Aarhus Universitet

Datalogisk Institut

Master's Thesis

Semi-autonom indendørs navigation for luftbåren robot

Morten Daugaard, 20051715
Thomas Thyregod, 20051688

Vejleder:
Ole Caprani

1. august 2012

Abstract

This thesis investigates the possibility of making a quadrokopter, AR.Drone, fly semi autonomously in an indoor environment. A software platform is implemented, which makes data from the built-in sensor devices on the AR.Drone available and maintains the control connection to the AR.Drone. The platform implemented in this thesis is used to facilitate the answering of a series of questions through experimentation. The performed experiments regards: How can the sensor apparatus aboard the AR.Drone be exploited to derive more abstract information about AR.Drones immediate environment (as in information not already accessible). How the derived information can be used by the AR.Drone to navigate by (experiments have shown that the AR.Drone is able to follow a planned route in an open space using visual markers on the floor).

The techniques tested in the experiments show tendencies of usefull solutions to infer information about the AR.Drones local area.

Resumé

Dette speciale handler om at klarlægge mulighederne, for at gøre en quadrokopter, AR.Drone, delvis selvflyvende i et indendørsmiljø. Mere specifikt er der implementeret en softwareplatform, som gør data fra AR.Dronens indbyggede sensorapparat tilgængelig og som har ansvaret for at opretholde kontrolforbindelsen til AR.Dronen. Den implementerede platformen er i specialet brugt til at facilitere besvarelsen af en række spørgsmål gennem eksperimentering. De udførte eksperimenter omhandler: Hvorledes sensorapparatet ombord på AR.Dronen kan udnyttes, til at udlede mere abstrakt information om AR.Dronens umiddelbare nærmiljø, som der ikke i forvejen er adgang til. Hvordan den udledte information kan anvendes af AR.Dronen til at navigere efter (eksperimenter har bl.a. vist at AR.Dronen er istand til at følge en planlagt rute i et åbent rum ved hjælp af visuelle markører på gulvet).

De testede teknikker i eksperimenterne viser lovende brugbare løsninger til udledning af information om AR.Dronens nærmiljø.

Indhold

Kapitel 1

Introduktion

AR.Dronen, [49], er en quadrokopter, [89], udviklet af det franske firma Parrot, [50]. AR.Dronen blev i 2010 introduceret som "the Flying Video Game" og promoveres stadig af producenten som et stykke legetøj, der fjernstyres fra brugerens iPhone eller iPad. Siden introduktionen har AR.Dronen vundet stor udbredelse, men ikke blot som et avanceret stykke legetøj. Forskere, [4], studerende [27], og hobbyister har taget AR.Dronen til sig som en platform og selv udviklet videre ovenpå den.

Figur 1.1: *the Flying Video Game* med to spillere med hver deres AR.Drone.

En stor del af forskningen vedrørende quadrokoptere har hidtil fokuseret på at optimere mekanikken, [58], stabiliseringen og motorstyringen. I mange eksperimenter har quadrokopteren været fjernstyret gennem ekstern observation. Eksempelvis i forskningen af små agile Mikro Arial Vehicles (MAV) i flok, [72] (publiseret 2012) er der fastmonteret kameraer i lokalet rundt om MAV-flokken til ekstern udregning og feedback af MAVens absolutte position. En anden tilgang som ligner tilgangen i dette speciales er, at alle beslutninger omkring navigation skal træffes udelukkende på baggrund af sensorinput fra sensorerne på quadrokopteren. Eksempler på anvendelse af denne tilgang kan findes i brugen af quadrokoptere til inspektionen af højspændingsledninger og broer, [41], og i eksperimenterne af Chen et al omkring navigation af AR.Dronen i forskellige indendørs miljøer, [4]. Ideen om at sensorinput til navigation skal komme fra quadrokopteren selv ses også anvendt af Shen et al. i deres forsøg med "Simultaneous localization and mapping" (SLAM), [63]. Her er quadrokopteren er påmonteret en kinect, [93] til at opfatte omgivelserne i 3 dimensioner, for derigennem selv orientere sig og konstruere modeller af omgivelserne.

1.1 Motivation

At specialet skulle beskæftige sig med AR.Dronen, var en beslutning der blev truffet meget tidligt. Argumentet for valget var, at vi ønskede at lære mere om, og få mere erfaring med, modeller af den fysiske verden og brug af modeller til kontrol af AR.Dronen. Vi havde en forventning og et håb om at den fysiske verden ville stritte imod vores modelrepræsentationer af denne.

Specialet er i løbet af projektet vokset ud af, og omkring AR.Dronen, gennem et ønske fra vores side om at undersøge den fjernstyrede quadrokopters egenskaber som robotplatform.

