Projektrapport

AR.Drone control via DevKit8000

Titel:	Projektrapport
Forfattere:	Mads Havgaard Mikkelsen
	Kasper Kirkeby Jacobsen
Vejleder:	Torben Gregersen
Projektnummer:	12042
Institution:	Aarhus Universitet Ingeniørhøjskolen
Sider:	30
Dato:	19-12-2012

Versionshistorie

Ver.	Dato	Initialer	Beskrivelse
1.0	15/14 2012	09636	

Godkendelsesformular

Forfatter(e):	Kasper Kirkeby Jacobsen og Mads Mikkelsen
Godkendes af:	Torben Gregersen
Projektnummer:	12042
Dokument-id: (filnavn)	Projektrapport.pdf
Antal sider:	30
Kunde:	Aarhus Universitet Ingeniørhøjskolen

Sted og dato:

_____
Torben Gregersen

_____
Kasper Kirkeby Jacobsen (09588)

_____
Mads Havgaard Mikkelsen (09636)

Resumé

Denne rapport beskriver design og implementering af et system styring af en AR.Drone quadrocopter. Systemet kan styre en AR.Drone og vise billeder fra dens kamera.

Programmet til styring af AR.Dronen har følgende funktionalitet:

- Vise direkte billeder fra AR.Dronens kamera
- Styre AR.Dronen fra knapper på det grafisk bruger interface
- Styre AR.Dronen med en Xbox kontroller
- Vise flyveinformation såsom højdemeter, batteriniveau m.m. på det grafiske bruger interface. Disse flyveinformationer er også kaldet navigationsdata
- Autonomflyvning af AR.Dronen efter en linje på gulvet

Hardware platformen for systemet er en single board computer af modellen DevKit8000 og har følgende funktionalitet:

- Skærm til visning af programmets grafiske bruger interface
- WiFi enhed til at kommunikere med AR.Dronen
- USB porte til Xbox wireless reciever og USB WiFi dongle
- Linux styresystem

Projektet er designet på grundlag af en kravspecifikation, hvor 4+1 modellen er brugt til designet af program strukturen. Under implementeringsprocessen er SCRUM metoden benyttet.

Denne rapport dækker opbygningen af programmet, med design af funktionaliteten og klasserne, samt implementeringen af det udviklede program, Linux, WiFi og Xbox kontroller på et DevKit8000.

Systemet er designet til at køre på DevKit8000, men på grund af manglende performance derpå, er alle test foretaget fra en computer.

Abstract

This report describes the design and implementation of a control system for a AR.Drone quadcopter. The system is capable of controlling a AR.Drone and stream video from the drone to the display.

The program for controlling the AR.Drone has the following functionality:

- Show live video feed from the AR.Drones camera
- Control AR.Dronen from buttons on the graphical user interface
- Control AR.Dronen with an Xbox controller
- Display flight information such as altitude, battery level etc. on the graphical user interface. These flight information is also called navigation data
- Autonomous flight of the AR.Drone following a line on the floor

The hardware platform for the system is a single-board computer of the model DevKit8000 and has the following functionality:

- Screen for displaying the graphic user interface
- WiFi device to communicate with the AR.Drone
- USB ports for Xbox wireless receiver and USB WiFi dongle
- Linux operating system

The project is designed on the basis of a requirements specification, where 4 +1 model is used to design the program structure. During the implementation process the SCRUM method has been used.

This report covers the development of the program with the design of functionality and classes, as well as the implementation of the program developed, Linux, WiFi and Xbox controls on a DevKit8000.
The system is designed to run on DevKit8000, but due to lack of performance, all tests have been made on a laptop.

Indhold

1 Indledning

Projektet er udviklingen af en styring til en AR.Drone[Ref 5] fra et DevKit8000[Ref 6], hvor det er muligt, via. WiFi at streame direkte video fra AR.Dronen og styre den. AR.Dronen er en quadrocopter drone fra firmaet Parrot. DevKit8000 er en single board computer med touch skærm fra Embest. AR.Dronen bliver som regel solgt til folk, der vil styre den med mobile enheder der enten har iOS eller Android, da det er til de platforme Parrot har udgivet officielle styringsprogrammer. AR.Dronen bliver også brugt som legetøj af gadget interesserede og personer, der skriver deres egne programmer til den.

Ingeniørhøjskolen Aarhus har i et stykke tid haft en AR.Drone liggende, som ikke er blevet brugt. Det fik Torben Gregersen til at slå et projekt op, hvis formål var at undersøge mulighederne for udviklingen af en styring til AR.Dronen fra DevKit8000, der kan bruges til fremtidige projekter med AR.Dronen.

Projektet er bygget op omkring en kravspecifikation, der er udarbejdet af Torben Gregersen og gruppen i fællesskab. Kravspecifikationen har dannet grundlag for en foranalyse af de opstillede krav, for at vurdere, om der har været grundlag og mulighed for at løse dem.

Projektoplæggets krav var:

- Implementere en WiFi løsning på DevKit8000
- Styring af AR.Dronen over inputs fra DevKit8000's LCD touch skærm
- Vise video fra AR.Dronen på DevKit8000's LCD skærm
- Styre AR.Dronen vha. direkte video fra AR.Dronen

Senere i projektets forløb er disse krav blevet udvidet af gruppen, til også at inkludere:

- Styring af AR.Dronen vha. inputs fra en Xbox kontroller
- Autonom styring [1]af AR.Dronen efter en linje på gulvet

Designet af projektets software er bygget op omkring en 4+1 view model, der indeholder forskellige stadier af projektets forløb. Projektets stadier i 4+1 view modellen har været:

- Logical view
- Process view
- Deployment view
- Implemention view

Der er i alle stadierne er taget udgangspunkt i use cases. Til implementeringen af designet er SCRUM metoden blevet brugt.

AR.Dronen's interne styringssoftware og hardware er et lukket system og kan derfor ikke om programmeres. AR.Dronen kommer med en API og protokol beskrivelser, hvilket giver muligheder for at skrive eksterne programmer til at styre den. Parrot har lagt en SDK ud, og simpliciferer processen for udvikling af eksterne programmer til AR.Dronen. Vi har fået inspiration fra Parrot's AR.Drone SDK.

[1] Autonom styring vil fremover i rapporten blive referet til som autonavigation.

2 Projektbeskrivelse

Projektet opstod ved, at Torben Gregersen ønskede at have AR.Dronen til at indgå i et projekt med fokus på styring og video streaming fra AR.Dronen, samt at undersøge, hvad et DevKit8000 kan bruges til. Torben ønskede styringen og video streaming udviklet, så man i fremtidige projekter vil kunne benytte sig af AR.Dronen og dens muligheder uden at skulle udvikle styring og video streaming. DevKit8000 var en ønsket platform af Torben, for at få styringen ned på en mobil enhed med embedded processor.

Vi valgte at udbygge projektet med autonavigation for at demonstrere, hvor nemt projektet kan udbygges og fordi der var plads i tidsplanen. Ved at bruge de eksisterende klasser, der blev udviklet til styringen og video streaming af AR.Dronen, blev autonavigationen implementeret.

Vi prøvede at styre AR.Dronen på en Android mobiltelefon vha. telefonens gyroskoper og accelerometer, med FreeFlight 2.0 applikationen af Parrot. Styringen af AR.Dronen vha. telefonen virkede ikke naturlig og krævede stor koncentration og øvelse. Derfor fik vi idéen til at lave en styring af AR.Dronen med en konsol kontroller, da de normalt giver mere intuitiv styringsfornemmelse. Det begrunder vi med vores egen erfaringer fra spillekonsoller og deres kontroller.

2.1 Planlægning

Alt arbejde er foregået lokale 331, der er skolen netværks lab. I følgende afsnit vil vi beskrive tidsplanen for projektet, ansvarområder, iterationen af projektet og hvilke metoder vi har brugt til at planlægge projektet.

2.1.1 Ansvarsområder

Overordnet design af interfaces og funktionalitet er lavet i fællesskab.

