Strukturering og design

AR.Drone control via DevKit8000

Titel:	Strukturering og design
Forfattere:	Mads Havgaard Mikkelsen
	Kasper Kirkeby Jacobsen
Vejleder:	Torben Gregersen
Projektnummer:	12042
Institution:	Aarhus Universitet Ingeniørhøjskolen
Sider:	49
Dato:	19-12-2012

Versionshistorie

Ver.	Dato	Initialer	Beskrivelse
1.0	12/10 2012	09636	
1.1	19/11 2012	09588	Tilføjelser af Autonavigation

Godkendelsesformular

Forfatter(e):	Kasper Kirkeby Jacobsen og Mads Mikkelsen
Godkendes af:	Torben Gregersen
Projektnummer:	12042
Dokument-id: (filnavn)	Strukturering og design.pdf
Antal sider:	49
Kunde:	Aarhus Universitet Ingeniørhøjskolen

Sted og dato:

_____
Torben Gregersen

_____
Kasper Kirkeby Jacobsen (09588)

_____
Mads Havgaard Mikkelsen (09636)

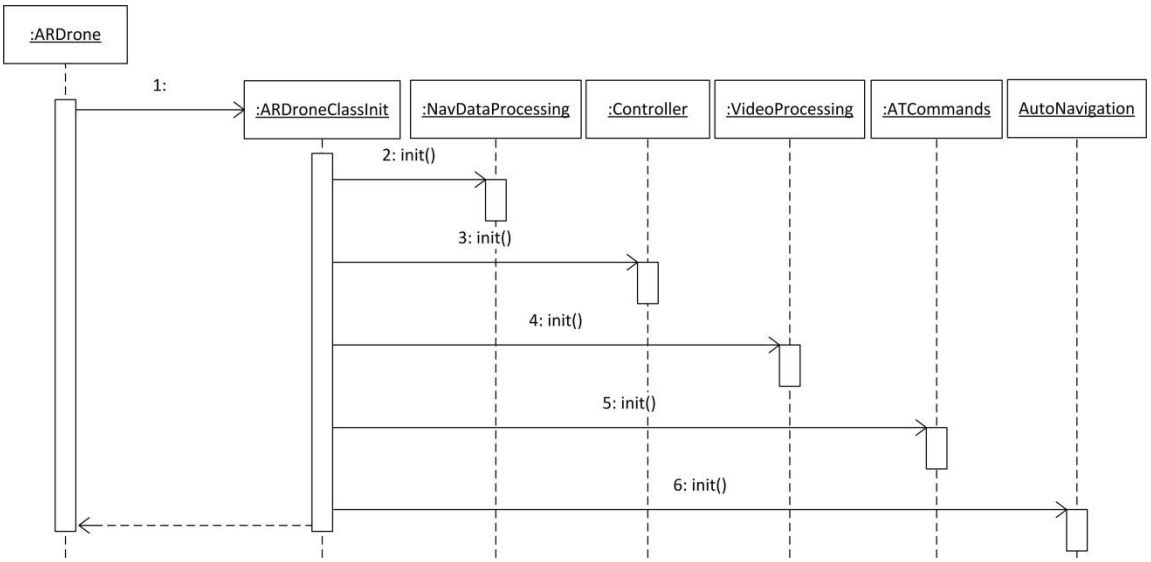
1 Logical view

I følgende afsnit beskrives logical view i dette projekt. Der er en beskrivelse af sekvensdiagrammer for hver use case, samt et statisk klasse diagram, der er lavet på baggrund af sekvensdiagrammerne.

1.1 Sekvensdiagrammer

I følgende afsnit beskrives de forskellige use cases fra kravspecifikationen. Hver use case har et sekvensdiagram der er vist og beskrevet i dette afsnit.

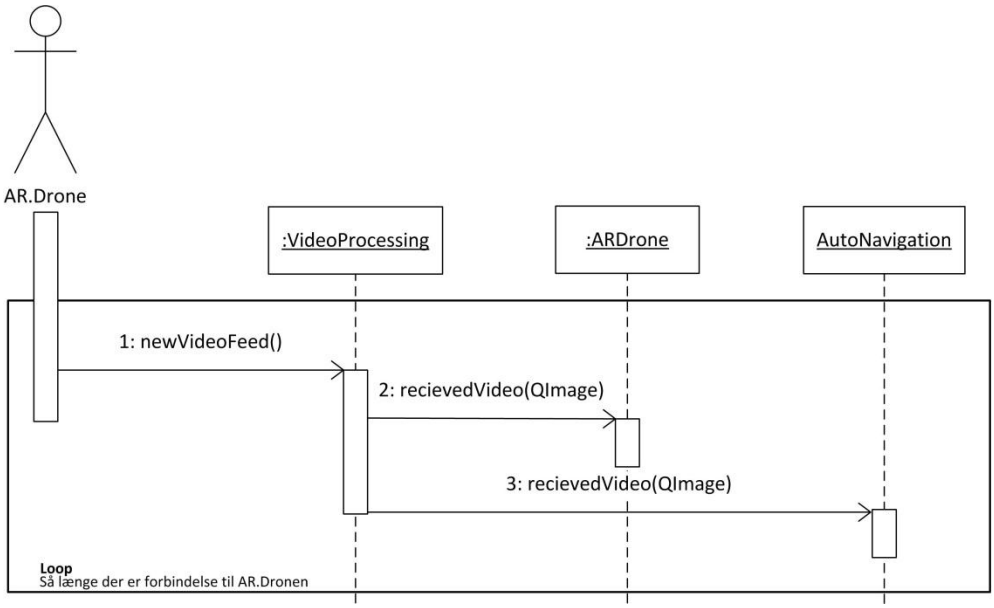
1.1.1 System initialisering



Figur 1 - Sekvensdiagram system initialisering

Figur 1 viser initialiseringen af de forskellige klasser i programmet, når programmet startes. Hele initialiseringen af de klasser der har kommunikation og indflydelse på AR.Dronen, samt de klasser der påvirker programmet fra AR.Dronens inputs, håndteres af klassen ARDroneClassinit.

1.1.2 Opdater video feed

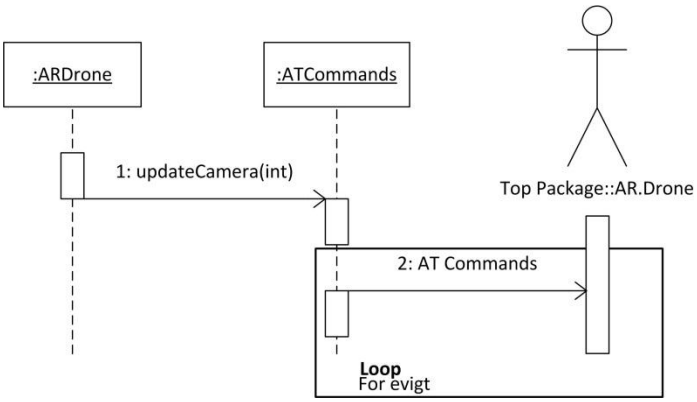


Figur 2 - Sekvensdiagram opdater video feed

AR.Dronen sender kontinuert dens video feed til den klient der er forbundet til den, og har anmodet om video feed. Video feedet fra AR.Dronen decodes fra data til billede af VideoProcessing klassen, der sender billedet til klassen ARDrone, hvorefter ARDrone klassen opdaterer det grafiske brugerinterface med det nye billede. Hvis AutoNavigation er enabled, hvor AR.Dronen følger en linje på gulvet, sendes billedet også til AutoNavigation.

Billedet sendes til ARDrone og AutoNavigation klasserne vha. et signal fra VideoProcessing. Dette signal har ARDrone og Autonavigation klasserne forbundet til et slot tilhørende klasserne

1.1.3 Skift video kamera



Figur 3 - Sekvensdiagram skift video kamera

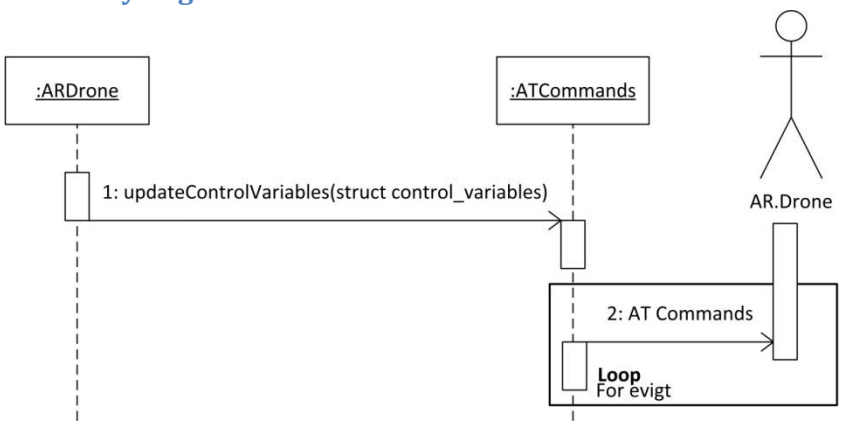
AR.Dronen har to kameraer, et i front, og et i bunden. Det er muligt at ændre video feedet fra AR.Dronen, så der enten modtages fra bund kameraet, front kameraet eller mixet (split screen m.m.). Dette videofeed ændres ved at ControllerProcessing klassen og ARDrone klassen, kalder funktionen updateCamera, når brugeren vil ændre kamera input. updateCamera sender derefter AT kommandoer til AR.Dronen om at ændre video feedet.

ATCommands klassen indeholder en tråd, hvori der sendes data til AR.Dronen i en evig løkke. Alle kommandoer der ønskes sendt til AR.Dronen, går igennem ATCommands klassen. For at sende kommandoer til AR.Dronen fra andre klasser end ATCommands klassen, kaldes ATCommands klassens funktioner.

updateCamera funktionen opdaterer en lokal video variable der indeholder en integer der beskriver hvilket video feed der er ønsket. Denne variabel kan indeholde fra 0-3, da der er 4 forskellige kamera feeds. I tråden, hvori den evige løkke kører, detekteres at videovariablen har ændret sig, og sender den nye værdi af videovariablen til AR.Dronen.

Samme metode med at ændre en variabel, der derefter detekteres inde i den uendelige løkke, hvorefter ændringen sendes til AR.Dronen, bruges i alle sekvenser der kommunikerer med AR.Dronen.

1.1.4 Styring af AR.Dronen med touchskærm



Figur 4 - Sekvensdiagram touchskærm manøvrering

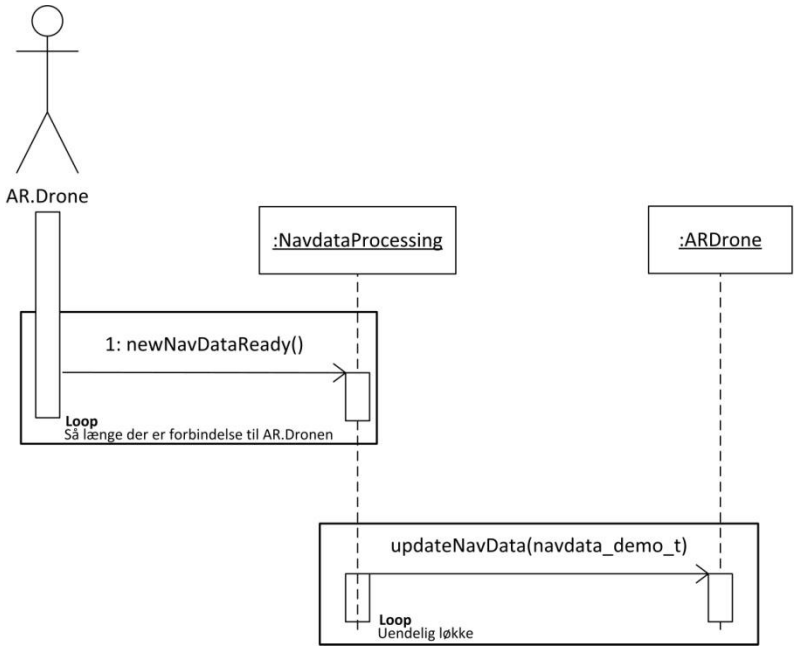
Når der trykkes på en knap i programmet til styring af AR.Dronen, kaldes funktionen updateControlsVariables i ATCommands, der sender de nye styringsvariabler til AR.Dronen. Disse styrings variabler gemmes i en struct og indeholder alle nødvendige data til at styre AR.Dronen, x y og z retningen, op, ned samt land, let, rotation og emergency. Structen der indeholder styringsvariabler skal indeholde variabler som vist I Tabel 1:

Tabel 1 - Oversigt over styrings variabler

Variabel	Beskrivelse
float phi	Højre venstre (-1.0 til 1.0)
float theta	Frem og tilbage (-1.0 til 1.0)

float yaw	Rotation om sig selv. (-1.0 til 1.0)
float gaz	Op og ned (-1.0 til 1.0)
int camera	Camera variablen skifter kamera feed fra AR.Dronen. Camera kan være fra 0-4 0: Front kamera 1: Bund kamera 2: Front kamera med småt bund kamera i venstre hjørne 3: Bund kamera med småt front kamera i venstre hjørne
bool takeoff	Takeoff variables sættes til true for at lette AR.Dronen
bool landing	Landing variablen sættes til true for at lande AR.Dronen.
bool emergency	Variabel der toggle emergency tilstanden. Sættes til true for at toggle tilstanden

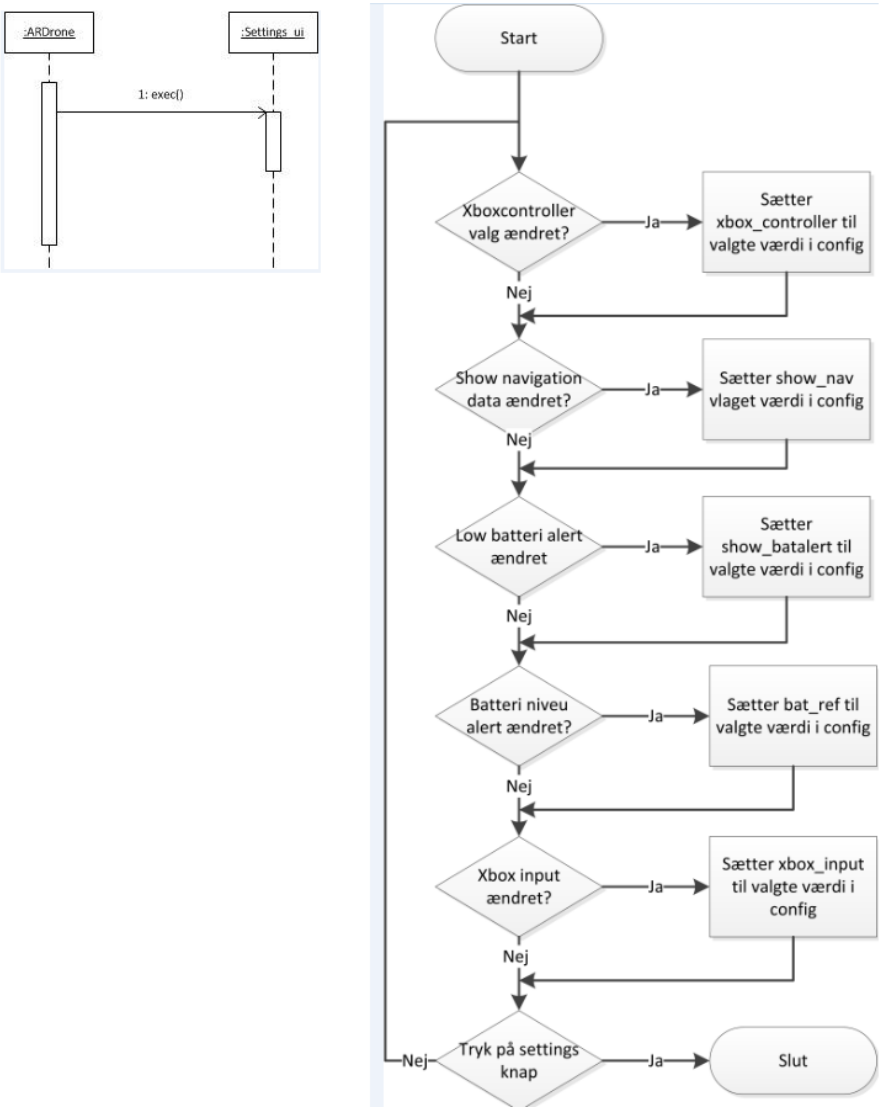
1.1.5 Opdater navigationsdata



Figur 5 - Sekvensdiagram opdater navigationsdata

Så længe klienten har forbindelse til AR.Dronen, sender den navigationsdata, og dette modtages modtages i programmets NavdataProcessing klasse. NavdataProcessing kører en uendelig løkke, hvor den behandler navigationsdata der modtages fra AR.Dronen. Den behandler altid den sidste navigationsdata der modtages, dvs. hvis der modtages 2 data pakker, før den er færdig med at behandle den foregående, behandles kun de nyeste navigationsdata og de ældste navigationsdata bliver kasseret.