Igennem vores forundersøgelser af AR.Dronen er vi blevet overrasket over hvor stor interessen er og hvor stor aktivitet der er i diverse brugergrupper på internettet omkring quadrokoptere. Udfra denne erfaring mener vi, at en softwareplatform, udover at skulle støtte vores eget videre arbejde i specialet, også skal tilbyde et let tilgængeligt interface til AR.Dronen. Interfacet skal være anvendeligt for brugere, som ikke nødvendigvis er erfarne programmører. Et perspektiv kunne være, at AR.dronen og den konstruerede platform kunne bruges til at lære om programmering.

Hvis platformen laves modulopbygget, vil man kunne genbruge komponenter og derigennem evt. kunne effektivisere udviklingen af dedikerede styringer til nye arbejdsopgaver for AR.Dronen.

Langt størstedelen af arbejdet i projektet handler om at klarlægge og forstå AR.Dronen og dens muligheder.

1.2 Teseformulering/Mål

Specialet har fire mål som formuleres og uddybes således:

1. **Konstruktion af klientplatform:** Der skal konstrueres en softwareplatform til at lette fjernkontrol af AR.Dronen fra PC. Den konstruerede platform skal:
   (a) **Facilitere eksperimenter:** Understøtte de videre eksperimenter i specialet.
   (b) **Tilgængeliggøre et interface til AR.Dronen:** Konstrueres så den tilgængeliggør et let anvendeligt interface til AR.Dronen.
2. **Semiautonom navigation:** Det skal undersøges hvorvidt en quadrokopter, der er designet til kontrol via fjernstyring, kan navigere semiautonomt i et indendørs miljø.
3. **Udledning af yderligere navigationsdata:** Mulighederne for at bruge AR.Dronens ombordværende sensorsystem (udvidede sensorsystem) til at udlede højere ordens viden om navigationsmiljøet skal undersøges, samt hvorledes den ny viden kan bruges som støtte for den semi-autonome styring af AR.Dronen. Herunder undersøges konkret mulighederne for at bruge:
   (a) Lokalisering via WIFI-signalstyrker.
   (b) Lokalisering via visuelle markøre.
   (c) Detektering af navigationsmiljøtype.
   (d) Odometrisk afstandsmåling.
   (e) Optisk triangulering af afstande.
4. **Vurdering af AR.Dronen som robotplatform:** Som en del af specialets metodeevaluering skal AR.Dronens anvendelighed som robotplatform undersøges.

1.3 Metode

AR.Dronen kan flyve både indendørs og udendørs. Specialet er afgrænset til at behandle indendørs navigation. For at skabe en god basis, der kan arbejdes videre udfra, defineres specialets domæne yderligere gennem følgende brugsscenarie. Der er fundet inspiration i indendørsområder som de der er beskrevet i artiklen med den selvkørende museumsrobotguide Minerva fra 1999, [71].

Figur 1.2: Minerva i hel figur, med ansigtstræk igang med at lave guidede tours på Smithsonian museum.

Minerva er udstyret med kamera og afstandsmåler til at registrere publikum og omgivelserne, for at kunne navigere i forhold til disse. Minerva handler også om menneske-maskin interaktion, som generelt er et vanskeligt område, men løsninger til navigationsproblemet alene findes allerede i litteraturen. Et spørgsmål man kan stille sig er, hvad sker der hvis den kørende robot i museumsmiljøet udskiftes med en flyvende AR.Drone.

Figur 1.3: AR.Dronen med semi-autonom styring, svæver over de andre gæster på museet og tager billeder.

Hvilke muligheder og begrænsninger giver AR.Dronens sensorer og manøvreegenskaber. Hvilke muligheder ville det give at bruge AR.Dronen som robotplatform. Et scenarie er at lade gæster fjernstyre AR.Dronen rundt på museet. En videre udbygning er at lave et webinterface så gæster kan besøge museet fra internettet. Spørgsmålet er hvad det vil medføre. Det er nemt at forestille sig at den meget fjerne kontrol vil give en klar oplevelse af disembodyment. Der er også et sikkerhedsaspekt ved at lade en quadrokopter næsten selvstyrende flyve rundt blandt mennesker. En anden udbygning er at lade gæster optage en personaliseret flyvning, eller en forprogrammeret rutesekvens og dele den med andre som et postkort eller en souvenir.

Herover beskrives billedligt et scenarie for specialets domæne. Af hensyn til at lave praktiske forsøg, bliver de konkrete navigationsløsninger designet til at virke i Ada-bygningens, [38], kontorer og gangarealer, samt Zusebygningen, [39], på Katrinebjerg i Århus.

I det efterfølgende gennemgåes elementerne i den valgte metode.