Ansvarsområde	**Person**
Video	MM
GUI design	MM
Styrings kommandoer	KJ
Autonavigation	KJ
Devkit8000 implementering	MM
Xbox Kontroller	MM
Navigation data	MM
Rapport	MM, KJ
Kravspecifikation	MM, KJ

Mads Havgaard Mikkelsen(MM) og Kasper Kirkeby Jacobsen(KJ)

2.1.2 Iterationer af projektet

Eftersom projektet har mange små funktionaliteter, har vi valgt at lave flere iterationer af projektet, hvor flere funktioner er blevet tilføjet i hver iteration. Ved at dele projektet op i iterationer er der en klar plan for hvad der i den kommende tid skal laves. At dele det op i iterationer gør tidsplanen overskuelig og nemt håndgribelig. Projektet blev delt op i 4 iterationer, hvor indholdet i iterationerne kan ses nedenfor:

Iteration 1:

- Use case 2: Opdater video feed

Iteration 2:

- Use case 7: Manøvrering af AR.Drone med Xbox kontroller og klar til at modtage manøvrering data fra Interface.
- Use case 3: Skift video kamera (Knap mangler dog stadig på interface)
- Dele af Use case 5 Opdater navigation data bliver gjort færdig.

Iteration 3:

- GUI bygget færdig mangler dog settings knap
- Use case 5: Opdater navigation data færdig

- Use case 4: Styring af AR.Drone med touchskærm færdig

Iteration 4:

- Use case 1: System initialisering færdig (med en lille mangel)
- Use case 6: Skift settings færdig
- Use case 8: Auto-navigation efter linje på gulvet færdig

2.1.3 Tidsplan

	Planlagt dato	Faktisk dato
Iteration 1	3/9/12	27/8/12
Iteration 2	12/10/12	4/10/12
Iteration 3	12/11/12	12/11/12
Iteration 4	1/12/12	26/11/12
Accepttest	3/12/12	4/12/12
Afleverings dato	19/12/12	

Tidligt i projektet kunne vi se at, vi kunne nå hele projektet i god tid, og umiddelbart efter efterårsferien begyndte vi at snakke om udvidelser af projektet. Det betød at autonavigationsdelen af projektet blev tilføjet. Xbox kontrolleren tilføjelsen har været med i projektet fra starten af.

2.1.4 Projektstyring

Under projektet har der været ugentlige møder med gruppens vejleder, Torben. Gruppen har arbejdet 3-5dage om ugen på IHA Katrinebjerg. På arbejdsdagene blev der afholdt statusmøde om morgenen for at tale om fremgang, problemer og målsætning for dagen.

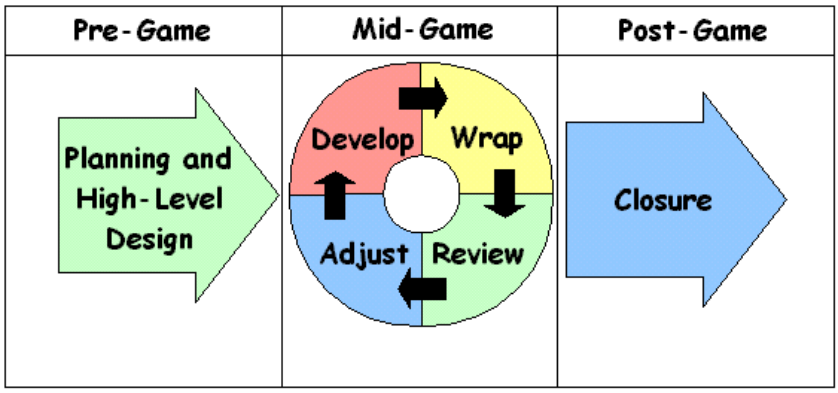
For at holde styr på alle relevante filer for projektet, har vi brugt en fælles mappe på fildelingstjenesten Dropbox, så begge medlemmer af gruppen har haft adgang til alle filerne. Det kræver dog at man kordinerer arbejdet. Word filer og lignende kan konflikte, hvis mere end en person ændrer i dem på samme tid. Qt Creator der er blevet brugt til at udvikle softwaren tillader flere personer at sidde i projektet på samme tid, da den ved udefrakommende ændringer spørger om man vil have opdateret sit projekt.

2.1.5 Metoder

Vi vil i dette afsnit komme ind på nogle af de arbejdsmetoder vi har brugt i løbet af projektet. Under design afsnittet vil vi beskrive metoderne der er benyttet til at strukturere designet.

2.1.5.1 SCRUM

SCRUM er en software udviklings metode, som bliver brugt verden over i software projekter [Ref 4]. Figur 1 viser et SCRUM forløb.

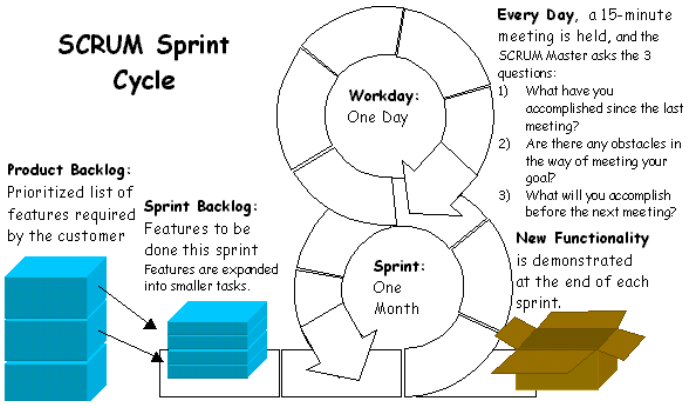


Figur 1 - SCRUM forløb [Ref 4]

Samme metode er blevet brugt i dette projekt. I dette projekt har pre-game i Figur 1 været at specificere af krav, lave en forundersøgelse og at udarbejde en strukturering. Mid-game har været udvikling af softwaren, der opfylder Use case kravene fra kravspecifikationen. Mid-Game's faser kort beskrevet:

- **Develop:** Implementer, test og dokumenter
- **Wrap:** Færdiggør, evaluer og integrer
- **Review:** Gennemgang af det implementerede software
- **Adjust:** Rettelser og ændringer i krav og planer

Mid-game forløbet har bestået af SCRUM sprint cyklusser. Hver enkelt iteration i projektet, er foregået som en SCRUM sprint cyklus. Figur 2 viser en SCRUM sprint cyklus, hvor det fremgår hvordan arbejdsdagen og målene for den enkelte dag planlægges. Perioden for vores SCRUM sprint cyklus, har været afhængig af iterationens omfang.



Figur 2 - SCRUM sprint cyklus [Ref 4]

SCRUM forløbets post-game har i dette projekt bestået i at udfærdige projekt rapporten og aflevere det samlede produkt til Torben.

Vi valgte at benytte SCRUM metoden, da det er en fleksibel metode at løse et projekt på, da det giver mulighed for ændringer undervejs i projektet. Derudover giver det en god sparring, når man dagligt gennemgår de problemer man har haft.

2.1.5.2 Blog

På vores blog, www.mklsn.net, har vi med jævne mellemrum skrevet milepæle omkring projektet. Det er gjort for at dele vores erfaringer med andre, og til dels også for o s selv. Vi har kunnet gå tilbage i bloggen og se, hvordan vi har lavet diverse implementeringer mm.. Bloggen blev mest benyttet i starten af projektet.

2.2 Udviklingsværktøjer

2.2.1 Word

Alle dokumenter er skrevet i Microsoft Office Word 2007. Programmet er valgt, da begge gruppe medlemmer er vant til at bruge dette program.

Der er lavet en template for hvordan dokumenterne skal se ud så alle dokumenter har samme udseende.

2.2.2 Visio 2010

Microsoft Visio indeholder mange værktøjer, der giver gode muligheder for hurtigt og effektivt at lave flowcharts, sekvensdiagrammer og figurer.

Microsoft Visio er blevet brugt til at lave de fleste figurer, flowcharts og sekvensdiagrammer.

2.2.3 Qt Creator

Qt Creator er en programmerings IDE, der med fordel kan bruges, hvis man benytter Qt frameworket til at opbygge sit program. Qt Creator er blevet brugt til alt programmeringen der er lavet i projektet. Qt Creator er også anvendt til at kompile programmet med da det giver et nemt interface til kompiling med forskellige kompilerer (MinGW, ARM og gcc).