1.1.6 Skift settings

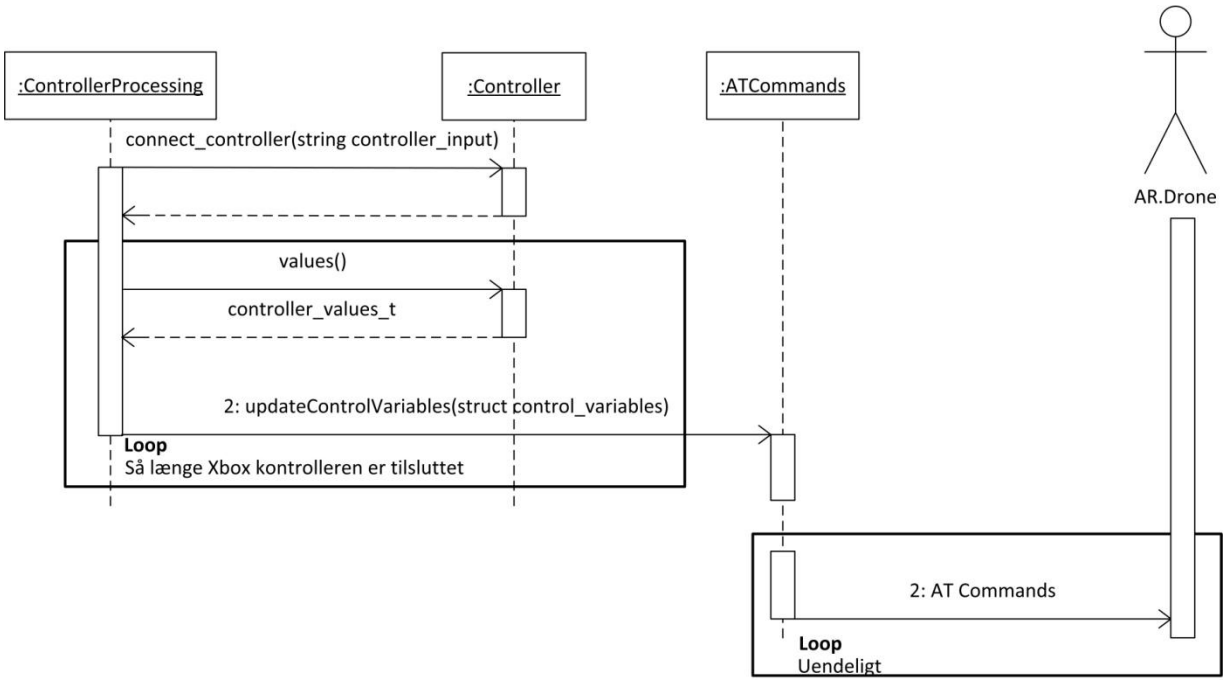


Figur 6 – Sekvensdiagram og aktivitetsdiagram skift settings

Figur 6 viser sekvensen og aktivitetsdiagram, når indstillingsmenuen åbnes. Til venstre i Figur 6 ses sekvensdiagrammet, og til højre ses aktivitetsdiagrammet.

Aktivitetsdiagrammet i Figur 6 viser logikken i indstillingsmenuen. Logikken og sekvensen bliver startet, når der trykkes på indstillingsknappen i toppen af højre hjørne i programmet, og lukkes igen når der trykkes på Luk eller Gem knap. Alle valg der foretages i indstillingsmenuen skal opdateres efter indstillingsmenuen lukkes. Indstillingerne skal gemmes, så computeren husker på dem til næste gang programmet startes op.

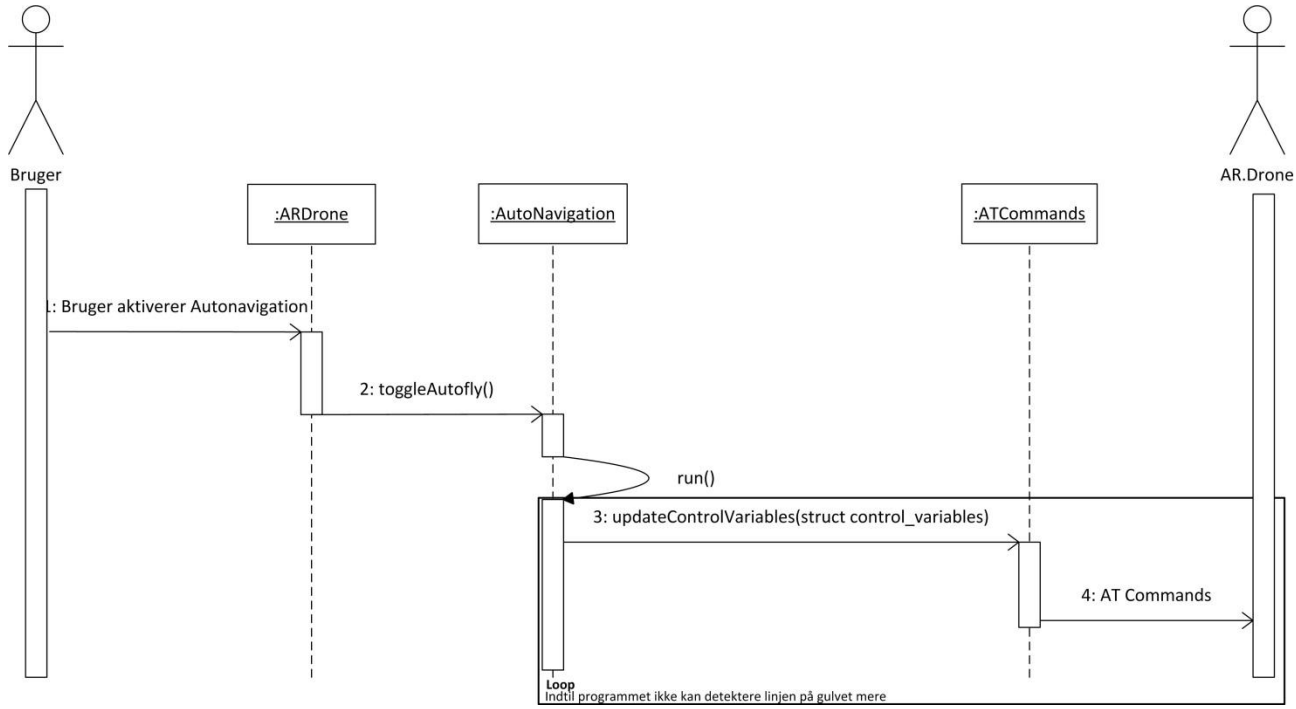
1.1.7 Styring af AR.Drone med Xbox kontroller



Figur 7 - Sekvensdiagram Xbox kontroller manøvrering

I Figur 7 vises hvordan inputs fra Xbox kontrolleren skal behandles og oversættes til styringsvariabler, som derefter skal sendes til AR.Dronen via. ATCommands klassen. Styringsvariabler sendes fra ControllerProcessing til ATCommands vha. funktionen updateControlVariables, der som parameter har en struct, og indeholder alle styringsvariabler. Fra ATCommands klassen sendes styringsvariablerne til AR.Dronen.

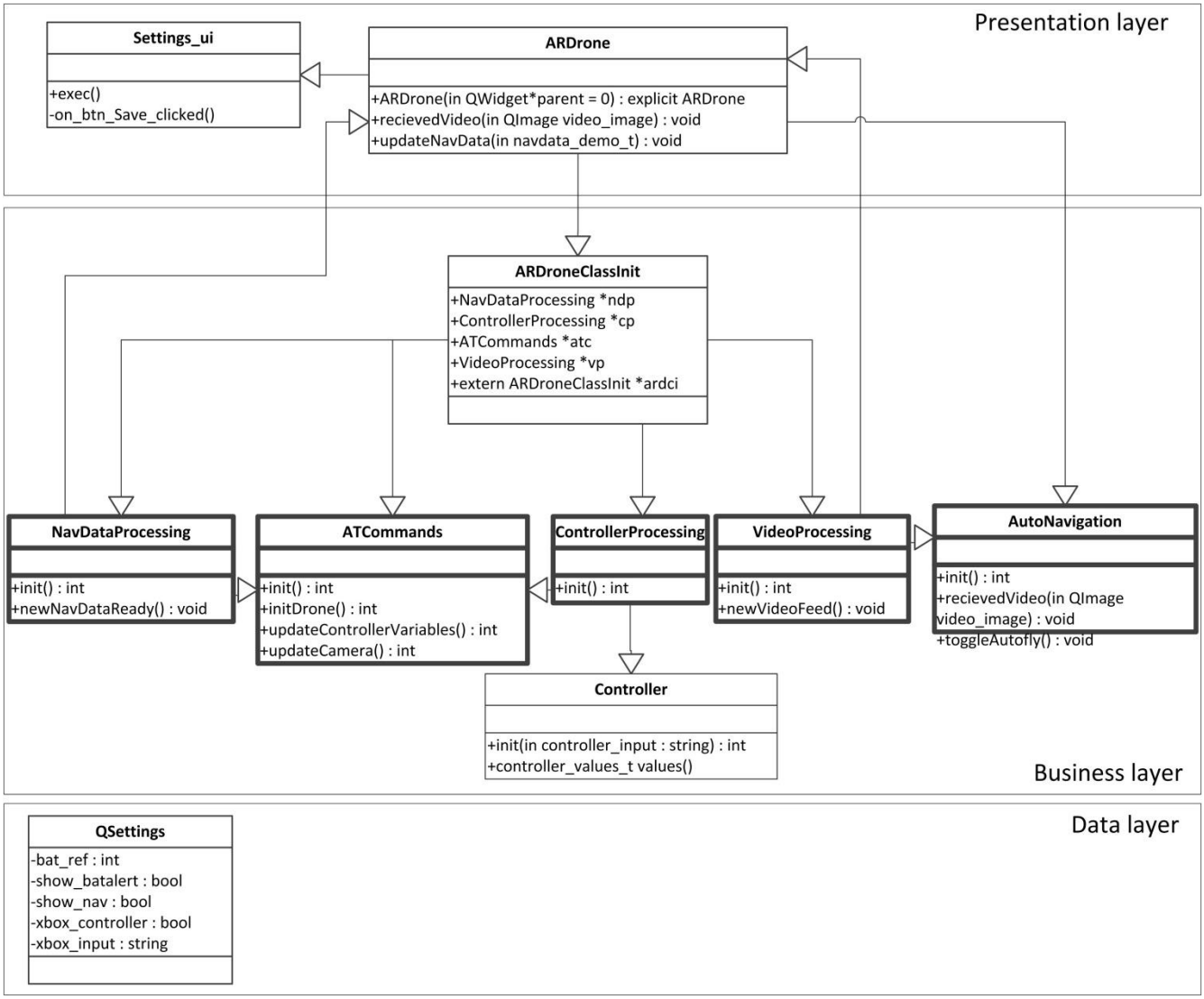
1.1.8 Auto-navigation efter linje



Figur 8 - Auto-navigation efter linje på gulvet

Når brugeren aktiverer autonavigation i programmet, initaliseres AutoNavigations klassen. Under initaliseringen af AutoNavigations klassen nulstilles diverse variable, samt et uendelig loop startes. AutoNavigation klassen består af et loop, der kører uafbrudt, og behandler videofeedet fra AR.Dronen for at kunne navigere ud fra en markeret linje på gulvet. Da AR.Dronen skal navigeres efter en linje på gulvet, er det derfor nødvendigt, at der i initialiseringen af AutoNavigation klassen skiftes videofeed, så AR.Dronen streamer fra bundkameraet. Løkken afbrydes først, når brugeren stopper den ved at trykke på den knap i programmet, der startede autonavigationen. Viser signalbehandlingen af videofeedet, at der er en linje under AR.Dronen, navigeres ud fra denne linje ved at kalde funktionen updateControlVariables i ATCommands klassen.

1.2 Statisk klassediagram



Figur 9 - Statisk klasse diagram

Figur 9 viser det statiske klasse diagram for programmet, og er bygget på grundlag af de funktioner og strukturen i sekvensdiagrammerne. Klasserne, som er markeret med fed kant, er selvkørende, og har en løkke der holder dem kørende.

Bussinesslayer vil fremover blive refereret til som styringsklasserne. Business layer er bygget op, så de samlet kan bruges i ethvert andet Qt projekt, og er derfor ikke afhængige af klasserne i presentation- og data layer.

1.2.1 Presentation layer

Presentation layer indeholder klasserne der præsenterer indhold m.m. for brugeren. I dette projekt bruges en grafisk bruger overflade i form af et display til præsentationsmidlet.

1.2.1.1 ARDrone

ARDrone klassen styrer programmets grafiske interface, både det der vises på skærmen samt de inputs der kommer fra brugeren, som f.eks. tryk på knapper m.m.. ARDrone klassen indeholder 3 funktioner der påvirkes af andre klasser:

- ARDrone(in QWidget*parent = 0)
- recievedVideo(in QImage video_image)
- updateNavData(in navdata_demo_t)

1.2.1.1.1 ARDrone konstruktor

ARDrone() er konstruktoren der kaldes, når der oprettes en instans af ARDrone klassen. I konstruktoren initialiseres styringsklasserne ved at oprette en instans af klassen ARDroneClassInit.

1.2.1.1.2 recievedVideo

recievedVideo er et slot, der er forbundet til signalet der sendes, når der kommer nyt videofeed fra AR.Dronen. Funktionen har til formål at opdatere programmet og præsentere brugeren for det seneste videofeed.

1.2.1.1.3 updateNavData

updateNavdata er et slot, der er forbundet til signalet der sendes, når der er ny navigations data fra AR.Dronen. Funktionens formål er at modtage navigationsdata og præsentere det for brugeren igennem programmets grafiske brugerflade.

1.2.2 Business layer

I business layer behandles data til og fra AR.Dronen, samt Xbox kontrollerens input.

1.2.2.1 ARDroneClassInit

Klassen ARDroneClassInit bruges til at initiatialisere styringsklasserne i programmet. ARDroneClassInit klassen er den eneste klasse, der skal initialiseres hvis der ønskes at gøre brug af dette projekts styringsklasser. I dette projekt kaldes ARDroneClassInit fra ARDrone klassens konstruktor funktion.

1.2.2.2 ATCommands

ATCommands klassen kører et uendeligt loop der sørger for at sende styringskommandoer til AR.Dronen hele tiden (hvert 20ms). Klassen indeholder en init() funktion, der sørger for at oprette en socket til AR.Dronen og forbinder diverse slots til signaler.

1.2.2.2.1 initDrone

initDrone er en funktion, hvorfra der sendes initialiserende kommandoer til AR.Dronen. Igennem initDrone sendes AR.Dronen ud af bootstrap tilstand, så den sender navigations data demo. Funktionen bliver kaldt fra klassen NavDataProcessing.