1.3.1 AR.Dronen

AR.Dronen, som beskrives i detaljer i kapitel 2, tilfredstiller behovet for en quadrokopter-platform. AR.Dronen stiller en række sensorer tilrådighed og præsenterer et veldefineret interface til fjernstyring. Alternativt kunne man have bygget en platform fra bunden, som i [58].

Anvendelsen af AR.Dronen besværliggøres af den korte batteritid (10-12 minutters flyvetid). Dette problemet er dog minimeret, ved at anvende seks batterier og fire opladere. Desuden er der, under eksperimenter der ikke krævede flyvning, anvendt en strømforsyning fra en PC i stedet for et batteri.

Med AR.Dronen er man hurtigt i luften og der kan hurtigt bygges software der anvender det offentliggjorte interface. Til gengæld har man ingen indflydelse på det styreprogram der kører på AR.Dronen og AR.Dronen kan kun styres gennem interfacet. Det er muligt at implementere et alternativt styreprogram til AR.Dronen, men det har ikke været et mål med dette speciale.

Der gives en detaljeret vurdering af AR.Dronen som robotplatform i sektion 2.6.

1.3.2 Klientplatform og klientværktøjer

Der er implementeret en klientplatform til indsamling af AR.Dronens sensorstrømme og fjernstyring af AR.Dronen. Klientplatformen har fasciliteret de eksperimenter som beskrives i sektion 5 og dermed også udviklingen af de forskellige tasks i sektion 3.3.

Klientværktøjerne: Sensordisplay, Mapcreator og Taskcreator, som alle beskrives i sektion 4, er udviklet for at lette arbejdet med klientplatformen. Blandt andet ved at give brugeren adgang til en grafisk repræsentation af AR.Dronens sensorstrømme. Klientværktøjerne har vist sig meget anvendelige i forbindelse med forberedelse og udførelse af eksperimenter med klientplatformen.

1.3.3 Python

Klientplatformen implementeres i Python. Grunden til dette er primært Pythons læsbarhed og simple syntaks. Læsbar kode fordrer ikke bare hurtig udvikling fra vores side, men også at andre udviklere hurtigere kan sætte sig ind i koden og videreudvikle på den. Pythons simple syntaks betyder, at klientplatformen også vil henvende sig til mindre erfarne udviklere. Da det

har været et mål, at skabe en let anvendelig og let udbyggelig klientplatform, er Python et oplagt valg.

Eksemplerne i sektion 3.1 viser hvor lidt kode der behøves, for at anvende klientplatformen. Desuden henvises til implementationen af klientværktøjerne, [22], [18], [23], for eksempler der anvender klientplatformen.

Udfordringen ved valget af implementationssprog er, at Python som et dynamisk typet, fortolket sprog, ikke performance-mæssigt kan sammenlignes med mere maskinnære sprog som C og C++. For at overkomme denne udfordring, er der gjort brug af OpenCVs billedalgoritmer (beskrevet nedenunder) og Psyco, [61], i forbindelse med afkodning af billeder fra AR.Dronen (se desuden sektion 3.2.1).

1.3.4 OpenCV

OpenCV er et bibliotek af optimerede algoritmer, indenfor området computervision, [46], [80]. OpenCV blev oprindeligt startet af Intel (med bidrag fra optimeringseksperter i Intel Russia og Intels Performance Library Team), men udvikles nu af robotforskningslaboratoriet Willow Garage, [85]. Det har fra starten i 1999 været et mål, at skabe ikke bare åben, men også optimeret kode, til en grundlæggende computer vision infrastruktur.

OpenCV blev i starten udviklet i C, men den seneste udvikling er udelukkende foregået i C++. Udover C og C++, findes der en række interfaces til biblioteket, blandt andre et Python interface, [80]. Python interfacet fungerer som en wrapper om OpenCVs C/C++ algoritmer. Ved anvendelse af dette interface, kombineres altså Pythons brugervenlighed og OpenCVs hastighed.

OpenCVs optimerede algoritmer har, på grund af deres effektivitet, været uundværlige, blandt andet i implementationen af PositionSilhouetDetector (sektion 3.2.1). Forskellen i afviklingshastighed på en blobdetektor implementeret i ren Python og en der anvender OpenCVs algoritmer, var tydelig under eksperimenterne i sektion 5.2.

1.3.5 Eksperimenter

For at teste AR.Dronen og klientplatformen, er der udført en række eksperimenter, se kapitel 5. Eksperimenterne omhandler generelt udledning af kontekstuel information fra AR.Dronens sensorstrømme. Mere specifikt testes metoder til lokalisering, afstandsbedømmelse, måling af tilbagelagt afstand (Odometri) og korridordetektering.

Lokalisering via WIFI-fingerprinting: Ved hjælp af en WIFI-fingerprinting-algoritme, er muligheden for at lokalisere AR.Dronen i forhold til et antal fastlagte positioner blevet undersøgt. Eksperimenternes opstilling og resultater kan ses i sektion 5.1.