2.3 Analyse og specifikation

Dette afsnit beskriver det arbejde der er lavet før vi er begyndt på designet af selve projektet. Foranalysen er vigtigt for at kunne lave et projekt, hvor målsætningerne er realistiske og kravene præcise. Foranalysen og kravspecifikationen beskrevet i dette afsnit.

2.3.1 Foranalyse

For at kunne lave en kravspecifikation med en realistisk målsætning, er der lavet en forundersøgelse af forskellige emner og dele af projektet. Det har dannet grundlag for en vurdering af hvilke muligheder og begrænsninger, der har været I projektet. Undersøgelsen har haft fokus på AR.Dronen og den hardware oplæget lagde op til der skulle udvikles til. Denne analyse er blevet udvidet under projektets forløb efter flere krav blev tilføjet.

2.3.1.1 GUI toolkit

Qt er valgt som programmerings GUI toolkit til udvikling af softwaren til projektet. Qt er et veldokumenteret framework bygget op i C++ [Ref 7], der understøtter mange platforme med native GUI design. Qt GUI toolkit kan bruges til mange forskellige script- og programmeringssprog. Vi har at benytte det til C++, da C++ er kompatibelt med langt de fleste moderne platforme, deribland ARM arkitekturiske processorer som er den slags processor DevKit8000 benytter sig af.

Da Qt ikke kun er et GUI toolkit, men også et framework kommer det med mange cross-platform funktioner og egenskaber, der gør det nemt at udvikle i til forskellige platforme. Qt kan køre uafhængigt af X11. X11 er et vindue manager server, som afvikles direkte på Linux's frame buffer vha. Qt's egen vindue server [Ref 8].

2.3.1.2 WiFi

Vi har valgt at implementere WiFi som USB input, da der allerede var tilgængelige WiFi USB enheder på skolen. Vi valgte en WiFi model fra D-Link, DWL-G122, da det indeholder et chipset RT73, der er kompatibelt med Linux [Ref 9].

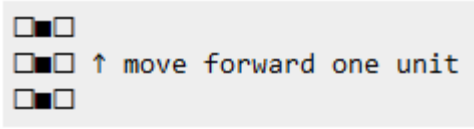
2.3.1.3 Kontroller

Vi har valgt en Xbox 360 kontroller som styrings kontroller til projektet. Dette valg faldt naturligt, da vi havde udstyret, samt god erfaring med at bruge sådan en kontroller til computer vha. en Xbox 360 Wireless Receiver.

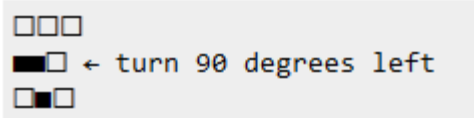
2.3.1.4 Autonavigation

For at kunne flyve efter en linje skulle det først undersøges, hvilke muligheder AR.Dronen havde for at se linjen. Derefter skulle det undersøges hvordan informationen kunne omsættes til kontrol af AR.Dronen. Det blev hurtigt konstateret, at AR.Dronen kun kan følge en linje vha. kameraet der peger nedad, da AR.Dronen ikke har andre sensorer der kan se eller registrere end linje under den. Der skulle findes en metode til at bruge kameraet til at give navigations kommandoer. Ved søgning på internettet blev der ikke fundet andre systemer, der styrer et flyvende objekt efter en linje under det. Der var ikke nogle færdige løsninger, til at følge en linje, på internettet der kunne implementeres.

På et forum på nettet fandt vi en, der ville få en robot til at følge en tape linje på et gulv vha. et kamera. Idéen fra forummet var at få delt det kamera billede som man har til rådighed op i et 3x3 grid, hvor man bestemmer om hvert felt i grid'et skal være enten sort eller hvidt, se Figur 3 for eksemplet fra forum'et. Feltet skal være sort, hvis tape linjen går igennem feltet, og hvidt hvis det ikke gør. Når dette grid er bestemt, kan man ud fra en algoritme få robotten til at tage valg som f.eks. kør frem, drej til venstre, højre osv.. Målet er hele tiden at have et sort felt i midten af grid'et, så robotten hele tiden er centreret oven den linje den følger. Svaret til hans spørgsmål blev brugt som udgangspunkt til metoden brugt til at følge linjen[Ref 2].



What a left turn looks like:



Figur 3 - Eksempel fra forum [Ref 2]

2.3.1.5 AR.Drone

Undersøgelser om AR.Dronen viste, at der er 4 vigtige områder, man skal vide noget om, før man laver et program til den. Kommunikation, video, manøvrering og tilstande er de 4 vigtige områder af AR.Dronen man skal undersøge. Kommunikationen foregår med en AD-HOC forbindelse, hvor en klient opretter forbindelse til AR.Dronens WiFi netværk. Der bruges 3 UDP porte til kommunikation.

Port	Beskrivelse
5554	Porten hvorpå AR.Dronen sender navigationsdata på. I navigationsdata modtages alt information omkring AR.Dronen's tilstande og flyveinformationer. Denne port skal modtages fra.
5555	Video streaming sker på port 5555. Denne port skal modtages fra.
5556	AT kommandoer til AR.Dronen sendes på denne port. AT kommandoer indeholder alt fra konfiguration til styringskommandoer. Denne port sendes til.

AR.Dronen har 2 kameraer, som den kan bruge til at sende video data til klienten. Den har et kamera på fronten, der peger fremad og et kamera i bunden, der peger ned af. Videoen kan sendes i to slags videokodeks et UVLC (VLIB) og P264 (H264). AR.Dronen sender som standard med UVLC, skal P264 benyttes skal dette aktiveres. UVLC sender ikke komprimerede billeder, mens P264 kun sender ændringer i billedet. P264 kræver mindre båndbrede men mere kompliceret decoding. Vi valgte kun at benytte os a UVLC, da det er det simpleste videokodeks af de to, da P264 er komprimeret og derfor skal decodes på en mere kompliceret måde.

AR.Dronen kan manøvrere i et x-y-z plan, rotere om sig selv og har derudover kommandoer til at lette og lande. AR.Dronen har flere forskellige tilstande at sende navigationsdata ud. Af disse tilstande findes

bootstrap der slet ikke sender nogen navigationsdata, der findes navigationsdata demo tilstand der kun sender der vigtigste information, og der findes en debugging tilstand der sender information omkring alt hardware og software på AR.Dronen.

AR.Dronen har flere forskellige flyvetilstande, der beskriver om den er flyvende, landet, hovering eller er i emergency tilstand. Disse tilstande kan læses i navigationsdataen.

Parrot har lagt en SDK ud på deres hjemmeside, som frit kan bruges til projekter med AR.Dronen. Denne SDK er blevet brugt som reference og hjælpemiddel under udviklingen af programmet.

2.3.2 Kravspecifikation

Kravspecifikationen i projektet er bygget op på grundlag af de krav, der blev stillet i projektoplægget, samt vores egne krav. Disse krav er vist i indledningen. På grundlag af kravene og vores idé om, hvordan applikationen skulle virke, og det grafiske interface skulle se ud, blev der opstillet en use cases til at matche de ønskede egenskaber.

I forundersøgelsen blev der undersøgt hvilke krav der var mulige at løse, samt hvilke ekstra krav vi kunne stille til projektet. Ud fra foranalysen konkluderede vi at alle kravene der var blevet sat, var mulige at løse.

Der er løbende blevet tilføjet ekstra krav, eftersom vi har haft tid til at udvide projektets omfang. Tabel 1 viser use casene for projektet.