1.2.2.2.2 updateControllerVariables

updateControllerVariables bruges til at opdatere de ATCommands lokale styringvariabler, som sendes til AR.Dronen. Funktionen bliver kaldt enten fra ControllerProcessing-, AutoNavigation- eller fra ARDrone klassen.

1.2.2.2.3 updateCamera

updateCamera bruges til at skifte kamera outputtet fra AR.Dronen. Funktionen bliver kaldt fra enten ControllerProcessing eller fra ARDrone klassen.

1.2.2.3 ControllerProcessing

ControllerProcessing klassen kører et konstant loop i en tråd, hvori der tjekkes om der er ændringer i inputtet fra Xbox kontrolleren. Når der er nye inputs fra Xbox kontrolleren, processeres disse inputs om til styringsvariabler og sendes til ATCommands klassen.

1.2.2.4 Controller

Controller klassen initialiserer Xbox kontrolleren og kommunikerer med denne. Controller klassens funktion values() kaldes fra ControllerProcessing klassen når der ønskes at der aflæses inputs fra Xbox kontrolleren. values() funktionen aflæser Xbox kontrollerens input og returnerer dem i structen controller_values_t, der indeholder alt information omkring analog inputs og digitale inputs (knapper).

1.2.2.5 VideoProcessing

VideoProcessing klassen kører i et uendeligt loop, idet klassen konstant modtager videodata fra AR.Dronen. Klassen laver et færdigt billede af dataen der modtages fra AR.Dronen, og det færdige billede sendes til ARDrone klassen og AutoNavigation klassen vha. et signal.

1.2.2.6 AutoNavigation

AutoNavigation initialiseres når der ønskes, at AR.Dronen skal navigere efter en linje på gulvet. AutoNavigation klassen kører i et uendeligt loop, indtil brugeren deaktiverer funktionen.

1.2.3 Datalayer

1.2.3.1 QSettings

De eneste data der gemmes i programmet, er indstillingerne der bruges i programmet. Disse indstillinger er at finde under QSettings klassen, der er en integreret Qt klasse, der gør det nemt at gemme indstillinger på computerens hukommelse. QSettings skal for at kunne genkende indstillinger til det specifikke program, have opgivet en organisations- og applikationsnavn.

Organisation	**Applikation**
ARDrone	Settings

De forskellige indstillinger der indgår i programmet er forklaret i Tabel 2.

Tabel 2 Indstillings oversigt

Navn	**Beskrivelse**
bat_ref	Referencen på minimum værdi batteriet skal have før den viser alarm.
show_batalert	Enable/Disable indstilling der indikerer om brugeren vil alarmeres hvis batteriet niveau falder til under bat_ref.
show_nav	Enable/Disable indstilling der indikerer om brugeren vil have vist

	navigationsdata på frontskærmbilledet.
xbox_controller	Enable/Disable indstilling hvor brugeren kan deaktivere eller aktivere Xbox kontrolleren.
xbox_input	Stien til Xbox input filen gemmes I denne indstilling.

2 Process view

Process view beskriver kommunikationen imellem klasserne. Dette afsnit vil beskrive funktionskald og signaler imellem klasserne.

Slots og signaler er en fordel at bruge i de tilfælde hvor en funktion skal kalde flere funktioner, ved f.eks nye inputs såsom video. Ved at lave et signal, kan der statisk kobles slots op til signalet som automatisk kaldes når signalet sendes. Når signalet sendes, bliver alle de funktionerne der er koblet op til signalet kaldt, hvilket gør det til en fleksibel løsning, da man altid kan tilslutte nye slots til signalet. Dette gør det nemt at tilslutte styringsklasserne i et andet projekt, da man bare skal koble nye slots op på de signaler som styringsklasserne sender.

I dette afsnit, vil alle slot og funktions kald beskrives. På tegningerne i det kommende afsnit, beskriver pilen hvor fra funktionen/slottet kaldes.

2.1 Funktionskald

2.1.1 initARDrone

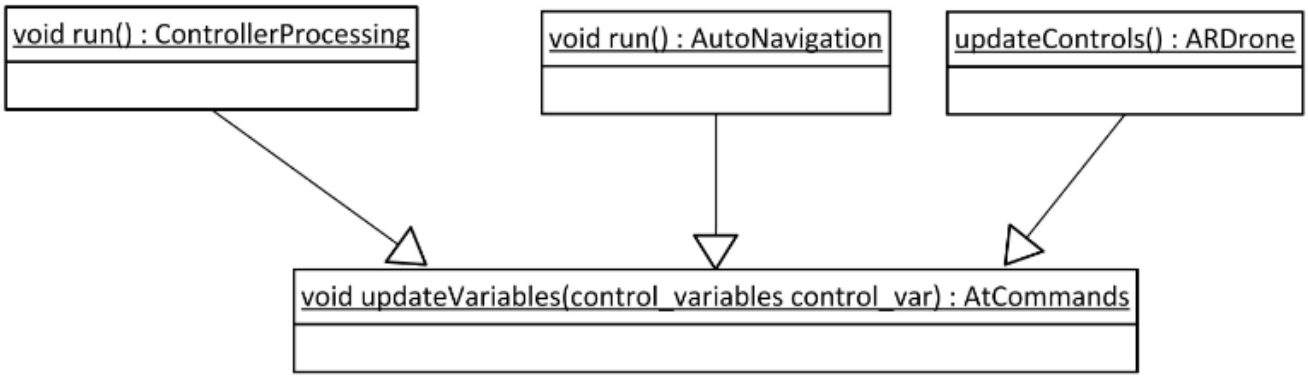
Figur 10 - initARDrone signal til at få AR.Dronen ud af bootstrap

Figur 10 viser hvor AR.Dronen initialiseres fra bootstrap mode og til navigationsdata demo mode. Funktionen tager en integer som parameter, der beskriver hvilket step i initialiseringen funktionen skal foretage sig. Der er to steps der kan foretages:

1: Gå ud af bootstrap mode og aktiver navigationsdata demo mode.
2: Acknowledge.

Funktionen kaldes fra NavDataProcessings hovedtråd, run()

2.1.2 newControllerInputs

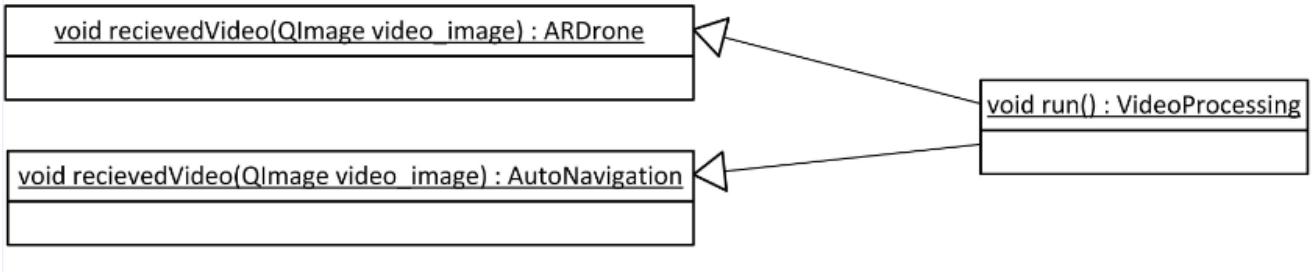


Figur 11 - newControllerInputs signal til manøvrering af AR.Dronen

Figur 11 viser funktionerne der opdaterer styringsvariablerne fra programmets grafisk interface, fra Xbox kontrolleren og fra AutoNavigations klassen. Funktionen updateVariables kaldes når brugeren ønsker at udføre en ny manøvre til AR.Dronen. Funktionen updateVariables tager styringsvariablerne som parameter.

updateVariables() kaldes fra ControllerProcessing, hvis en Xbox kontroller er tilsluttet, og der ændres på dennes input (analoge sticks, knapper mm.). updateVariables() kaldes også fra ARDrone klassen, der håndterer inputs fra skærmen. Funktionen kaldes fra AutoNavigations klassen, når denne er aktiv, og der navigeres efter en linje på gulvet.

2.1.3 newVideoData

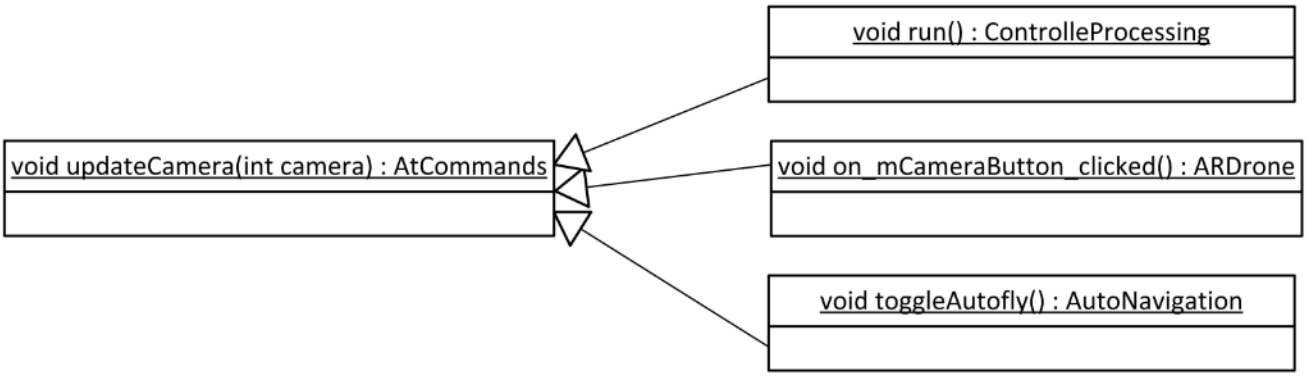


Figur 12 - newVideoData signal til ny video fra AR.Dronen

Figur 12 viser funktionerne der kaldes når der er kommet nyt video feed fra AR.Dronen. Funktionerne har QImage som parameter, der indeholder det seneste video feed fra AR.Dronen. Funktionerne kaldes fra VideoProcessing klassen, hvor video feedet fra AR.Dronen først behandles, og derefter bliver brugt som parameter. Funktionerne kaldes over et signal VideoProcessing sender.

Funktionen receivedVideo modtager QImage fra VideoProcessering og opdaterer programmet grafiske interface med dette QImage, der er det seneste video feed der er modtaget fra AR.Dronen. Derved viser skærmen altid det nyeste video feed fra AR.Dronen. AutoNavigation klassen har funktionen recievedVideo der modtager et QImage fra VideoProcessing, for at kunne lave videobehandling på videofeedet. Fjernstyringsprogrammet skal navigere AR.Dronen efter en linje på gulvet vha. videobehandlingen af det modtagende videofeed.

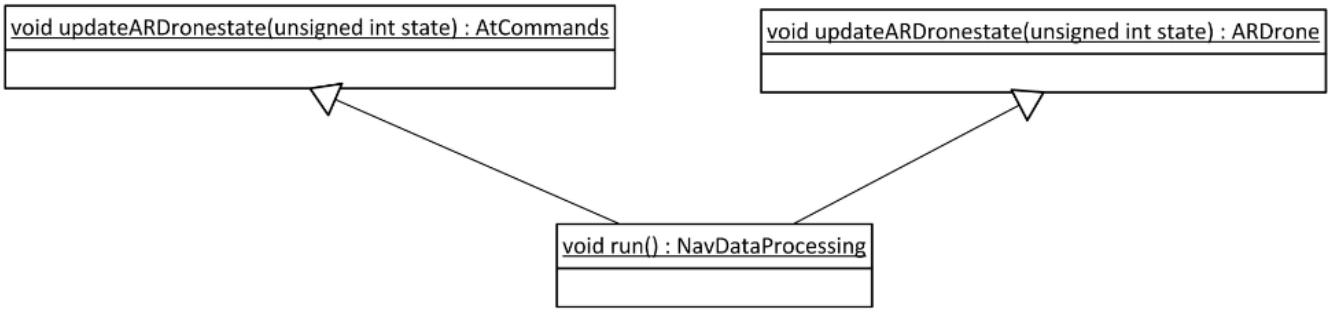
2.1.4 newCamera



Figur 13 - updateCamera bruges til at ændre AR.Dronen video feed til en anden kamera vinkel

Figur 13 beskriver funktionskaldene af updateCamera, der kaldes når brugeren ønsker at ændre video feedet fra AR.Dronen. Der er 4 video feeds at vælge imellem, og dette valg af video feed sendes med som parameter. Funktionen updateCamera er i klassen ATCommands, hvorfra der sendes kommando til AR.Dronen om at skifte video feed. Funktion kaldes fra ARDrone klassen, hvorfra brugeren kan skifte video feed ved at trykke på en knap på det grafiske interface. AutoNavigationens kalder updateCamera når autonavigations startes, for at skifte kamera, så der modtages videofeed fra bundkameraet, som der derefter kan navigere ud fra. updateCamera kan også opdateres fra Xbox kontrolleren, og funktionen kaldes derfor også fra ControllerProcessering.

2.1.5 newARDronestate

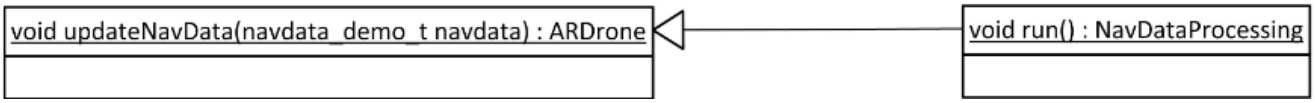


Figur 14 - newARDronestate signalet opdaterer klasserne med AR.Dronens status

Figur 14 viser funktionerne der kaldes når AR.Dronen har skiftet tilstand. Funktionerne bruges til at opdatere diverse variabler i programmet med AR.Dronen's tilstand, så de forskellige klasser kender til tilstanden. Funktionerne tager en unsigned int som parameter, hvor de forskellige bits i denne integer beskriver AR.Dronen's tilstand. Disse tilstande kan ses under Deployment views afsnittet 3.2.3.2 AR.Drone state.

Funktionerne i ATCommands og i ARDrone klasserne, bruges til at holde styr på om AR.Dronen flyver eller er i emergency, og for at opdatere displayet, så brugeren bliver informeret, hvis tilstanden ændrer sig kritisk i form af emergency eller lign..

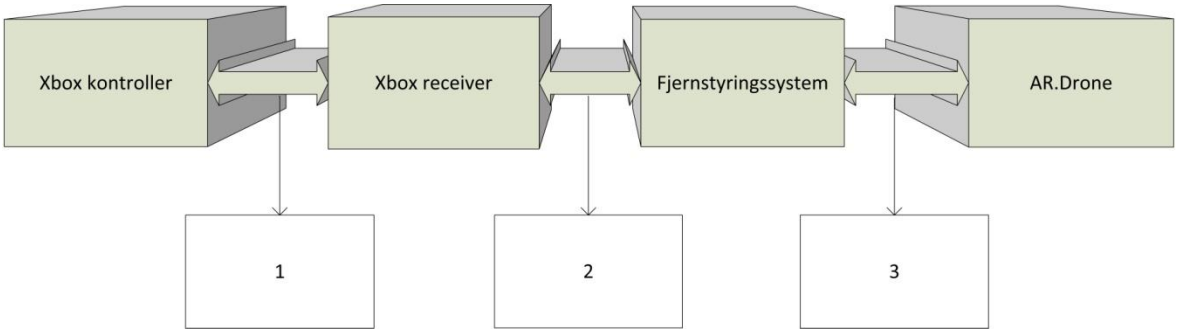
2.1.6 newNavData



Figur 15 - newNavData signalet der indeholder navigationsdata fra AR.Dronen

updateNavdata() har parameteren navdata_demo_t, der er en struct som indeholder navigationsdata demo fra AR.Dronen. Funktionen kaldes fra NavDataProcessing, når der modtages ny navigations data fra AR.Dronen. Funktionen updateNavdata() er i ARDrone klassen og opdaterer displayet med navigationsdata fra AR.Dronen så de er synlige for brugeren. updateNavData kaldes når NavDataProcessing sender signal ud, om at der er kommet ny navigationsdata.

