specifikt bruges barometerværdierne til regulering af dronens flyvehøjde. Det har vist sig problematisk at udføre fornuftig højderegulering pga. barometerets følsomhed, og derfor skal metoden til højdereguleringen genovervejes. Denne problematik kunne eventuelt løses ved en kombination af accelerometer- og barometeraflæsninger. Dronen uploader jævnligt logposter til serveren indeholdende brugerrelevante oplysninger om dronen. Ved ankomst på destinationen skulle dronen tage billeder. Denne funktionalitet har i udviklingsforløbet været nedprioriteret jf. de opstillede iterationer i "Kravspecifikation"-dokumentet, og der er derfor hverken implementeret kamera eller SD-kort. Af samme årsag er antikollisionssensorerne ikke implementerede.

Den implementerede funktionalitet er testet og verificeret vha. en accepttest.

Udviklingsteamet har anvendt en modificeret udviklingsmetode, der er tilpasset teamets kompetencer og tidligere arbejdsprocesser. Metoden er inspireret af Scrum, RUP og ASE-modellen, og har givet en effektiv projektstruktur, hvor der til ingen tider har været tvivl om arbejdsopgaverne i den nuværende eller forestående udviklingsfase. Artefakterne fra ASE-modellen og en udførlig tidsplan har medvirket til et afstresset og kontrolleret

arbejdsmiljø i teamet.

Som projektet skred frem, blev det udviklingsteamet mere og mere klart, at der var en ulogisk opdeling af use casene, og der er derfor blevet tilføjet en række use cases på et relativt sent tidspunkt i udviklingsforløbet. Disse tilføjelser krævede samtidig en revidering af implementeringsiterationerne. Denne erfaring har gjort teamet opmærksom på vigtigheden af struktureringen af use cases, og dette vil i fremtidige projekter bringes mere i fokus.

En af de væsentligste erfaringer teamet har gjort sig, er, hvor mange ressourcer der skal bruges på at udtænke og udføre tests for et system som det udviklede.