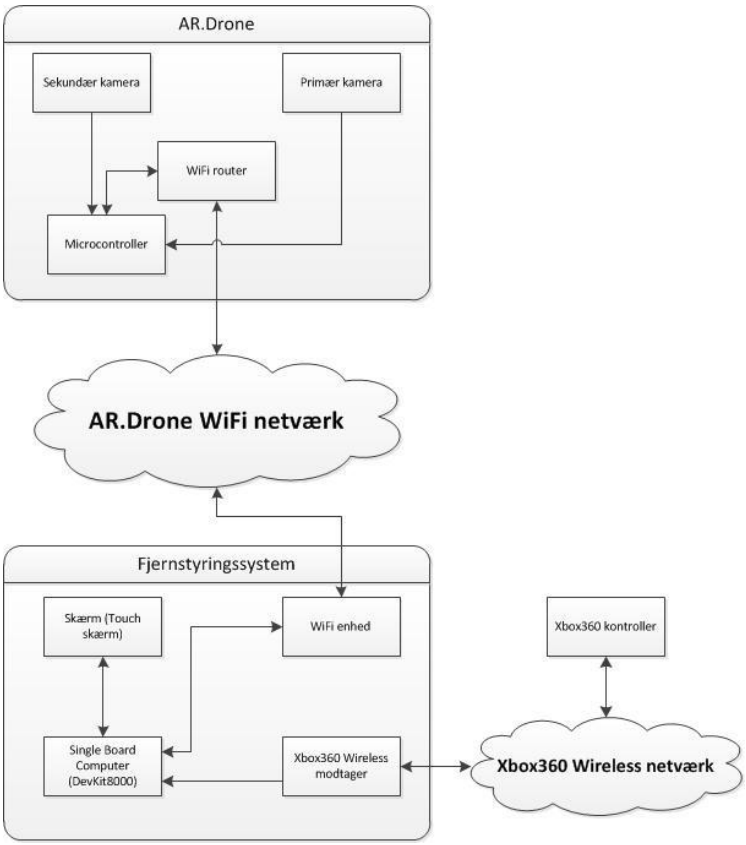
Tabel 1 - Use cases i projektet

Use case navn
System initialisering
Opdater video feed
Skift videokamera
Manøvrering af AR.Drone med touchskærm
Opdater navigationsdata
Skift settings
Manøvrering afAR.Drone med Xbox kontroller
Autonavigation efter linje på gulvet

Beskrivelse af use casene i Tabel 1 kan ses i Kravspecifikationsrapporten.

2.4 Design

På grundlag af de stillede krav endte vi med et system, som vist i Figur 4. Figur 4 viser de overordnede hardware blokke.



Figur 4 - Oversigt over systemet

Dette afsnit er med i rapporten for at argumentere for de metoder vi har brugt til at designe projektet, og for at beskrive de enkelte metoder. Ved gennemlæsning af dette afsnit vil design processen i projektet blive beskrevet.

Alt softwaren programmeres vha. Qt frameworket i C++.

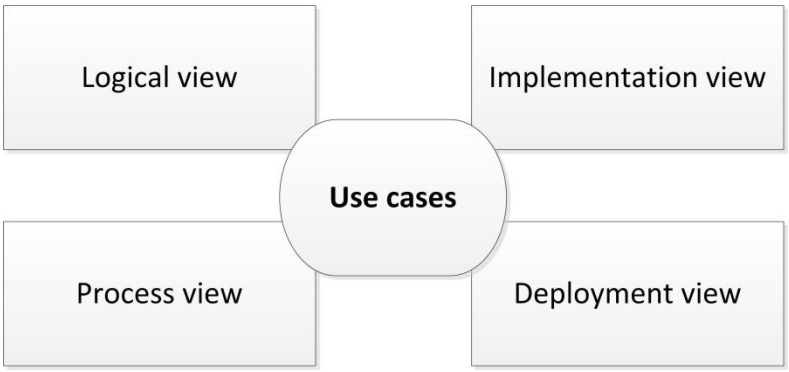
2.4.1 4+1 view strukturering

For at lave en god strukturering af projektet blev der brugt 4+1 view modellen. 4+1 view modellen strukturerer op omkring use cases (+1 viewet) og har derudover 4 views der er forskellige stadier af struktureringen. Illustrering af dette er vist på Figur 5.

- **Use cases:** Funktionel beskrivelse af et krav stillet i kravspecifikationen.
- **Logical view:** Sekvensdiagrammer af use cases, statisk klassediagram, klasse og funktions beskrivelse
- **Process view:** Sammenhæng mellem de forskellige klasser og funktioner, beskrivelse af delte resourcer klasserne imellem.

- **Deployment view:** Protokoller og kommunikation imellem forskellige enheder i projektet.
- **Implementation view:** Beskrivelse af hvordan softwaren sættes op, kompileres, installation, software eksekvering mm..

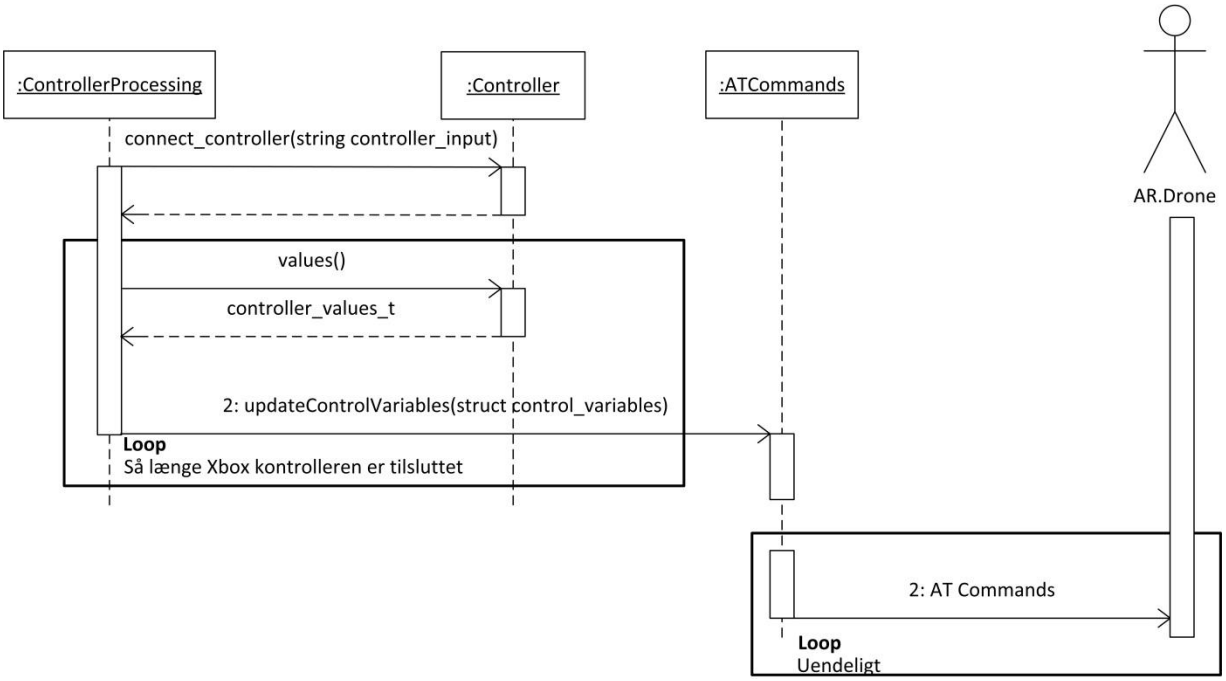
De forskellige views er lavet i samme rækkefælge som ovenstående liste.



Figur 5 - 4+1 view model

Figur 5 viser oversigten over 4+1 view, og hvordan de bygges op omkring use cases.

Ud fra de 8 use cases, som projektet bygger op omkring, er logical view blevet udarbejdet. Sekvensdiagrammer for de forskellige klasser er blevet lavet som en del af logical view, så det er klart hvordan de forskellige klasser skal opføre sig internt. Sekvensdiagrammerne beskriver også hvilke funktioner der kaldes og af hvilke klasser de kaldes og også i hvilken rækkefølge det gøres.