3 Deployment view



Figur 16 - Deployment view diagram

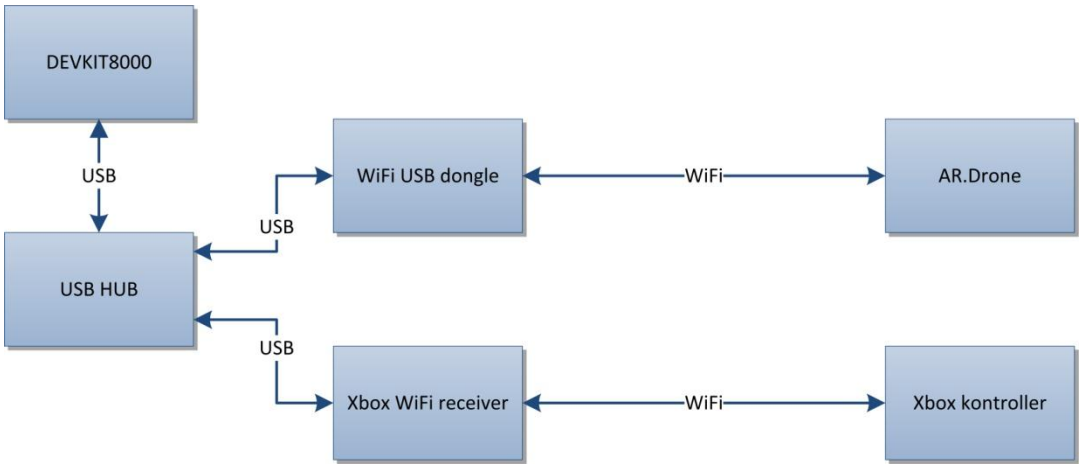
Fjernstyringssystemet i Figur 16 består af DevKit8000, WiFi modul og USB hub.

AR.Dronen kommunikerer med Devkit8000 over WiFi via den WiFi USB dongle der er sat i DevKit8000's USB hub. AR.Dronen bruger et standard 802.11 (2.4GHz) WiFi netværk, dette er repræsenteret med pil 3 på Figur 16.

Xbox kontrolleren kommunikerer med fjernstyringssystemet over en Xbox wireless receiver der er sluttet til DevKit8000's eksterne USB hub. Der bliver brugt et separat 802.11 (2.4GHz) WiFi netværk af Xbox kontrolleren og receiveren. Xbox kontrollerens kommunikation til Xbox wireless receiver er repræsenteret med pil 1 på Figur 16, og Xbox wireless receiver til fjernstyringssystem med pil 2.

Både WiFi USB dongle og Xbox WiFi receiver er sluttet til DevKit8000's eksterne USB hub med USB2.0 A kabler.

Mellem USB hub og fjernstyringssystemet er der et USB2.0 A kabel.



Figur 17 - HW grænseflader

3.1 Protokoller

3.1.1 AT kommandoer fra fjernstyrings systemet til AR.Dronen

Kommandoer til AR.Dronen sendes som at-kommandoer i en string. Stringen må maximalt være 1024 bytes lang ifølge AR.Dronens developer guide [Bilag 1]. Alle kommandoer skal indeholde et sekvensnummer som en del af kommandoen, for at AR.Dronen ikke ved en fejl eksekverer gamle kommandoer. Sekvensnummeret skal inkrementeres for hver ny kommando der sendes. Tabel 3 viser en liste over AT-kommandoer AR.Dronen kan modtage.

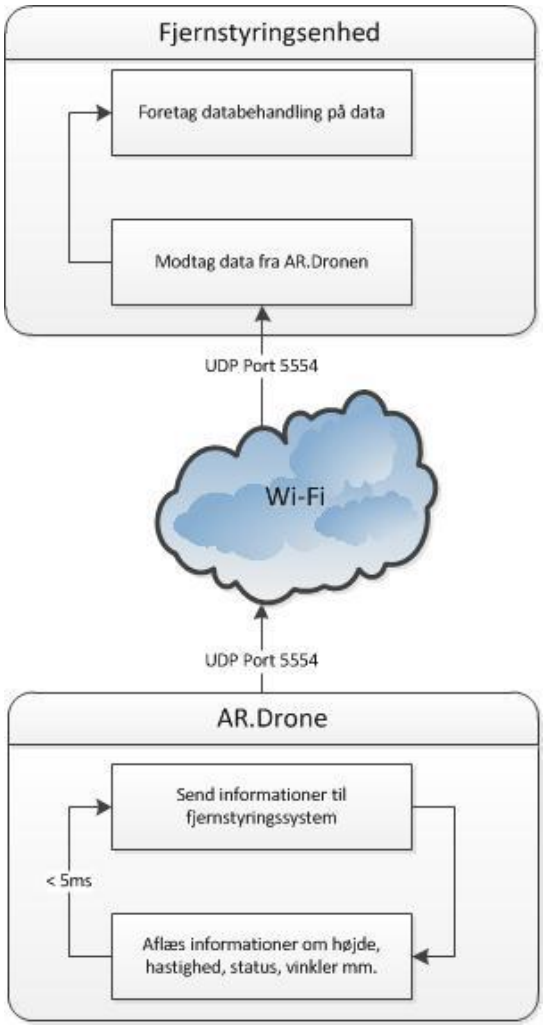
Tabel 3 - AT-kommandoer

Kommando	**Byte array sendt til AR drone**
Sæt AR.Dronen til navigationsdata demo	"AT*CONFIG=" + sekvensnummer ", \"general:navdata_demo\",\"TRUE\"\r"
Konfigurations parameter til AR.Dronen. Alle konfigurations parametrene kan ses i AR.Dronens Developer Guide [Bilag 1]	"AT*CONFIG=" + sekvensnummer ", \"parameter\",\"TRUE\"\r"
Acknowledge af modtaget navigationsdata demo	"AT*CTRL=" + sekvensnummer + ",0"
Takeoff	"AT*REF=" + sekvensnummer + ",290718208\r"
Landing	"AT*REF=" + sekvensnummer + ",290717696\r"
Lad AR.Dronen svæve	"AT*PCMD=" + sekvensnummer +",0,0,0,0,0\r"
Bevægelse af AR.Dronen	"AT*PCMD=" + sekvensnummer + ",1," + Phi + "," + Theta + "," + Gaz + "," + Yaw +"\r" Phi, Theta, Gaz og Yaw er tal mellem -1 og 1 gemt i en double.
Ændring af kamera	"AT*CONFIG=" + sekvensnummer + ",\"video:video_channel\",\"" + feed_nummer + "\"\r" Feed_nummer er en int mellem 1 og 4
Emergency mode toggle	"AT*REF=" + sekvensnummer + ",2300838144\r";

3.2 Navigations data

3.2.1 Overblik

AR.Dronen sender periodisk navigationsdata på UDP port 5554. Disse navigationsdata bliver sendt med et interval på mindre end 5ms.
Hvert modtaget navigationsdata indeholder forskellige informationer omkring AR.Dronens tilstand som f.eks. vinkler, højdemål, hastighed, aktiv kamera, batteriniveau og mange andre informationer. Disse data vil alle blive listet senere i afsnittet. Figur 18 er det overordnede system for navigationsdata kommunikation mellem AR.Dronen og programmet. Figur 18 viser kommunikations interfacet og den overordnede processering af data i de to systemer.



Figur 18

3.2.2 Initialisering

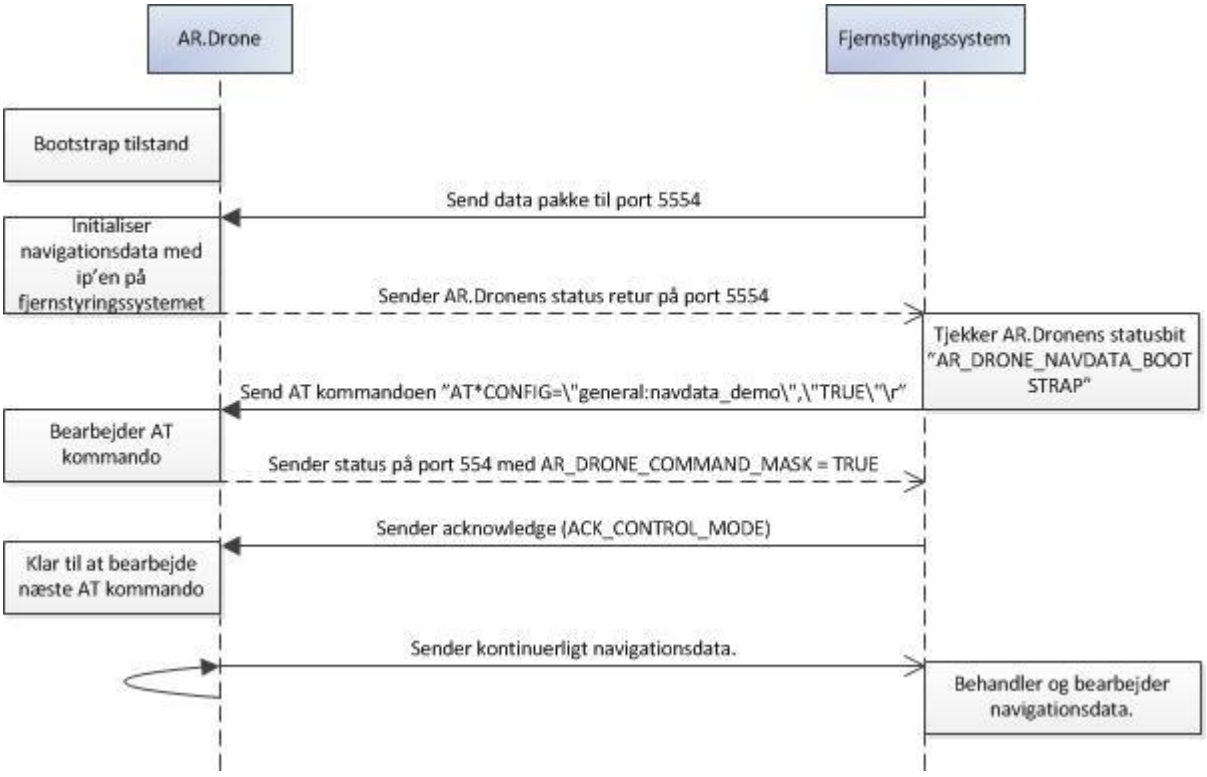
For at modtage navigationsdata er det nødvendigt at initialisere AR.Dronen. For at initialisere og aktivere navigationsdata på UDP Port 5554, skal der på samme port sendes en handshake der består af en pakke med vilkårlig data. Ved at foretage en handshake bliver AR.Dronen opmærksom på at klienten vil modtage navigationsdata og lytter på porten.

AR.Dronen kan have tre tilstande:

- Bootstrap tilstanden, hvor det kun er status og sekvens tælleren der bliver sendt
- Almindelig tilstand hvor kun navigationsdata demo bliver sendt.
- Debug tilstand, hvor alt navigationsdata sendes med diverse debug informationer.

De forskellige navigationsdata beskrives i Tabel 7 i afsnittet 3.2.3.5 id. I langt de fleste tilfælde er det den almindelige tilstand der ønskes, da det giver klienten informationer omkring AR.Dronens tilstand m.m..

For at komme ud af boostrap tilstanden og over i den almindelig tilstand, skal AR.Dronen initialiseres korrekt. AR.Dronen initaliseres korrekt hvis fremgangsmåden i Figur 19 følges.



Figur 19

Figur 19 viser sekvensdiagrammet over initialiseringen og opstarten af navigationsdataen. Når initaliseringen af AR.Dronen er fuldført, sender AR.Dronen kontinuert data til programmet (< 5ms).

3.2.3 Data

Navigationsdata der modtages fra AR.Dronen har struktur som illustreret i Tabel 4. Navigationsdata er gemt og sendt i little endian – med det menes at de første bytes i dataen (læst fra venstre) er LSB og de sidste bytes er MSB. ”Header”’ens bytes er derfor LSB, og ”cks data”’ens bytes er MSB. At dataene er i little-endian har ikke nogen indflydelse på den måde data behandles, da data alligevel skal deles op og ikke læses som en numerisk værdi.

Tabel 4 Data struktur for navigationsdataen

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

3.2.3.1 Header

Header værdien er en fastsat 32-bit værdi, som i hex er 0x55667788. Denne del af dataene bruges til at verificere, at det er navigationsdata man har modtaget, og at der ikke er sket en fejl i overførslen fra AR.Dronen til programmet. Derfor skal denne værdi altid tjekkes til at være 0x55667788, før dataene bearbejdes.

3.2.3.2 AR.Drone state

AR.Dronens tilstand er opgivet i en 32-bit værdi. Hver bit beskriver en tilstand i AR.Dronen. De forskellige bit nr. beskriver tilstanden, som er angivet i Tabel 5

Tabel 5 AR.Dronen tilstands tabel

Bit nr.	**Tilstand**
0	Flyvning (0 = Landet , 1 = Flyver)
1	Video (0 = Slået fra, 1 = Slået til)
2	Vision (0 = Slået fra, 1 = Slået til) (Front kamera??????)
3	Kontrol (0 = kontrolleret efter eulers vinkel , 1 = kontrollere efter vinkel hastigheden)
4	Højde (0 = højdemåler kontrol slået fra, 1 = højdemåler kontrol slået til)
5	Bruger tilbagesvar (Start knap tilstand, 0 = ikke trykket ned, 1 = trykket ned)
6	Kommando acknowdlege (0 = ingen modtaget, 1 = en modtaget)
7	Firmware (1 = Firmware er OK)
8	Firmware opdatering (1 = Der er nyere firmware tilgængelig)
9	Firmware opdaterer (1 = AR.Dronen er ved at opdatere firmwaren)
10	Navigationsdata Demo (0 = Sender alt navigationsdata, 1 = Sender kun navigationsdata demo) (Er navigationsdata demo slået til sender den en reduceret mængde data, er den slået fra sender den alt information som indeholder meget debugger information)
11	Navigationsdata bootstrap (0 = Options er sendt, 1 = Ingen navigationsdata options er sendt)
12	Motor (0 = OK, 1 = Motor fejl)
13	Kommunikation (0 = OK, 1 = Kommunikation er tabt)
14	

15	Lavt batteri (0 = OK, 1 = Lavt batteri)
16	Bruger nødstilfælde (1 = Bruger nødlanding aktiv)
17	Timer udløbet (0 = OK, 1 = Udløbet)
18	
19	Vinkler (0 = OK, 1 = Vinklen ude af rækkevidde)
20	Vind (0 = OK, 1 = For meget vind til at flyve)
21	Ultralyd (0 = OK, 1 = Defekt ultralyd)
22	Cutout (0 = OK, 1 = Cutout detected)
23	PIC version (0 = OK, 1 = Forkert PIC versions nummer)
24	AT Codec (0 = Slået fra, 1 = Slået til)
25	Navigationsdata tråd (0 = Slået fra, 1 = Slået til)
26	Video tråd (0 = Slået fra, 1 = Slået til)
27	Acquisitions tråd (0 = Slået fra, 1 = Slået til)
28	Control watchdog (0 = OK, 1 = Forsinkelse på control eksekvering (> 5ms)
29	ADC watchdog (0 = OK, 1 = Forsinkelse i uart2 dsr (> 5ms)
30	Kommunikations watchdog (0 = OK, 1 = Kommunikations problemer)
31	Nødlanding (0 = OK, 1 = Nødlanding)

3.2.3.3 Sequence number

Sekvensnummeret er angivet i en 32-bit integer. Sekvensnummeret holder styr på kommunikationen mellem programmet og AR.Dronen, ved at tælle antal gange der er sendt navigationsdata. Ved hele tiden at holde styr på sekvensnummeret, sikrer programmet sig at den ikke læser gammel navigationsdata.