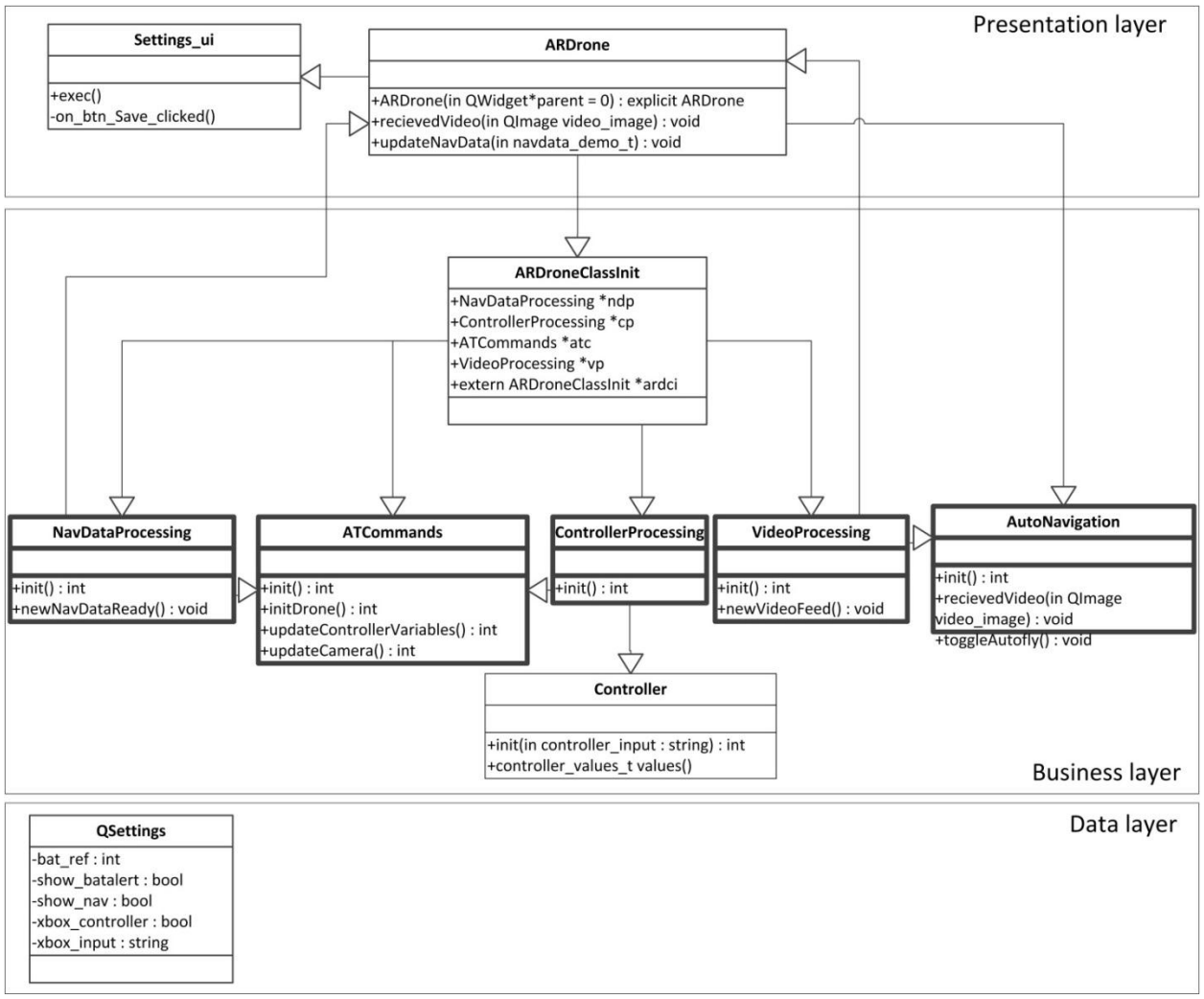


Figur 6 - Sekvensdiagram for Xbox kontroller styring og processering¨

Figur 6 viser sekvensdiagrammet for styring med Xbox kontrolleren. Fra Figur 6 kan man determinere hvilke klasser der snakker sammen og hvilke funktioner de bruger til det. De tykke bokse på Figur 6 hvor der står Loop i, er en løkke, hvori betingelsen for løkken er angivet. Objekterne i disse løkker, er de objekter der er selvkørende. Klasser der har disse selvkørende objekter, er også markeret med fed i Figur 7, der viser det statiske klassediagram over programmet. Det statiske klassediagram, som vist på Figur 7, består af klasser, variabler m.m. der kan determineres ved kun at observere vores sekvensdiagrammer. Det viser at man ud fra sekvensdiagrammer kan skabe sig et overblik over projektet og samle det i et statisk klasse diagram. Pilene i Figur 7 viser hvorfra de forskellige klasser kaldes, og er også determineret ud fra sekvensdiagrammerne.

I implementeringen vil der komme flere funktioner og variabler, men de er ikke essentielle for programmets struktur, og er derfor ikke med.

Alle sekvensdiagrammerne i projektet er bygget op efter samme skabelon som vist på Figur 6. Resten af projektets sekvensdiagrammer kan ses i Dokumentations rapportens design afsnit.



Figur 7 - Statisk klassediagram

2.4.2 Video

Data, der modtages fra AR.Dronen's video port (5555), skal behandles så det ender ud i at blive et billede. Processen og klassen der processerer dataene fra data til billede, kaldes for et kodeks. AR.Dronen kan sende i to slags formater, VLIB og P264, hvor begge formater kræver hver sit kodeks. AR.Dronen's Developer Guide [Ref 1] beskriver begge kodeks og hvordan de kan dekodekseres fra data til billede. Parrots AR.Drone SDK kommer med et kodeks programmeret i C til begge formater.

Som kodeks benyttede vi os af et projekt fra Nokia der var skrevet i netop Qt frameworket [Ref 10]. Det gjorde det nemt at implementere i vores projekt, da source filerne allerede var modificeret til Qt's variabler.

I Nokia's projekt er alle de forskellige structs og variabler, der i AR.Dronens SDK kodeks for VLIB er samlet i headerfilen vlib.h. Derudover er alt koden fra AR.Dronen's SDK VLIB kodeks, gemt i video.h.

Dette har gjort det væsentligt nemmere for os at tage fat i, da filerne tilknyttet kodekset i AR.Dronens SDK ligger spredt i 10-15 filer, og de i Nokias projekt er samlet til 2 filer.

2.4.3 Navigationdata

Navigationsdata modtages fra AR.Dronen på port 5554. AR.Dronen sender ud til dens klient med alle data fra den, som f.eks. højdemåler, hastighed, tilstand og en mange anden data. Navigationsdataene sendes som pakker, og hver pakke har struktur som vist i Figur 8.

Header 0x55667788	Drone state	Sequence number	Vision flag	Option 1			...	Checksum block		
				id	size	data	...	cks id	size	cks data
32-bit int.	32-bit int.	32-bit int.	32-bit int.	16-bit int.	16-bit int.	...	...	16-bit int.	16-bit int.	32-bit int.

Figur 8 - Navigations data pakke struktur

Der fremgår af Figur 8, at der kan forekomme mange mængder data i samme pakke (”Option” strukturen). De mange forskellige data informationer AR.Dronen sender, er i ”Option” strukturen, der gentager sig for hver ”Option” den sender. Hver ”Option” har forskellige data, der hver især skal pakkes ud i forskellige structs. ”id” i ”Option” bruges til at genkende hvilken struct dataene skal pakkes ud i. ”size” beskriver mængden af data der bliver modtaget i byte for dette ”id”.

F.eks er der en ”Option” der hedder NAVDATA_DEMO, og er den data der indeholder de vigtigste informationer om AR.Dronen, som f.eks kontrol tilstanden, batteri niveau tilbage, højde måler og meget andet. NAVDATA_DEMO har id’et 0, og når vi modtager en ”Option” med id’et 0, ved vi hvilken struct den tilhørende data skal gemmes i.

Alle de ”Options” der modtages fra AR.Dronen gemmes i diverse structs, der er tilknyttet id’et. Vi sætter AR.Dronen i navigations data demo mode, der indeholder disse ”Options”, for kun at modtage de vigtigste information om AR.Dronen:

- **NAVDATA_CKS_TAG:** Checksum for den sendte data, ”Checksum block” på Figur 8, bruges til efterfølgende at tjekke, om dataene blev modtaget korrekt uden fejl.
- **NAVDATA_DEMO_TAG:** Indeholder flyve informationer og diverse generelle informationer om AR.Dronen. Det er den vigtigste struktur, da den indeholder alle basis informationer.

Alle de forskellige ”Option” strukturer kan ses i navdata.h, og er kopieret fra AR.Dronens SDK hvor variablernes typer derefter er ændret til Qt format. Variablerne ændres til Qt variabler, for at det forbliver krydsplatforms kompatibelt, da Qt holder styr på hvilket styresystem variablerne skal tilpasses

Checksum bruges til at beregne om der er gået data tabt i overførslen. For at beregne checksummen af den modtagede data bruges en simpel algoritme, der er opgivet i AR.Dronens SDK.

2.4.4 Xbox kontroller

Data behandlingen af Xbox kontrolleren designes, så der er en tråd, der hele tiden venter på nye inputs fra Xbox kontrolleren. Alle inputs fra Xbox kontrolleren gemmes i en struktur med de digitale inputs (knapper) og analoge inputs.

Når Xbox kontroller inputs behandles tjekkes det hvilke knapper der er aktive. De analoge værdier bliver sendt videre til AR.Dronen for at kontrollere den.

2.4.5 AT Commands

Klassen atcommands er blevet lavet for at kunne sende AR.Dronen at-kommandoer fra programmet. Denne klasse har til formål at håndtere kommunikation til AR.Dronen. Klassen bruges til at sætte AR.Dronen i den rigtige tilstand, når den startes op, da den altid startes i bootstrap, så den sender den ønskede navigationsdata til klienten. Derudover er det denne klasse der sender styrings kommandoerne til AR.Dronen. Styringen har en struct, referet til som styringstruct, der indeholder alt data der skal sendes. Styringsstructen indeholder styringsdata, der beskriver AR.Dronens bevægelser på x, y, z akserne, samt rotation, emergency, landing og takeoff. Styringsstructen kan opdateres ved at andre klasser kalder en funktion i ATCommands klassen, hvori data skrives til styringsstructen. ATCommands klasse styrer også hvilket kamera video skal sendes fra.

AR.Dronen skal ifølge ARDrone SDK(side 29) [Ref 1] modtage nye At Commands hvert 30 ms for at have stabil flyvning. Derfor sender programmet At Commands hvert 20 ms for at være på den sikre side.

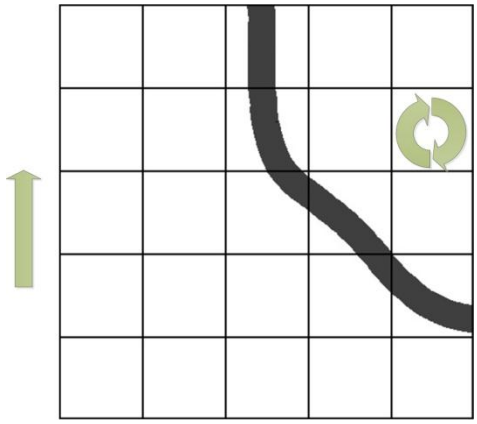
2.4.6 Autonavigation

Autonavigationen skal gøre to ting, for at få AR.Dronen til at flyve efter en sort linje på gulvet. Autonavigationen skal først omdanne billedet fra AR.Dronen til et grid der kan bruges af en algoritme, og derefter med denne algoritme skal den sende at-kommandoer der får den til at flyve efter linjen.

Billedet fra AR.Dronen omdannes til et 5x5 grid, hvor hvert enkelt felt bestemmes til at være enten sort eller hvidt, alt efter hvad kameraet ser i dette felt. Programmet kigger på alle pixels i hvert enkelt felt i grid'et, og hvis et felt indeholder nok mørke pixels, sættes dette grid felt til 1 ellers 0.

Når gridet er fundet, bruges der en algoritmen til at flyve efter en linje på gulvet. Denne algoritme prøver hele tiden at holde AR.Dronen centreret over linjen ved at dreje og flyve den sidelæns. Målet er at have linjen til at gå gennem grid felt (1,3) og (5,3). Den flyver fremad, når et af de 5 felter i toppen af Figur 9, er sort.

Eksempel på linje og kommandoer kan ses i Figur 9.



Figur 9 - Eksempel på linje og kommandoer

På Figur 9 kan et eksempel på en linje ses. Pilene indikerer hvilke kommandoer AR.Dronen skal have for at følge linjen ifølge algoritmen. Hvis man billede behandlede Figur 9 vil det give et grid output som følger:

0	0	1	0	0
0	0	1	0	0
0	0	1	1	0
0	0	0	1	1
0	0	0	0	0

Dette vil give AR.Dronen kommandoerne flyv frem, da et af felterne i toppen af grid'et indeholder linjen. Derudover vil den få en kommando om at dreje til højre om sig selv for at prøve at få linjen til igen at gå så lige så muligt igennem billedet, dvs. linjen går igennem felterne (1,3) og (5,3).

Flere eksempler på linje i grid'et kan ses i Dokumentationsrapporten under afsnittet Implementation view.

2.5 Implementering

Implementeringsafsnittet er med i rapporten for at beskrive implementeringen og opsætningen af softwaren og enheder til DevKit8000, der kræves implementeret, for at accepttesten i sidste ende kan opfyldes.

Softwaren der er udviklet er beskrevet tidligere i afsnittet 2.4 Design, og der vil ikke blive gået i dybden med udviklingen af dette, da design delen er mere sigende end sourcekode beskrivelse. For at få dybere indsigt i implementering af denne kode, medfølger source koden i [BILAG 3].

Der er en detaljeret beskrivelse af implementeringen kan findes i Dokumentationsrapporten under afsnittet Implementation view.

2.5.1 WiFi

En WiFi enhed kan ikke bare tilsluttes og forventes at virke på et Linux styresystem, da det ikke er garanteret, at WiFi enheden har en kompatibel kernel driver installeret. I dette projekt benyttede vi os af en D-Link DWL-G122 USB WiFi, der har et RT73 chipset. For at få installeret en RT73 chipset driver i Linux kernelen, skal der kompileres en udgave af Linux kernelen, hvor RT73 driveren er aktiveret. Når Linux kernelen er kompileret, skal kernel modulerne kompileres, som indeholder enhedsdriverne. Disse kernel moduler skal flyttes over på styresystemets filsystem.

Når WiFi driveren er installeret, er der ikke noget grafisk interface til at oprette forbindelse til et WiFi netværk, som vi er vant til med Windows og andre grafiske vindue baserede styresystemer, da den Linux der er installeret på DevKit8000, er konsol baseret. Derfor skal der manuelt søges efter WiFi netværket til AR.Dronen, og WiFi interfacet skal derefter sættes op til at oprette forbindelse til AR.Dronens Ad-Hoc netværk.

2.5.2 Qt

Qt frameworket skal installeres på DevKit8000 for at den udviklede software, der er baseret på Qt, kan afvikles derpå. Når Qt skal afvikle programmer på embeddede processorer, hvilket DevKit8000 har, der er kompileret fra en anden computer kræver det at Qt's libraries har eksakt samme sti som på den computer programmet blev kompileret fra. Derudover skal Qt frameworket kompileres til den specifikke embeddede processor, der i DevKit8000's tilfælde er en processor med ARM arkitektur, for at den kan afvikle softwaren.

2.5.3 Xbox kontroller

Xbox kontrolleren kræver ligesom WiFi enheden, at Linux kernelen har installeret en kernel driver der kan håndtere Xbox kontrolleren. Linux kernelen kommer heldigvis med driver for joysticks, deriblandt også Microsoft joysticks. Disse kernel drivere kan også håndtere Xbox kontrollerer.

Ligesom med WiFi enheden, skal Linux kernelen og kernel modulerne re-kompileres.

2.5.4 Software afvikling på DevKit8000

Softwaren der er blevet udviklet i projektet, kan ikke afvikles på eksakt samme måde på DevKit8000, som på en computer med alm. styresystem. Grunden til det skyldes, at der ikke er nogen vindue

manager til at tage sig af grafikken, og en evt. GUI da Linux udgaven på DevKit8000 er konsol baseret. Derfor skal man ved afvikling af softwaren fortælle Qt frameworket igennem programmets start parameter, at den skal afvikle softwaren på Linux's framebuffer, der sørger for at "tegne" det grafiske interface. Parameteren hedder –qws (Qt Windowing Server), og får programmet til at være server application, dvs. den selv opretter et QtApplication objekt og tegner sig selv i framebufferen. På normale desktop computerer er der installeret en en vindue server, der tager sig af alt det, og derfor er –qws parameteren ikke nødvendig på disse computerer.