3.2.3.4 Option

Option delen af navigationsdatene indeholder følgende elementer som vist i Tabel 6:

Tabel 6

id	*size*	*data*
16-bit int	16-bit int	……..

Option indeholder de forskellige navigationsdata. Der kan være flere options i hver data sekvens der sendes, fra AR.Dronen, hvor hver option genkendes på dens ”id”.

3.2.3.5 id

Id'et består af 16-bit, der bruges til at genkende optionen. Der findes forskellige slags options, der er vist og beskrevet i Tabel 7. Alle structs der refereres til i Tabel 7, kan findes i [Bilag 3] i filen navdata.h

Tabel 7

id	**Navn**
0x0000	NAVDATA_DEMO_TAG

1	NAVDATA_TIME_TAG
2	NAVDATA_RAW_MEASURES_TAG
3	NAVDATA_PHYS_MEASURES_TAG
4	NAVDATA_GYROS_OFFSETS_TAG
5	NAVDATA_EULER_ANGLES_TAG
6	NAVDATA_REFERENCES_TAG
7	NAVDATA_TRIMS_TAG
8	NAVDATA_RC_REFERENCES_TAG
9	NAVDATA_PWM_TAG
10	NAVDATA_ALTITUDE_TAG
11	NAVDATA_VISION_RAW_TAG
12	NAVDATA_VISION_OF_TAG
13	NAVDATA_VISION_TAG
14	NAVDATA_VISION_PERF_TAG
15	NAVDATA_TRACKERS_SEND_TAG
16	NAVDATA_VISION_DETECT_TAG
17	NAVDATA_WATCHDOG_TAG
18	NAVDATA_ADC_DATA_FRAME_TAG
19	NAVDATA_VIDEO_STREAM_TAG
20	NAVDATA_GAMES_TAG
0xFFFF	NAVDATA_CKS_TAG

3.2.3.5.1 size

size er en 16-bit integer, hvis formål er at beskrive længden på *data* sektionen i Tabel 6. Size er størrelsen af bytes som *data* er, og bruges for at klienten ved, hvor mange bytes den skal læse for at have læst alt data.

3.2.3.5.2 data

data indeholder navigationsdata tilhørende det specificerede id. Hvert id har deres egen struct, som navigationsdataene læses over i. Disse structs kan ses i kildekoden i filen navdata.h [Bilag 3].

3.2.3.6 Checksum block

Checksummen for de sendte navigationsdata er altid den sidste data blok i den modtagende navigationdata. Den checksum der modtages, er beregnet fra AR.Dronen og samme funktionen, bruges i klienten til at beregne checksum af de modtagne navigationsdata.

Checksummen bruges til at efter beregne om de navigationsdata klienten har modtaget stemmer overens med de navigationsdata AR.Dronen har sendt. De bruges så klienten ved, om den har modtaget alle navigationsdata, eller om der er sket en fejl under kommunikationen. Er der sket en fejl under kommunikationen, skal programmet se bort fra de modtagne navigationsdata, da der er chance for fejl.

AR.Dronens SDK indeholder kildekoden til funktionen, der skal bruges til at beregne checksummen af de data klienten har modtaget.

Tabel 8 viser data strukturen af checksum blokken.

Tabel 8

Checksum block		
cks id	*size*	*cks data*
16-bit int	16-bit int	32-bit int

3.3 Xbox kontroller interface

Xbox kontrollerens interface til fjernstyringsystemet er over et WiFi netværk, der oprettes imellem den og Xbox Wireless receiver.

Figur 20 viser en Xbox kontroller hvorpå knappernes- og de analoge pindes funktionalitet i programmet er vist.

Figur 20 - Xbox kontroller interface

3.3.1 Analog 1

Ved brug af x aksen på analogpinden kan man styre AR.Dronen sidelæns, hvis man sætter pinden helt til højre vil AR.Dronen flyve med max hastighed til højre og tilsvarende til venstre. Y aksen styrer hastighed og retning af flyvning frem og tilbage.

3.3.2 Analog 2

Ved brug af x-aksen roterer AR.Dronen omkring sig selv, hvis pinden sættes til højre roterer AR.Dronen omkring sig selv til højre og ligeledes til venstre hvis pinden sættes i den retning. Y-aksen styrer ændring af flyvehøjde, hvis pinden sættes nedad vil AR.Dronen flyve nedad og omvendt opad hvis den sættes op.

3.3.3 Start

Start knappen bruges til enten at lette AR.Dronen. Hvis AR.Dronen er tændt og tilsluttet, men ikke flyver, vil start knappen starte AR.Dronen.

3.3.4 Back

Tryk på back knappen bruges til landing af dronen. AR.Dronen vil sænke sig ned til jorden og stoppe motorerne.

3.3.5 A

Ved tryk på A knappen ændres det video feed AR.Dronen sender.

3.3.6 Y

Ved tryk på Y knappen sættes emergency mode til og fra. Hvis AR.Dronen flyver og der trykkes på knappen slukkes alle motorerne og den styrter. Knappen bruges også til at få AR.Dronen ud af emergency mode igen, hvis den har ramt en væg eller lignende, og er gået i emergency mode af den grund.

4 Implementation view

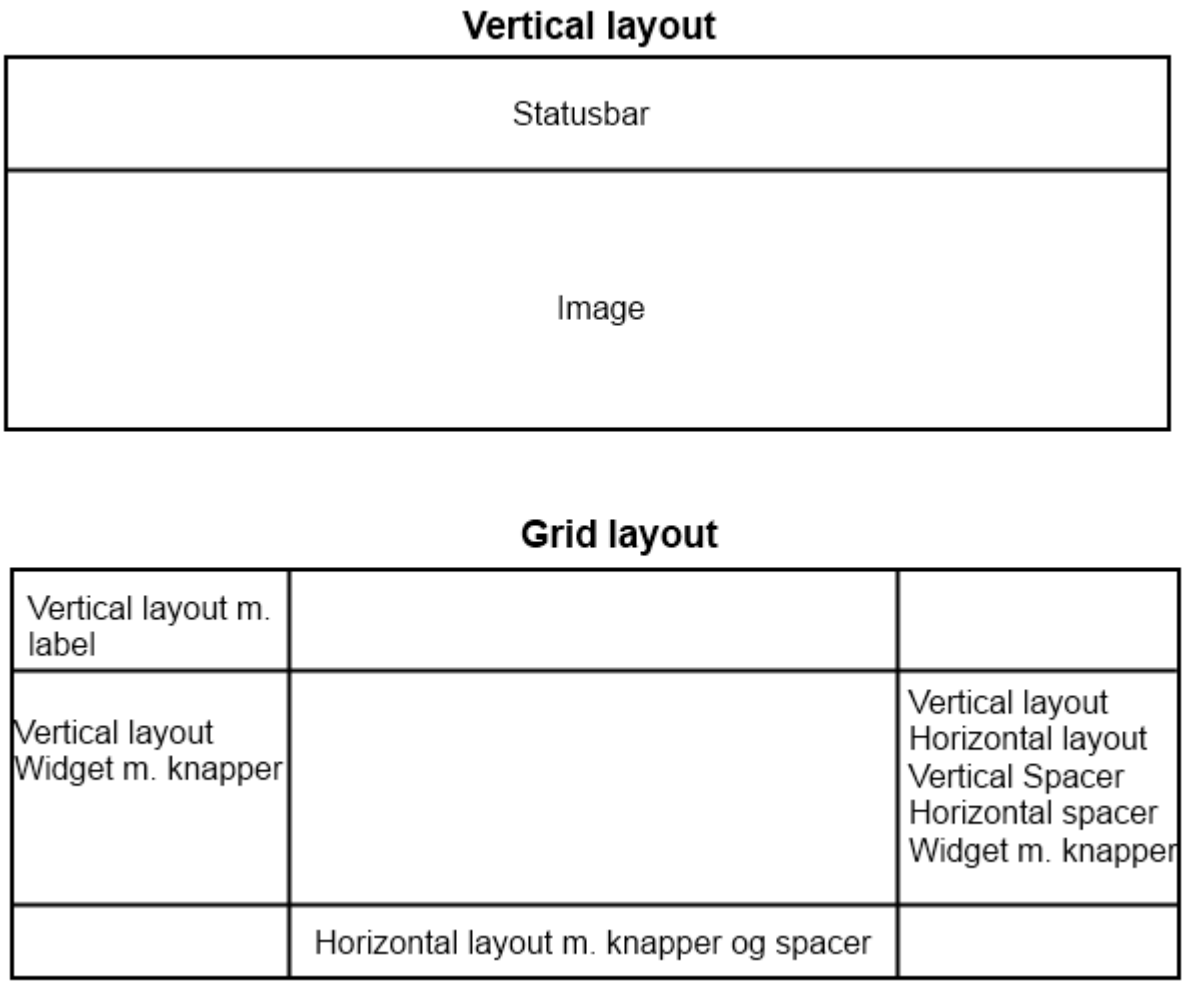
4.1 Software implementering

Den implementerede software kan ses i [Bilag 3] Source code. Kun elementer i koden der ikke er selvforklarende, eller kan forklares vha. kommentarer, er beskrevet i dette afsnit. Alt koden i [Bilag 3], er dokumenteret vha. kommentarer.

4.2 Grafiske bruger interface

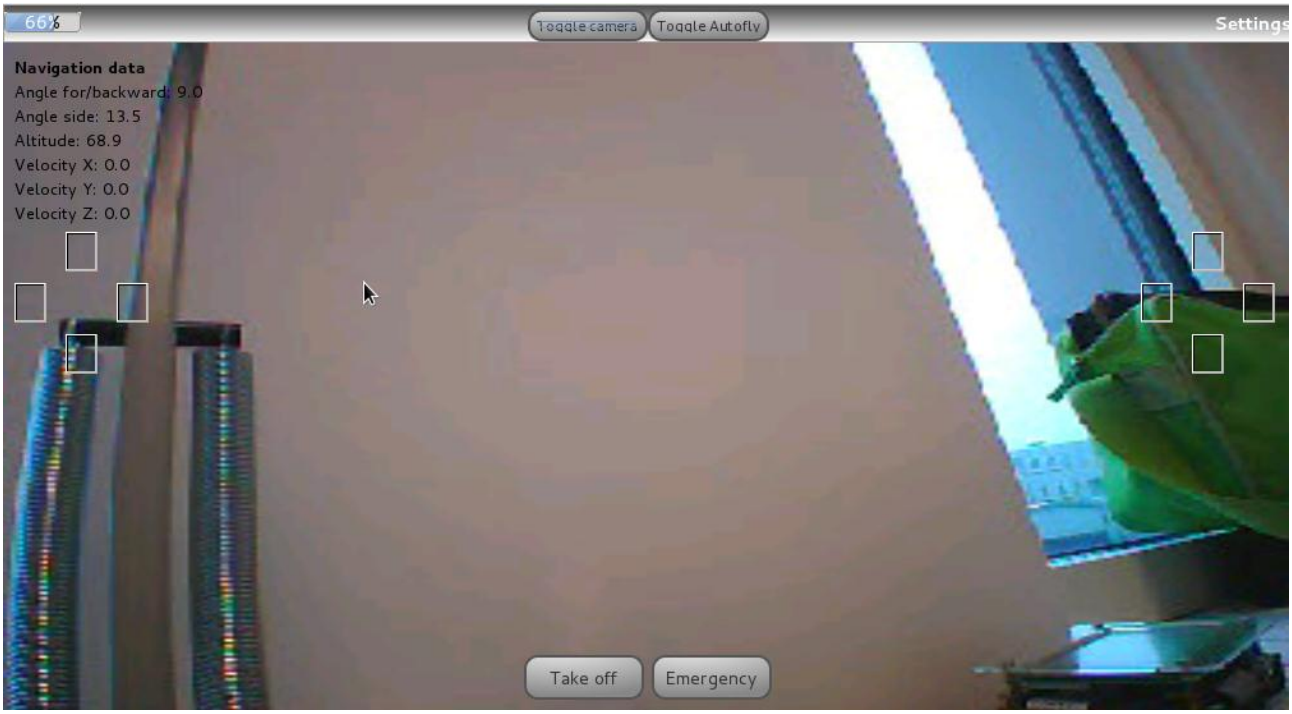
Under normale omstændigheder, ville det ikke være spor interessant at beskrive implementeringen af et bruger interface, der er lavet i en IDE med GUI designer. I dette projekt har der været et interessant aspekt i udviklingen af den grafiske bruger interface, da det hele ikke har været ”Drag and Drop” i en editor.

For at opnå det ønskede design, der blev opstillet som krav, krævede det flere layouts oven på hinanden (knapper til at styre AR.Dronen ovenpå video feedet). Da det i Qt Creators GUI designer ikke er muligt at lave layouts på layouts vha. ”Drag and Drop”, blev det nødvendigt at programmere sig ud af det. Figur 22 viser de to layouts skitseret, der skulle placeres oven på hinanden.



Figur 21 Opdeling af GUI design

Grid layout'et i Figur 21 er programmeret oven på "Image" i det vertikale layout. Grid layout, indeholder de forskellige knapper, tekst og komponenter der er at se oven på video feedet fra AR.Dronen, som kan ses på Figur 22.



Figur 22 GUI for fjernstyringssoftware

Det har været vigtigt at få de forskellige komponenter i det grafiske bruger interface, til at være så dynamisk som muligt, og hele tiden have relativ afstand til hinanden. Dette var vigtigt da det skal være muligt for brugeren at ændre på størrelsen af vinduet, samt ikke mindst for at få det til at passe ned på DevKit8000, der kører med en væsentlig lavere opløsning end en alm. desktop computer (480x272).

Koden til implementeringen af det grafisk bruger interface, kan findes i programmets source under ardrone.cpp i konstruktoren funktionen [Bilag 3].

4.3 Autonavigation

Når autonavigation er aktiv, omsættes billedet fra kameraet til et 5x5 grid, som bruges af en algoritme til styring af AR.Dronen.

Når programmet modtager et nyt billede fra AR.Dronen og autonavigation er aktiv, løber programmet alle pixels i billedet igennem og bestemmer om den enkelte pixel er mørk eller ikke er. Hvis et felt i grid'et indeholder nok mørke pixels vil det blive markeret med et 1-tal ellers et 0.

Når dette grid med 1'er og 0'er er fundet bruges algoritmen til at bestemme hvilke manøvre AR.Dronen skal foretage.

4.3.1 Algoritme

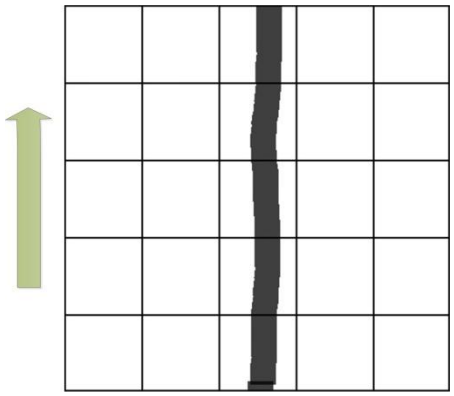
Formålet med algoritmen er hele tiden at holde AR.Dronen centreret over linjen, dvs. have linjen gående igennem felterne 1,3 og 5,3 i Figur 23.