2.5.5 Software på Microsoft Windows platforme

Softwaren der er udviklet i dette projekt er også kompatibel og implementeret til Microsoft Windows styresystemer. Det er gjort enkelt, da Qt frameworket gør krydskompilering til en nem ting, da den ved brug af Qt's egne klasser tager højde for krydskompilering mellem diverse styresystemer og platforme. Der skal dog bruges andre driver og anden kode ved brug af Microsoft Windows platform til Xbox kontrolleren

Softwaren blev gjort Microsoft Windows kompatibel, da Kasper b.la. har brugt Windows som udviklingsmiljø. Mange andre kan få gavn af softwaren, da Windows er det mest udbredte styresystem.

2.6 Test

Test af et projekt er vigtigt for at kunne dokumentere, at de dele der er lavet virker enkeltvis, og at de også virker som en helhed. Man tester for at fejl i programmet lettere kan lokaliseres, hvis kun en enkelt af testene er lavet vil det være sværere at finde ud af hvor fejlen helt præcist er. Hvis fejlen findes i enhedstesten ved man, at fejlen ligger i den enhed man tester. Findes fejlene derimod i integrationstesten, og de enkelte enheder blev testet til at virker, ved man at det er i integrationen af enheden at fejlen ligger.

Under projektet er der lavet enhedstest af dele af projektet, for at test funktionaliteten af de forskellige dele af projektet. Delene fra enhedstestene blev integreret i det samlede projekt undervejs og testet sammen i en integrationstest. Som afslutning på projektet er der blevet lavet en accepttest for at se om kravene og use cases i kravspecifikationen er blevet opfyldt.

2.6.1.1 Enhedstest

Enhedstestene er foretaget for at teste de enkelte klassers funktionalitet, før det er blevet integreret i systemet. Disse tests er bl.a. foretaget vha. at Qt Creators debugger værktøj, der gør det nemt at indsætte breakpoints og watches til at overvåge de forskellige variabler, der ønskes overvåget. Nogle dele kunne ikke testes udelukkende vha. debuggeren, da der mangler udefrakommende inputs. Derfor blev der brugt små test programmer [Bilag 2 og Bilag 4], der simulerede de inputs enheden skulle bruge fra AR.Dronen.

Fejl er nemme at finde i de enkelte enheder i disse test, da kun en enkelt enhed bliver testet ad gangen og området fejlen kan ligge i, er derfor begrænset.

2.6.1.2 Integrationstest

Integrationstestene er blevet lavet undervejs når enhederne er blevet færdigudviklet, testet og integreret i det samlede system. Der er lavet en integrationstest hver gang en enhed er blevet integreret, for at undgå for store integrationer, der vil gøre fejlfinding sværere end højst nødvendigt.

2.6.1.3 Accepttest

Accepttest er den endelige test, der verificerer at kravene fra Use cases er blevet opfyldt.

Test og testresultater kan ses i afsnittet Test i Dokumentationsrapporten.

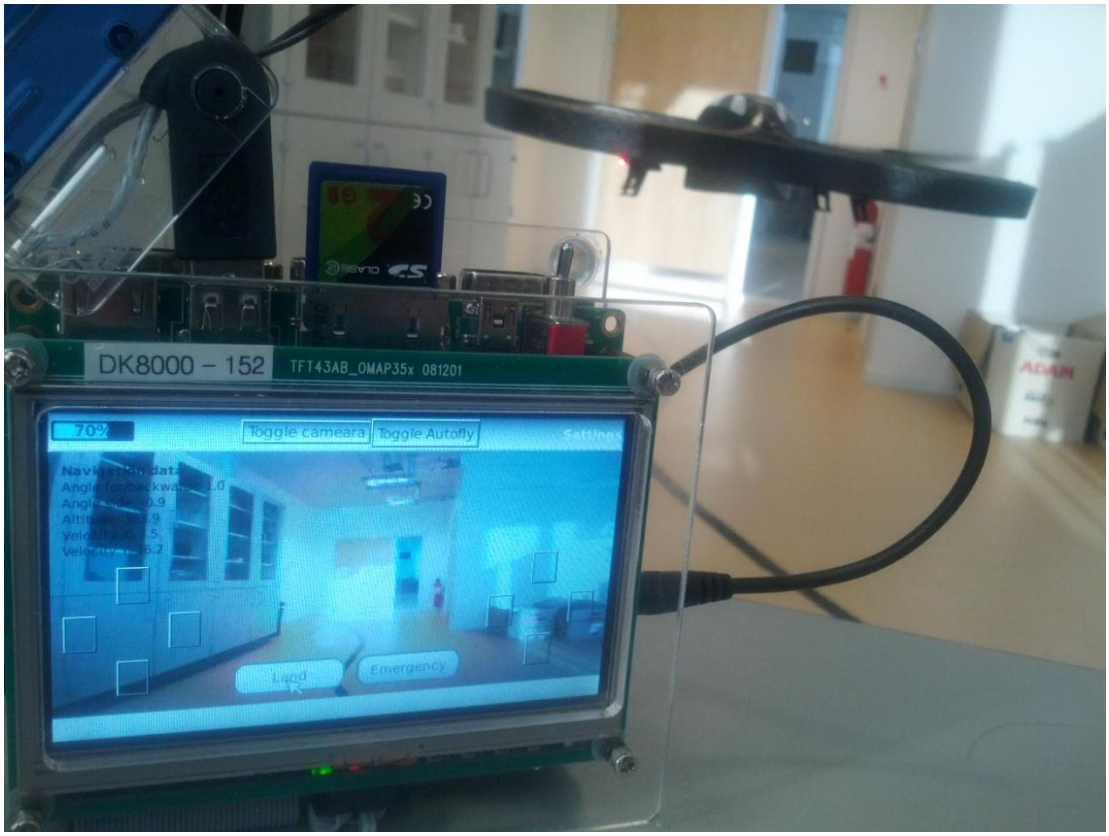
2.7 Resultater

AR.Dronen kan styres vha. programmet fra DevKit8000, og videofeedet bliver vist på skærmen. Programmet kører meget langsomt på DevKit8000, og der er nogle sekunders forsinkelse fra man trykker på en knap til AR.Dronen reagerer. Det direkte video feed fra AR.Dronen er også et par sekunder bagud og viser ikke med de 15fps, som AR.Dronen transmitterer med. Hvis programmet køres fra en Linux x86 computer, er der ikke nogen mærkbart forsinkelse på styringen og videoen.

AR.Dronen kan styres med en Xbox kontroller når programmet køres fra computeren. Kontrollen er meget mere flydende og mere intuitiv end kontrollen fra skærmknapperne. AR.Dronen kan også styres med Xbox kontrolleren fra DevKit8000, men her er der en forsinkelsen på 2-3sekunder. Kontrollen fra skærmen har den ulempe at der kun kan udføres en handling ad gangen, dvs. flyv frem, op, ned eller lignende, da DevKit8000 ikke har multitouch, og man på computeren ikke har mulighed for at trykke på 2 knapper ad gangen. Kontrollen af AR.Dronen med Xbox kontroller føles lettere end kontrollen fra en Android mobiltelefon, da den er meget mere intuitiv, og vi er vant til den samme slags styring fra konsolspil.

Programmet kan følge en linje på gulvet med autonavigationsprogrammet. Linjen må ikke have for skarpe sving, og linjen skal være sort på et lyst gulv og ca. 4cm bred. Flyvning foregik i ca. 1.7-1.9m højde over gulvet. Flyvningen er lidt ustabil, men AR.Dronen følger tydeligt linjen. Det var ikke muligt at få AR.Dronen til at følge linjen fra DevKit8000, pga. den langsomme reaktionstid og forsinkelserne som opstod.

Det lykkedes at modtage og pakke navigationsdataene korrekt ud, så batteriniveau, højdemeter m.m. kunne vises på skærmens grafiske interface. I indstillingsmenuen er der mulighed for at vælge, hvornår der skal vises alarm for lavt batteri niveau, om navigations dataene skal vises på skæren og ændre Xbox kontrollerens input adresse samt om styring med Xbox ko ntrolleren skal være aktiv.