1,1	1,2	1,3	1,4	1,5
2,1	2,2	2,3	2,4	2,5
3,1	3,2	3,3	3,4	3,5
4,1	4,2	4,3	4,4	4,5
5,1	5,2	5,3	5,4	5,5

Figur 23 - Grid inddeling

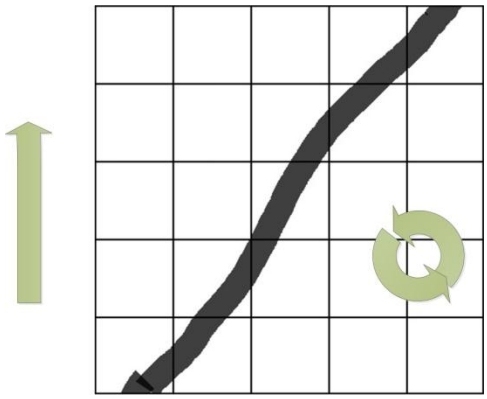
AR.Dronen vil flyve fremad hvis linjen går ud gennem toppen af billedet. Hvis AR.Dronen er forskudt i forhold til linjen flyver den sidelæns ind over den, og hvis AR.Dronen er skæv i forhold til linjen drejer den så linjen går lige under den igen. Hvis linjen går hele vejen igennem billedet fra bunden til toppen af billedet flyver AR.Dronen også fremad samtidigt med at den retter sig ind efter de andre betingelser.

Hvis AR.Dronen flyver fremad og billedet viser et billede, hvor linjen drejer ud af siden af billedet, vil AR.Dronen ikke længere flyve fremad, men i sted dreje på stedet. Dette gøres for igen at have linjen gående igennem billedet fra bunden til toppen af billedet.



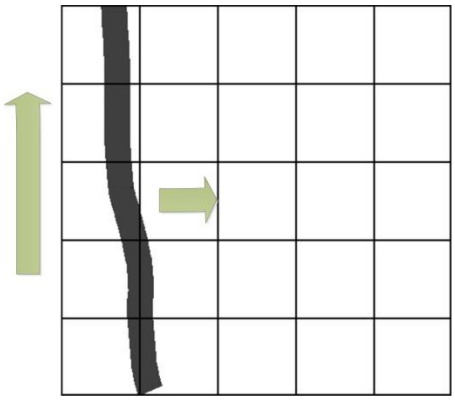
Figur 24 - Eksempel 1 autonavigation billede

Hvis AR.Dronen ser billedet som i Figur 24 skal den flyve fremad, da den peger lige på linjen og er centreret over den.



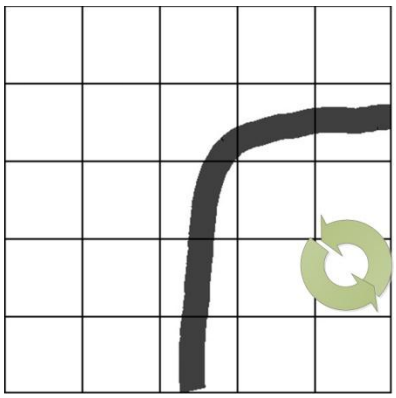
Figur 25 - Eksempel 2 autonavigation billede

Hvis AR.Dronen derimod ser et billede som på det ovenfor i Figur 25, skal den roterer sig selv da den ikke længere peger i samme retning som linjen. Den vil dog stadigt flyve fremad da linjen går ud gennem toppen af billedet.



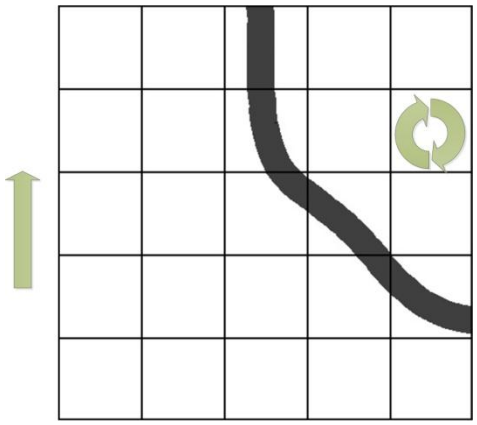
Figur 26 - Eksempel 3 autonavigation billede

På Figur 26 ses et eksempel hvor AR.Dronen er forskudt i forhold til linjen dette betyder at den skal flyve sidelæns ind over linjen igen mens den flyver fremad.



Figur 27 - Eksempel 4 autonavigation billede

Hvis billedet AR.Dronen ser, er som på Figur 27 skal den ikke længere flyve fremad da dette vil resultere i den kommer længere væk fra linjen. Den vil rotere på stedet, hvilket vil resultere i at den kommer tættere på målet at have linjen til at gå gennem 1,3 og 5,3.



Figur 28 - Eksempel 5 autonavigation billede

I Figur 28 vises hvad AR.Dronen skal gøre når den ser et billede hvor den går ud gennem midten i toppen som den skal men ikke kommer fra bunden af billedet men ind fra en af siderne. I dette tilfælde skal AR.Dronen stadig flyve fremad da linjen går ud gennem toppen af billedet, og den skal samtidigt rotere så linjen kommer til at være så tæt på centreret i billedet som muligt.

4.4 Opsætning af udviklingsmiljø

Tekst der forekommet med ***[Mads@mm ~]$*** foran henviser til terminal kommandoer der bliver afviklet på host computeren, og forekommer ***[root@DevKit8000 ~]$*** henvises der til terminal kommandoer på DevKit8000. Host beskriver den computer man udvikler på, og target beskriver det embeddede system, der i dette tilfælde er DevKit8000.

Til at udvikle på projektet bruges Qt creator der er en IDE til Qt frameworket og indeholder både compiler, source editor, GUI designer, debugger mm..

4.4.1 Installation af Qt Creator

Qt Creator hentes på Qt's hjemmeside:

http://qt-project.org/downloads#qt-creator

Og installeres derefter. Brugervejledning til Qt Creator kan findes på Qt's hjemmeside:

http://doc.qt.digia.com/qtcreator-snapshot/index.html

4.4.2 Installation af ARM krydskompiler

For at kompilere styresystemet, programmet og biblioteker til et embedded board der er baseret på en processor med ARM arkitektur, er det nødvendigt at have en krydskompiler, der kan kompilere til ARM processorer, da disse processorer ikke forstår biblioteker m.m. der er kompileret med en alm. x86 compiler.

Da Devkit8000 er et gammelt board, kan der ikke bruges den nyeste ARM compiler, da denne ikke er kompatibel med Devkit8000's processor. Den nyeste kompatible compiler der blev fundet frem til var fra 2009 og er vedlagt som [Bilag 6]. Denne kompiler installeres ved at eksekvere .bin filen der vedlagt som [Bilag 6]:

[Mads@mm Angstrom_distro]$ ./arm-2009q1-203-arm-none-linux-gnueabi.bin

Installationen opdaterer selv environment variablerne for Linux, så den kender til den nye kompiler. Det kan nu testes om kompileren virker ved at skrive følgende:

[Mads@mm ~]$ arm-none-linux-gnueabi-gcc

Er kompileren installeret korrekt, giver dette følgende output:

arm-none-linux-gnueabi-gcc: no input files

4.4.3 Kompilering af touchscreen library

For at installere touchskærmens touch funktionalitet korrekt på DevKit8000, er det nødvendigt at kompilere librariet 'tslib' til ARM processoren og installere den på boardets filsystem. Først skal tslib hentes:

[Mads@mm ~]$ git clone https://github.com/kergoth/tslib.git tslib

Skift til mappen, hvori tslib er downloadet til. Første trin er at lave en autogeneration af konfigureringsfilen:

[Mads@mm ~]$ ./autogen-clean.sh
[Mads@mm ~]$ ./autogen.sh

Derefter skal tslib konfigureres:

[Mads@mm ~]$ ./configure --host=arm-none-linux-gnueabi --prefix=/home/tslib --enable-shared=yes -- enable-static=yes

Tslib konfigureres til både at have statisk og shared libraries, da Qt ellers ikke kan kompileres med tslib. Derudover sættes host, til at være den installerede ARM kompiler, så den benyttes til at kompilere tslib. Prefix, er den sti, hvori der ønskes at tslib bliver installeret, denne kan frit vælges, dog skal der huskes på denne sti, da den skal bruges i næste afsnit. Før tslib kan kompileres og installeres, skal filen *config.h* ændres.
Udkommenter linjen med:

#define malloc rpl_malloc

Udkommenteres denne linje ikke, vil der meldes fejl under kompileren. Tslib kan derefter kompileres:

[Mads@mm ~]$ make
[Mads@mm ~]$ make install

Hvis alt gik som det skulle, burde man kunne se headers og libraries i */home/tslib/* eller hvor der blev valgt at installere det.

4.4.4 Kompilering af Qt everywhere (ARM) med touchskærms library

Programmet er som nævnt tidligere skrevet i C++ med Qt frameworket. Qt er ikke direkte kompatibel med embeddede processorer, det skal installeres korrekt for at et Qt program kan køre på embeddede systemer. Når en Qt applikation kompileres fra host'en husker den på hvorfra den hentede de libraries den har benyttet, og derfor er det vigtigt at installationen af Qt på target har præcis samme destination som på hosten. Installeres Qt kompileren for ARM processorer på host'en f.eks i /opt/qt-arm/ skal den have samme destination på target, så den også placeres i /opt/qt-arm/ på det embeddede board.

4.4.4.1 Kompilering

Der skal kompileres en Qt kompiler med en ARM kompiler, så Qt bliver kompatibel med ARM processorer. Dette gøres ved at kompilere Qt-everywhere, der er en source udgave af Qt der kan kompileres til mange forskellige systemer, med den i forvejen installerede arm-none-linux-gnueabi- kompiler. Qt-everywhere er vedlagt [Bilag 12]. Første step er at pakke Qt-everywhere ud:

[Mads@mm ~]$ tar -xvzf qt-everywhere-opensource-src-4.6.2.tar.gz

Skift mappe til der hvor Qt-everywhere er pakket ud.
Opret en ny mappe i mkspecs/qws/ kaldet linux-DEVKIT8000-g++.
Kopier filerne fra mappen mkspecs/qws/linux-arm-g++ over i den nye mappe der lige er oprettet (qmake.conf, qplatformdefs.h). Disse filer indeholder kompiler informationer og libraries. Qmake.conf skal nu ændres til følgende:

```
include(../../common/linux.conf)
include(../../common/gcc-base-unix.conf)
include(../../common/g++-unix.conf)
include(../../common/qws.conf)

QMAKE_INCDIR += /home/tslib/include/
QMAKE_LIBDIR += /home/tslib/lib/
QMAKE_LIBS   += -lts

# modifications to g++.conf
QMAKE_CC                = arm-none-linux-gnueabi-gcc
QMAKE_CXX               = arm-none-linux-gnueabi-g++
QMAKE_LINK              = arm-none-linux-gnueabi-g++
QMAKE_LINK_SHLIB        = arm-none-linux-gnueabi-g++

# modifications to linux.conf
QMAKE_AR                = arm-none-linux-gnueabi-ar cqs
QMAKE_OBJCOPY           = arm-none-linux-gnueabi-objcopy
QMAKE_STRIP             = arm-none-linux-gnueabi-strip

load(qt_config)
```

Bemærk, at /home/tslib skal erstattes med den sti, hvori der blev valgt at installere tslib. Nu er der opsat en konfigurationsfil der både indeholder tslib, og den rigtige kompiler. Næste step er at konfigurere Qt til at blive kompileret med disse indstillinger, samt en masse andre konfigurationer:

[Mads@mm ~]$ ./configure -opensource -confirm-license -prefix /opt/qt-arm -no-qt3support -embedded arm -little-endian -xplatform qws/linux-DEVKIT8000-g++ -qtlibinfix E -qt-mouse-tslib -qt-mouse-linuxinput -qt-kbd-linuxinput

OBS: Benyttes en 64-bit host til at kompilere Qt, skal der tilføjes en yderlig konfiguration til "configure":

-platform qws/linux-x86_64-g++

Derefter er Qt konfigureret til at være kompatibel med ARM processorer og touchskærm, og klar til at blive kompileret og installeret på host computeren. Dette gøres ved:

[Mads@mm ~]$ gmake
[Mads@mm ~]$ gmake install

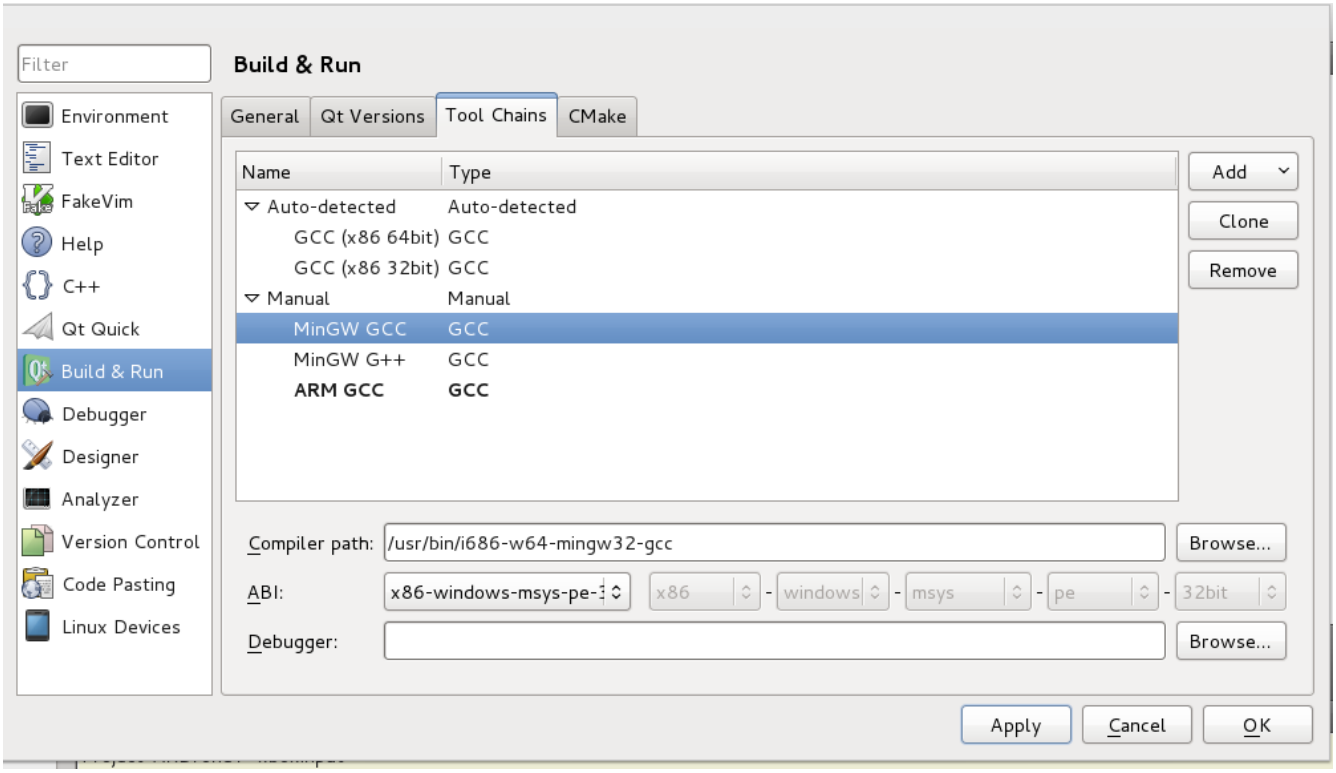
Dette kan tage op til mange timer, og når det er færdig med at installere og kompilere er Qt installeret under den mappe der blev valgt som prefix (***/opt/qt-arm*** *i eksemplet)*.

4.4.5 Opsætning af Qt Creator med Qt everywhere (ARM)

Efter Qt everywhere er blevet krydskompileret med ARM gcc kompileren, findes der en udgave af 'qmake' beregnet til platforme der har processor med ARM arkitektur. Denne 'qmake' fil er at finde under binaries mappen i den prefix den blev installeret til (**/opt/qt-arm/***bin*) i dette tilfælde.

4.4.5.1 Opsætning af toolchain og Qt versions

Qt Creator skal nu sættes op til at benytte denne qmake hvis man vælger at kompilere programmet til en processor med ARM arkitektur. Dette gøres i Qt Creator ved at gå ind i menuen Tools -> Options. I Options menuen, skift til Build & Run menuen, som vist på Figur 29

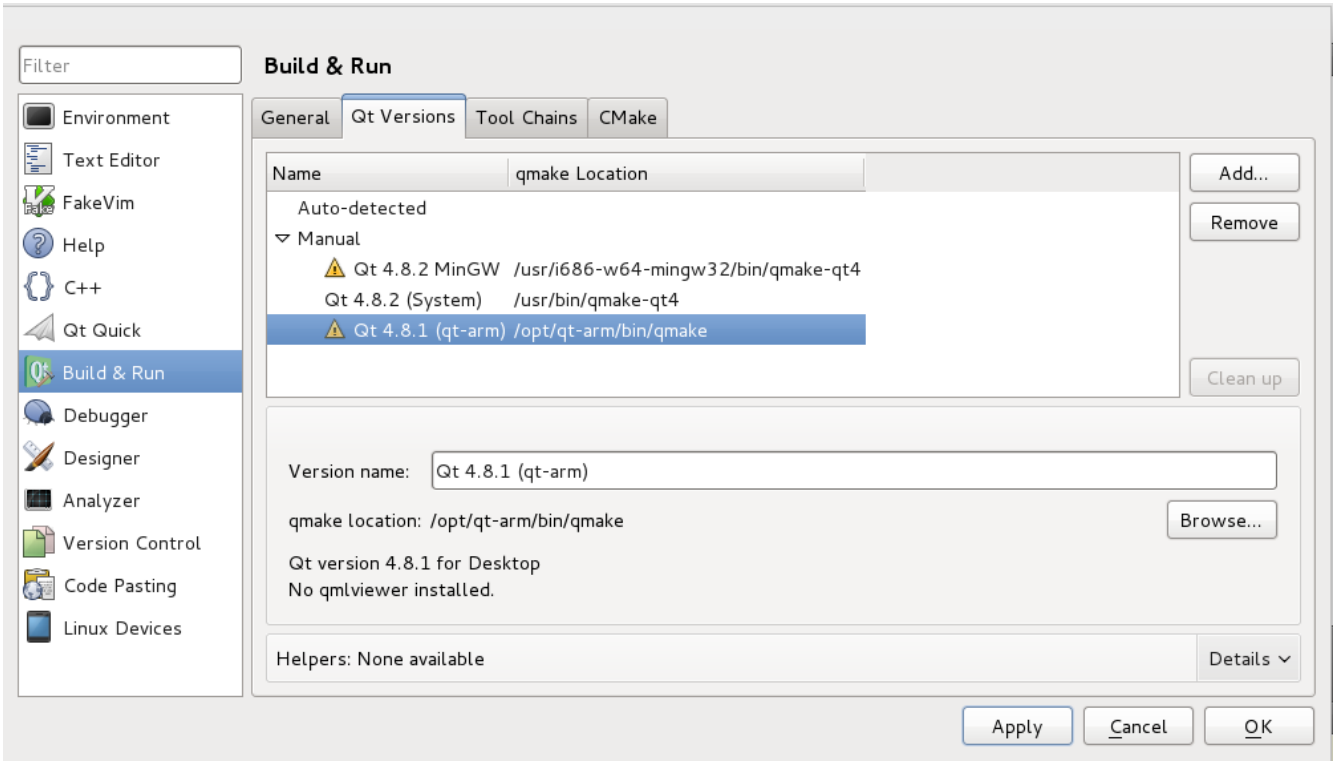


Figur 29 Build & Run indstillinger i Qt Creator

Tilføj en ny gcc kompiler ved at trykke Add -> GCC. En ny toolchain vises nu kaldet GCC i list viewet. Omdøb denne toolchain gcc kompiler til ARM GCC som i Figur 29

Compiler path skal ændres til at være ”arm-none-linux-gnueabi-gcc” eller den direkte sti hvor ARM gcc kompileren er installeret. Tryk Apply – en toolchain for ARM kompileren er nu tilføjet til Qt Creator. Nu skal qmake tilføjes.

Gå til fanebladet Qt Versions som vist på Figur 30.



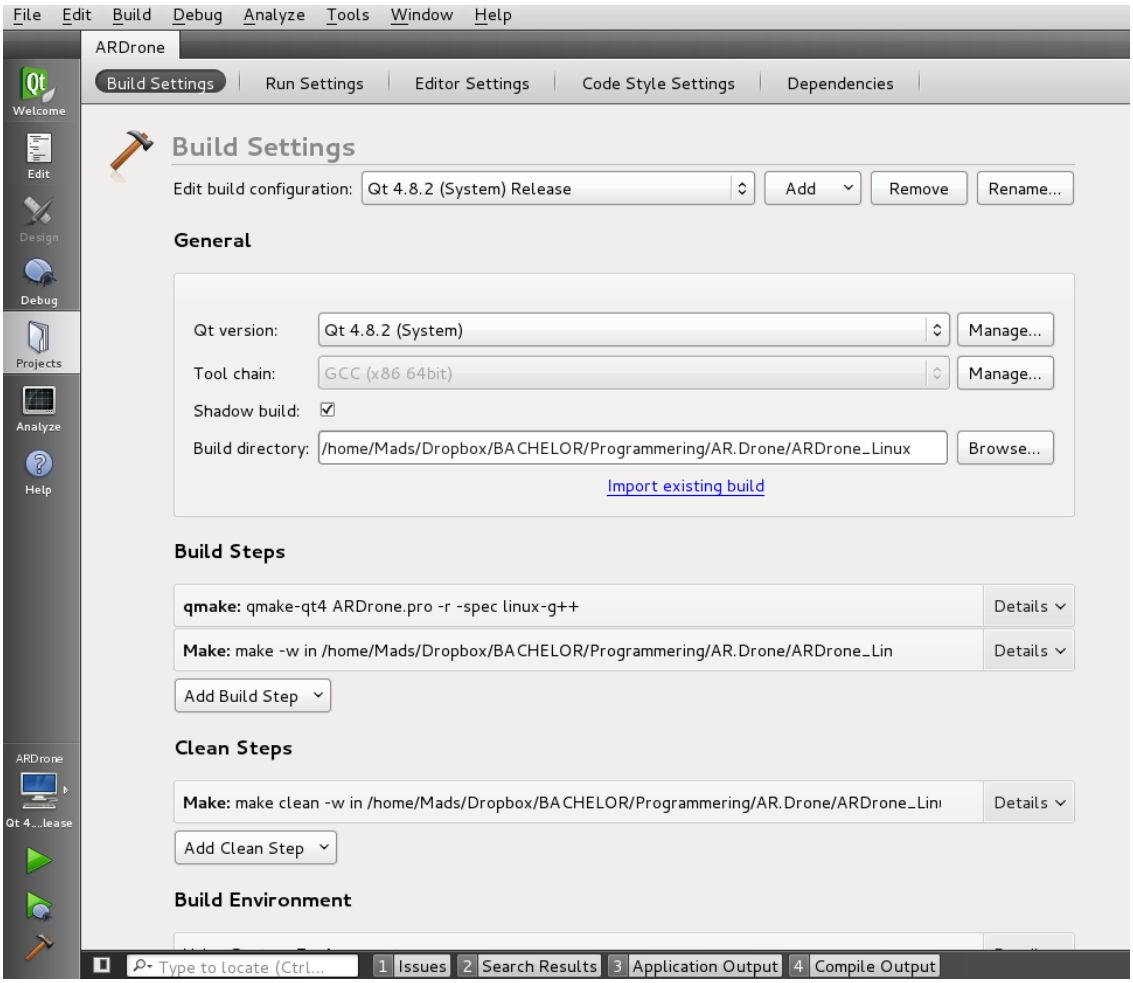
Figur 30 Qt version i Qt Creator

Tryk nu på Add…, og find den qmake der blev kompileret til ARM processorer. Tidligere i afsnittet blev der beskrevet hvor denne qmake var at finde. Vælg qmake filen, tryk Apply og derefter OK for at lukke indstillingsmenuen.

4.4.5.2 Opsætning af projekt

De nødvendige toolchains og Qt versioner (qmake) er nu tilføjet til Qt Creator. Næste step er at tilføje disse toolchains og qmake til projektet for at projektet kan blive kompileret til ARM arkitekturer.

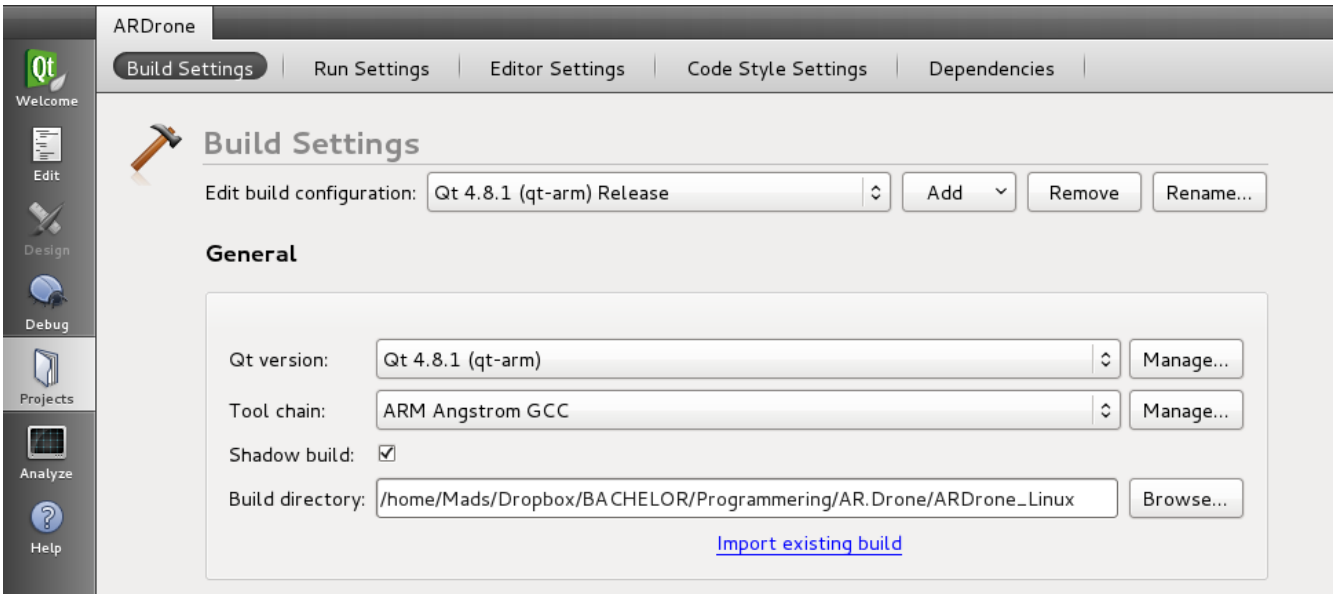
Dette gøres ved at åbne projektet, og i den venstre fane menu vælges Projects for at indstille på projektets build settings. Dette er vist på Figur 31.



Figur 31 Build settings for projektet

Inde i Build Settings vælges Add, i dropdown listen vælges den tilføjede ARM version af Qt.
Der er nu tilføjet en debug og release udgave af Qt versionen for ARM arkitekturer.

For nu at kompilere med Qt versionen for ARM arkitekturer, vælges denne Qt version i kompiler menuen som vist i Figur 32.



Figur 32 Den nye version af Qt kompiler for ARM arkitekturer er valgt

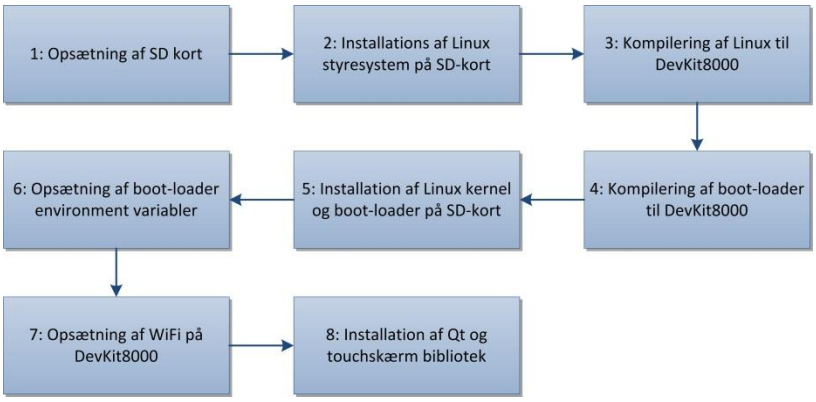
Ønskes der at ændre sti til hvor det endelig program bygges i, kan dette ændres i Build directory.

4.5 Opsætning af DevKit8000

I dette afsnit af implementeringen beskrives hvordan programmet, DevKit8000, WiFi'en, Xbox kontrollere, Linux kernen + styresystem implementeres. I dette afsnit tages der udgangspunkt i at et Linux styresystemet benyttes til implementeringen, da der skal krydskompileres og kompileres Linux relaterede biblioteker, styresystemer mm..

4.5.1 Overblik

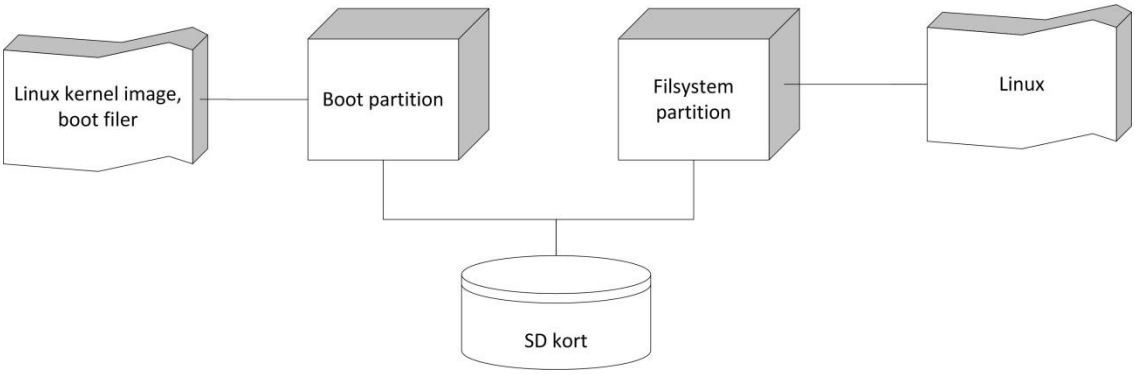
Figur 33 viser trin for trin hvordan DevKit8000 skal opsættes, for at den opfylder de nødvendige krav til systemet. De forskellige trin bliver beskrevet i dette afsnit.



Figur 33 - Opsætnings trin af DevKit8000

DevKit8000 har en ARM arkitektur baseret processor, kaldet OMAP3530, der bruger et anderledes instruktionsset end en almindelige desktop computerer der bruger x86 arkitekturer. Derfor er det nødvendigt at have en kompiler installeret, der kan oversætte programmet til et instruktionsset, som ARM baserede processorer bruger. Hvordan sådan kompiler installeres og bruges til at kompilere programmet, Qt frameworket, libraries m.m., vil blive beskrevet i dette afsnit.

Da DevKit8000 er et evaluation-board og det er beregnet til at blive eksperimenteret med, kommer det derfor heller ikke med nogen indbygget hukommelse. Derfor har DevKit8000 en SD-kort indgang, hvorpå der kan installeres et styresystem. For der kan installeres et styresystem på SD-kortet og det kan fungere som "harddisk" skal dette formateres og opsættes korrekt. Figur 34 viser opbygningen af SD-kortet som det skal være for at fungere som harddisk og boot enhed for DevKit8000.



Figur 34 - SD kortets opbygning

For at et wireless netværks adapter kan benyttes på DevKit8000, kræver det først og fremmest at Linux understøtter den firmware som netværks adapteren benytter, herefter kræves det er Linux kernelen har installeret disse kernel drivere, der kan få Linux til at snakke med netværks adapteren. Når disse kernel drivere til netværksadapteren er installeret, skal Linux' netværks interface sættes op til at benytte netværksadapteren og have at vide hvilket netværk den skal forbindes til.