Figur 10 - Devkit styring af AR.Drone

Figur 10 viser et billede af AR.Dronen styret fra DevKit8000, på billedet kan interfacet på DevKittet ses samt hvordan billedet vises.

Selvom det ikke var et krav, blev programmet implementeret på en Windows baseret x86 computer. Det kører ligeså godt til Windows som til Linux x86 computere.

3 Diskussion

Flyvning med AR.Dronen med det udviklede program virker godt. Hvis man skal styre AR.Dronen fra programmet udelukkende ved at kikke på skærmen og med knapperne på skærmen, er det ikke optimalt at flyve indenfor, da man ikke kan se til siderne, og derfor ofte vil ramme væge og andre forhindringer. Skal man flyve ved kun at observere AR.Dronen's direkte video på skærmen, kan dette godt lade sig gøre udendørs, da der er mere plads og knap så mange forhindringer.

3.1.1 Forbedringer

Softwarens kompatibilitet med DevKit8000 og andre embeddede processorer kunne godt forbedres. Dette skyldes ikke at koden til programmet ikke er optimeret, men nok nærmere platformen og Qt der ikke er en optimal løsning når skærmbilledet skal opdateres med mange fps. Vi er ikke klar over hvori problemet ligger, men vi har dog en mistanke om at det er opdaterings frekvensen af Qt's vindue server på framebufferen, der ikke kan opdatere med 15fps. 15fps er den frekvens vi modtager direkte video streams fra AR.Dronen. Dette tror vi fordi programmet kører uden problemer, hvis vi disabler direkte opdateringer af billeder fra AR.Dronen. Afvikles softwaren på en computer hvor Qt vindue server ikke benyttes, men der i stedet benyttes X11 vindue server, kører programmet også uden problemer. Vi har desværre ikke været i stand til at finde dokumentation for hvor mange fps Qt's vindue server kører med, så vores begrundelse bygger på gisninger.

Autonavigationen kan også forbedres. Der er flere ting der kan gøres for at få et bedre resultat. Først og fremmest kan man bruge en del tid på at justere de hastigheder, den laver de forskellige manøvre med. Det vil nok give et bedre resultat i sig selv, da der ikke er blevet brugt specielt meget tid på at justere algoritmen. Derudover kan grid inddelingen laves finere så det måske er 7x7 eller 9x9 grid i stedet for de 5x5 som det er nu. Hvis det laves på denne måde vil en algoritme kunne lave finere justeringer af AR.Dronen, og dermed få en mere stabil flyvning, da der er flere inputs og lave justeringer ud fra.

Mistes forbindelsen til AR.Dronen imens programmet kører, eller åbner man programmet uden at have forbindelse til AR.Dronen er der ikke muligt at få forbindelse til AR.Dronen genetableret, uden at lave en ny initialisering af programmet. Denne initialisering af programmet kan kun foretages ved at åbne og lukke programmet. Det ville være en markant forbedring hvis programmet står og pinger efter AR.Dronen, når der ingen forbindelse er til den, og automatisk genskaber forbindelse hvis der bliver genetableret forbindelse.

3.1.1.1 Udvidelser

Projektet vil i fremtiden kunne udvides på mange måder. Specielt da dette projekt har en styring der er nem at bruge, hvis man vil udvide programmet med flere funktioner.

En fremtidig udvidelse kunne være at få AR.Dronen til at kunne flyve efter GPS koordinater. For at kunne lave denne løsning vil man skulle sætte et lille stykke elektronik med et GPS modul, GSM modtager og WiFi modul på AR.Dronen. Denne elektronik vil skulle programmeres til at modtage kommandoer via GSM modulet og sende dem videre til AR.Dronen med dens WiFi. Elektronikken vil skulle være lille nok til at kunne side spændt fast oven på batteriet på AR.Dronen. Denne udvidelse ville øge rækkevidden på AR.Dronen meget, da dækningen for GSM og 3G er god i Danmark.

En anden smart feature vil være at have mulighed for at optage videoen AR.Dronen sender til programmet. Det vil kræve at gemme det stream af stilbilleder AR.Dronen sender ud i et video format.

Da programmet allerede er skrevet i Qt, der sørger for et nemt miljø til at krydskompilere til andre platforme, ville en eventuel udvidelse være at kompilere programmet til OSx (Mac). Udfordringen i at udvide det til OSx, vil være at få Xbox kontrolleren implementeret i programmet, da klasserne der snakker med Xbox kontrolleren i programmet er platforms specifikke.

4 Konklusion

Systemet kom til at køre på DevKit8000, men den ydelse det er muligt at få ud af programmet er ikke tilfredsstillende, eftersom programmet kører med en meget stort forsinkelse på alle kommandoer, der sendes til AR.Dronen, og der er stor forsinkelse på den modtagede video. Hvis programmet reelt skal bruges til noget, skal det køres fra en bærbar computer eller et board med et andet styresystem, der bedre kan afvikle programmet. En anden fordel ved at køre programmet fra en bærbar computer er, at det bliver mere mobilt, da computeren har et batteri, noget DevKit8000 ikke har.

Kontrollen med Xbox kontroller fungerer bedre end forventet. Kontrollen er meget flydende og intuitiv, i hvert fald hvis man har prøvet at bruge en konsol kontroller til at styre computerspil hvor, en bil eller lignende styres.

Der er lavet et ”proof of concept” med autonavigationen. Denne styring fungerer i praksis, men er ikke specielt funktionel. Konceptet fungerer, men der er stadig plads til forbedringer. Med autonavigationen er det vist at de udviklede klasser til programmet nemt kan benyttes i andre sammenhæng, som ved autonavigation, hvis man vil lave nye funktioner. Derudover viser det også, at billedet fra AR.Dronen er tilgængeligt for andre klasser i programmet.

Programmet egner sig bedre til udendørs flyvning, end indendørs flyvning. Det blev specielt klart, da vi tog AR.Dronen og en bærbar computer med udenfor, hvor vi fløj lidt rundt med både Xbox kontrolleren og med styring fra skærm. Styringen med Xbox kontrolleren er ikke meget anderledes ved udendørsflyvning kontra indendørsflyvning. Styring fra skærmens knapper er meget nemmere ved udendørsflyvning, da der er mere plads at flyve rundt, end ved indendørsflyvning. Udenfor skal man ikke være bekymret for at flyve ind i vægge og andre forhindringer. Når man bruger skærmbilledet til at styre fra, hvor man ikke kan se hvad der er ved siden af AR.Dronen, er det væsentlig nemmere at flyve udenddørs.

Projektet giver et godt udgangspunkt for andre projekter, der skal anvende en AR.Drone i deres projekt. Disse projekter vil med fordel kunne bruge dette projekts klasser, for ikke at skulle opfinde styringen m.m. til AR.Dronen igen .

5 Referencer

Hjemmesider checket 10-12-12

[Ref 1]
ARDrone_SDK_1_7_Developer_Guide
Kan findes som Bilag 1

[Ref 2]
Stackoverflow forum spørgsmål på internettet:
http://stackoverflow.com/questions/8463489/how-to-make-a-robot-follow-a-line-using-its-video-camera

[Ref 3]
Android market AR FreeFlight2.0 Android app:
https://play.google.com/store/apps/details?id=com.parrot.freeflight&feature=search_result#?t=W251bGwsMSwxLDEsImNvbS5wYXJyb3QuZnJlZWZsaWdodCJd

[Ref 4]
SCRUM beskrivels:
http://www.codeproject.com/Articles/4798/What-is-SCRUM

[Ref 5]
Parrots hjemmeside:
http://ardrone2.parrot.com/

[Ref 6]
Embest producent af DevKit8000:
http://www.embedinfo.com/english/product/devkit8000.asp

[Ref 7]
Qt dokumentation
http://doc.qt.digia.com/

[Ref 8]
Qt documentation af deres virtual framebuffer
http://doc.qt.digia.com/qt/qvfb.html

[Ref 9]
Liste over kompatible wireless chipsets til Linux
http://linuxwireless.org/en/users/Devices

[Ref 10]
Nokia's projekt med AR.Dronen
http://projects.developer.nokia.com/ardrone