Proceduren for hvordan alt det ovenstående kompileres, installeres og konfigureres korrekt vil blive gennemgået i dette afsnit.

4.5.2 SD kort formatering

DevKit8000 afvikler sit styresystem og filsystem igennem et SD kort, og det er derfor nødvendigt at dele SD kortet op i to partitioner, et til boot hvor styresystemets image placeres, og et til rootfs der er ens filsystem.

Det anbefales at have et SD kort på 2-4GB så der er plads nok til filsystemet.

Boot partitionen er en FAT partition, og filsystemets partition er et "extended filesystem" (ext3 eller ext4). For at dele SD kortet op i disse partitioner benyttes et shell script der er vedlagt som [Bilag 13]. Dette ekskveres i konsollen med følgende kommando:

[Mads@mm ~]$ ./omap3-mkcard.sh /dev/sdb

Hvor /dev/sdb er stien til SD kortets device file. Er der tvivl om hvad stien til SD kortets device file er, kan følgende kommando bruges til at liste alle computeres filsystemer hvori SD kortet vil indgå:

[Mads@mm ~]$ df –h

Når kortet er formateret, burde der på kortet kunne ses to partitioner, henholdsvis ”boot” og ”Angstrom” hvor ”boot” er bootpartitionen (FAT) og ”Angstrom” er filsystemets partition (ext3).
Det næste step er at få lagt et styresystem over på SD kortet.

4.5.3 Installation af filsystem på SD-kort

Som filsystem og Linux distrubition benyttes Ängstrom's udgave af Linux. Linux distrubition der blev benyttet i projektet er vedlagt som [Bilag 7].

Ängstrom installeres ved følgende kommando:

[Mads@mm ~]$ sudo tar -xvz -C /run/media/Angstrom/ -f devkit8000-rootfs.tar.gz

/run/media/Angstrom/ kan erstattes af stien til filsystemet på SD kortet, hvis ikke denne sti stemmer overens. Kommandoen udpakker indholdet i .tar.gz filen til filsystemets partition på SD kortet. Når dette er gjort, er filsystemet installeret.

4.5.4 Kompilering af Linux til Devkit8000

Da det ikke er muligt at bruge den Linux kernel (3.6.10) til embeddede processorer, som Devkit8000 består af, skal der benyttes en speciel udgave af Linux der er optimeret og tilpasset OMAP processorer. Der findes forskellige udgaver af disse Linux kerneler rundt om på nettet alt afhængig af hvilket embedded board man ønsker at bruge, og heldigvis findes der også en udgave til Devkit8000. Ofte kræver et embedded board deres egne patches af Linux kernelen for at virke, som oftest er tilgængelige på fabrikantens hjemmeside eller som bruger vedligeholdte git repositories. DevKit8000 havde en bruger vedligeholdt udgave af Linux kernelen på gitorious, der er en git repository.

Udgaven af Linux til Devkit8000 hentes:

[Mads@mm ~]$ git clone git://gitorious.org/devkit8000/linux-omap-devkit8000.git linux-omap-devkit8000

Når filerne er hentet, burde der nu være en mappe kaldet ”linux-omap-devkit8000” hvori filerne til at kompilere Linux er placeret. Skift til denne mappe i konsollen. Makefilen skal nu opsættes så den passer til Devkit8000 og har de moduler, m.m. der ønskes installeret. Dette gøres ved kommandoen:

[Mads@mm ~]$ make -j5 ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- xconfig

Der åbnes nu en menu hvori det er muligt at til og fravælge en masse moduler m.m.. Vælg åben fil, og vælg derefter den konfigurationsfil der er vedlagt som [Bilag 10]. Denne konfigurationsfil er konfigureret til

Devkit8000's LCD skærm, Wifi, Joystick input og en masse andre nødvendige moduler. Efter det er blevet konfigureret til at blive kompileret med moduler så WiFi, Joystick og LCD skærmen virker skal Linux nu kompileres. Dette gøres ved:

[Mads@mm ~]$ make -j5 ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- uImage

Når kompileren af det nye Linux image er færdig, vil der i stien *~/linux-omap-devkit8000/arch/arm/boot/* nu være en fil kaldet uImage, der er det image der skal bruges som Linux kernel på SD kortet.

4.5.4.1 Kompilering og installering af kernel moduler på filsystemet

For at de tilvalgte moduler der i forrige afsnit blev tilvalgt, er funktionelle i kernelen skal de kompileres og installeres på filsystemet. Først kompileres de ved følgende kommando:

[Mads@mm ~]$ make -j5 ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- modules

Derefter installers modulerne på filsystemet ved:

[Mads@mm ~]$ sudo make -j5 ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- INSTALL_MOD_PATH=/run/media/Angstrom modules_install

Hvor */run/media/Angstrom* erstattes af stien hvorpå filsystemets partition på SD kortet er mountet. Derefter er kernel modulerne installeret på filsystemets partition på SD kortet under */lib/modules/,* så styresystemet har mulighed for at loade og unloade Linux kerneler.

4.5.5 Kompilering og installering af boot loader

For at Devkit8000 kan starte op fra SD kortet, skal der installeres en bootloader på boot partitionen. På boardets boot-rom findes første etape af bootloaderen, der sørger for at boote fra SD kortet og eksekvere anden etape af bootloaderen der kaldes MLO. MLO er en fil læser boot partitionen på SD kortet, og eksekverer den tredje etape af bootloaderen der hedder u-boot. U-boot filen kan opsættes med forskellige variabler og kommandoer til opstart af kernelen, disse variabler vil blive beskrevet senere da der skal bruges nogle bestemte for at initialisere kernelen og DevKit8000 korrekt for at kunne starte op. U-boot eksekverer derefter uImage, der er Linux kernelens image, og styresystemet er derefter startet op.

Første stadie af bootloaderen, ligger på Devkit8000's flash hukommelse og skal ikke ændres, hvorimod MLO og u-boot skal kompileres og installeres på SD kortets boot partition.

U-boot og MLO filerne og source er vedlagt som [Bilag 8]. Først konfigureres u-boot til DevKit8000:

[Mads@mm ~]$ make CROSS_COMPILE=arm-none-linux-gnueabi- devkit8000_config

Efter u-boot er konfigureret kan det nu kompileres:

[Mads@mm ~]$ make CROSS_COMPILE=arm-none-linux-gnueabi- devkit8000

Derefter er u-boot.img og MLO at finde I hovedstien af u-boot.

4.5.6 Installation af kernel, u-boot og MLO

Efter både bootfilerne u-boot og MLO samt Linux kernel image uImage er kompileret skal det kopieres over på SD kortets boot partition. Dette gøre ved at eksekvere følgende kommandoer:

```
[Mads@mm ~]$ cp ~/u-boot/MLO /media/boot
[Mads@mm ~]$ cp ~/u-boot/u-boot.img /media/boot
[Mads@mm ~]$  cp ~/linux-omap-devkit8000/arch/arm/boot/uImage /media/boot
```

Stemmer stierne på filerne, samt på SD kortet ikke overens med de stier hvori de er blevet kompileret til, ændres disse så de matcher.

4.5.7 Opsætning af boot variabler

Når MLO, u-boot.img og uImage er kopieret over på SD kortets boot partition er det nu muligt at starte DevKit8000 via. SD kortet ved at holde boot knappen på DevKit8000 nede alt imens der tændes for strømmen til boardet. For at se output fra DevKit8000 tilsluttes seriel kabel mellem computer og DevKit8000, og igennem en terminal er det nu muligt at se output fra boardet med baudrate på 115200. Til Linux kan CuteCom terminal anbefales.

Når der i terminalen spørges om man vil stoppe autoboot af DevKit8000 trykkes på en vilkårlig tast, for at tilgå u-boot prompten hvori boot environment variabler kan indstilles.
For at DevKit8000 benytter den rigtige driver til touchskærmen, opløsning og filsystem skal 'bootargs' sættes med følgende kommando i u-boot prompten:

```
setenv bootargs 'console=ttyO2,115200n8 vram=12M omapdss.def_disp=lcd
omapfb.mode=lcd:480×272 root=/dev/mmcblk0p2 rw rootwait earlyprintk'
```

console konfigurerer terminaloutputtet fra DevKit8000, omapdss.def_disp og omapfb.mode konfigurerer touchskærmen til at være standard video output med opløsningen 480x272, *root* angiver hvor filsystemet er placeret og peger på SD kortets (MMC) anden partition, *earlyprintk* laver kernel debug output til terminalen.

Når *bootargs* er konfigureret, skal *bootcmd* konfigureres, så u-boot ved hvilken kommando der skal bruges til at boote Linux kernelen. Dette gøres ved følgende kommando:

```
setenv bootcmd 'mmc init; mmc rescan 0; fatload mmc 0 0×80300000 uImage; bootm
0×80300000'
```

Når bade *bootcmd* og *bootargs* er konfiguret skrives de til flash ved:

```
saveenv
```

Når de er gemt på kortets flash hukommelse, læser u-boot dem ved hver opstart og sender bootargs videre til kernelen som parameter. U-boot benytter bootcmd til at loade kernelen ned på boardets hukommelse og derefter booter fra den kernel. Disse boot kommandoer der er sat i bootcmd, eksekveres hver gang DevKit8000 startes op.

OBS: Når ikke Linux kernelen mm., er flashet til DevKit8000 NAND hukommelse, skal boot knappen på boardet holdes nede hver gang den startes, så der bootes fra SD kortet og ikke fra NAND hukommelsen.

4.5.8 Opsætning af WiFi forbindelse til AR.Dronen

For at oprette forbindelse fra DevKit8000 til AR.Dronen over WiFi, er det nødvendigt at kende AR.Dronens WiFi SSID. Dette gøres ved følgende kommando:

[root@DevKit8000 ~]$ iwlist wlan0 scanning

Hvorefter wlan0 interfacet der i dette tilfælde er WiFi USB donglen, skriver alle de WiFi netværk ud den kan finde. Find AR.Dronens WiFi netværk blandt disse – den hedder ardrone_123456, hvor 123456 er tilfældige numre.
For at der bliver oprettet forbindelse til AR.Dronens netværk, skal */etc/network/interfaces* på SD-kortets filsystempartition ændres til:

```
auto lo
iface lo inet loopback

iface wlan0 inet dhcp
        wireless_mode managed
        wireless_essid ardrone_123456
```

Dette konfigurer wlan0 interfacet, som er USB WiFi donglen, til at oprette forbindelse til AR.Dronen når netværket aktiveres. Wlan0 interfacet aktiveres ved:

ifup wlan0

4.5.9 Installation af Qt og touchskærms library

Som nævnt tidligere i afsnit 4.4.3, skal de samme libraries og kompiler der blev brugt til at kompilere et program på hosten bruges på target med samme stier mm..

Derfor skal mappen hvori Qt til ARM processorer blev installeret på hosten, kopieres over på SD kortets filsystem partition, så der bliver en mappe magen til den prefix der blev valgt tidligere. Dette gøres ved:

[Mads@mm ~]$ sudo mkdir /run/media/Angstrom/qt-arm
[Mads@mm ~]$ sudo cp –R /opt/qt-arm/* /run/media/Angstrom/qt-arm

Hvor mapperne erstattes med hvor SD kortets filsystem partition er mountet, og den prefix der blev valgt at installere Qt til. Derefter skal tslib kopieres til SD kortets filsystem partition. Først skal filen i */home/tslib/tslib/etc/ts.conf* ændres så linjen med

#module_raw input

Bliver udkommenteret så den i stedet bliver:

module_raw input

Derefter kan filerne kopieres til SD kortets filsystem partition.

[Mads@mm ~]$ sudo cp –R /home/tslib/* /run/media/Angstrom/usr/

Når filerne er kopieret til SD kortet, skal environment variablerne for det Linux der er installeret på SD-kortet sættes op, så det er muligt at benytte sig af disse Qt libraries og kompiler. Dette gøres ved at tilføje følgende til */run/media/Angstrom/etc/profile*:

```
export PATH=$PATH:/opt/qt-arm/bin:/opt/qt-arm/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/lib:/usr/lib:/opt/qt-arm/lib
export QTDIR=/opt/qt-arm
export TSLIB_TSDEVICE=/dev/input/touchscreen0
export TSLIB_CALIBFILE=/etc/pointercal
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_PLUGINDIR=/usr/lib/ts
export TSLIB_CONFFILE=/usr/etc/ts.conf
export QWS_MOUSE_PROTO=Tslib:/dev/input/touchscreen0
ifup wlan0
```

Da profile filen kaldes under opstart af Linux, skal DevKit8000 nu genstartes, så ændringerne træder i kraft. *Ifup wlan0* tilføjes da wireless ikke er aktiveret fra start når DevKit8000 startes. Derefter er det muligt at kalibrere touchskærmen ved at skrive følgende i konsollen:

[root@DevKit8000 ~]$ ts_calibrate

Derefter startes et kalibrerings program, og touchskærmen kalibreres manuelt ved at følge guiden på skærmen.

4.6 Software implementering

4.6.1 Kompilering

Dette afsnit vil beskrive hvordan programmet kompileres til både target og host.

4.6.1.1 ARM arkitektur

For at fjernstyringssoftwaren skal kunne eksekveres på DevKit8000, kræves det at dette også kompileres til en ARM processor. Da der lige er blevet kompileret en Qt kompiler til dette formål, benyttes denne til at kompilere fjernstyringssoftwaren. Dette gøres på følgende måde:

[Mads@mm ~]$ /opt/qt-arm/bin/qmake /path/to/projectfile/ARDrone.pro -o /path/to/install/at/ARM_build

Hvor stierne ændres til der hvor projektets fjernstyringssoftware projekt ligger, og til der hvor programmet ønskes installeret.
I terminalen, skift til den mappe der blev valgt at Makefilen skulle generes i *(/path/to/install/at/ARM_build)*

[Mads@mm ~]$ cd /path/to/install/at/ARM_build

Og kør derefter Makefilen med følgende commando:

[Mads@mm ~]$ make

Derefter er fjernstyringssoftwaren kompileret til ARM arkitekturen, og er nu klar til at blive lagt over på SD kortets filsystem partition.

Er Qt Creator opsat som beskrevet i 4.4 Opsætning af udviklingsmiljø, kan dette gøres i Qt Creator i stedet for den manuelle metode.

4.6.1.2 x86 arkitektur

For at kompilere til alm. processorer x86 arkitekturer, benyttes hostens alm. qmake til dette. Princippet og fremgangsmåden er identisk med for ARM arkitekture, som beskrevet foroven. Dog skal qmake stien ændres til at bruge qmake for x86:

[Mads@mm ~]$ qmake /path/to/projectfile/ARDrone.pro -o /path/to/install/at/x86_build

Og derefter skal Makefilen eksekveres som i forrige afsnit.

Dette kan også gøres automatisk i Qt Creator ved at vælge (System) Qt versionen, der indeholder en kompiler for alm. x86 processorer.

4.6.2 Afvikling

Dette afsnit beskriver hvordan programmet afvikles på host og target.

4.6.2.1 DevKit8000

Når programmet er lagt over på DevKit8000, og man er i samme sti som programmet er placeret, kan det eksekveres med følgende kommando:

[root@DevKit8000 ~]$ ./ARDrone –qws

Da fjernstyringssoftwaren er en GUI applikation, er *–qws* vigtig at få med som parameter da man fortæller Qt at det skal benytte sin egen vindue server, der kan afvikle GUI programmer på Linux' framebuffer.

4.6.2.2 Linux desktop

Skal programmes afvikles på en alm. Linux desktop computer kan det afvikles ved at man i samme sti som hvor programmet er placeres eksekverer følgende kommando:

[Mads@mm ~]$ ./ARDrone