Lokalisering via visuelle markører: Der er designet et antal forskellige visuelle markører. Markørerne er blevet testet for hvor godt de kunne genkendes af AR.Dronen, både i statiske opstillinger og i forbindelse med bevægelse. Disse eksperimenter har også fungeret som test af tasksystemet og de udviklede tasks. Eksperimenternes opstilling og resultater kan ses i sektion 5.2.

Afstandsbedømmelse via optical flow: Der er udført eksperimenter for at undersøge muligheden for at udlede afstande vha. AR.Dronens frontkamera (se sektion 5.3). Ved at lade AR.Dronen flyve sidelæns, trianguleres afstande ud til genstande foran frontkameraet. Dette gøres ved brug af den tilbagelagte afstand og målte vinkler til genstanden.

Korridordetektering: Der er eksperimenteret med muligheden for at detektere et korridormiljø (sektion 5.4). Med en sådan detektion, kan AR.Dronen bevæge sig i forhold til denne specifikke miljøtype. Algoritmen analyserer på indholdet af diagonale linier i billedet fra frontkameraet.

Odometrisk afstandsmåling: En del af den navigationsdata der modtages fra AR.Dronen er hastighedsestimater. Disse hastighedsestimater er anvendt til at implementere en odometrisk afstandsmåler. Implementationen integrerer over de modtagne hastighedsestimater. Eksperimenterne med DistanceTrackeren involverer manuelle og automatiserede flyvninger over mere eller mindre ensartede overflader (se sektion 5.5).

1.4 Medfølgende CD

På den medfølgende CD findes README.txt som forklarer hvordan de medfølgende programmer afvikles, samt følgende materiale:

- `/klientplatform/`
  Kildekoden til den udviklede klientplatform inklusiv de udviklede tasks

samt klientværktøjerne.

- `/AR.Drone addons/`
  Kernemodulerne, bibliotekerne, driver-kildekoden og loadscriptet som er anvendt i forbindelse med udvikling på AR.Dronen.

- `/report/`
  Pdf-version af denne specialerapport.

- `/webcache/`
  Off-line kopier af følsomme webkilder.

- `/video/`
  Videomateriale af AR.Dronen og klientplatformen i funktion.

1.5 Projekt-blog

Under specialets udvikling, er der anvendt en blog til at beskrive tekniske fremgangsmåder og til at fungere som adgangspunkt til det offentlige kode-repository. Bloggen hostes på Github og adressen angives i [20]. Bloggen indeholder udover en projektbeskrivelse, især informationer om de tidlige forsøg med AR.Dronen. Blandt andet beskrives proceduren for at aktivere AR.Dronens USB-port i detaljer. Et andet emne som behandles på bloggen er afvikling af kode på AR.Dronen, herunder proceduren for cross-compiling til ARM arkitekturen. Procedurerne er stykket sammen fra adskillige kilder og formålet med bloggen er blandt andet at kunne tilbyde en mere komplet kilde for andre udviklere.

Kapitel 2

Quadrokopter AR.Drone

AR.Dronen er en quadrokopter fremstillet og udviklet af det franske firma Parrot, [50], [88]. AR står Augmented Reality, [7], og relaterer til muligheden for at udvikle augmented reality applikationer ved hjælp af AR.Dronens sensorstrømme. En quadrokopter benytter fire faste rotorer til at generere opdrift og styre sin bevægelse i rummet. AR.Dronen kommer med et sæt sensorer bestående af: To kameraer, et vertikalt nedadrettet og et horisontalt fremadrettet, en nedadrettet afstandssensor, et gyroskop, samt et accelerometer. I 2012 ventes en opdateret udgave af AR.Dronen. Den nye version tilføjer yderligere et kamera i høj opløsning (HD), en trykmåler (digitalt barometer) og et kompas til platformen, [48].AR.Drone er opbygget som en letvægtskonstruktion af kulfiber, plast og skummateriale og kommer med to forskellige skumskjold til henholdsvis indendørs (figur 2.1(a)) og udendørs flyvning (figur 2.1(b)).

I det følgende afsnit, *AR.Dronens Hardware* gennemgåes de fysiske enheder som udgør AR.Dronen, i afsnittet *AR.Dronens fysik* gives en introduktion til hvorledes AR.Dronen bruger rotorerne til at skabe opdrift og bevægelse i luften. I afsnittet *AR.Dronens Software* og afsnittet *Kommunikation mellem AR.Dronen og klienter* beskrives kommunikationen mellem AR.Dronens styreprogram og forskellige typer klienter, mens afsnittet *Vurdering af AR.Dronen som robotplatform* opregner de observationer der er gjort om AR.Dronen under specialearbejdet.

(a) Indendørsskjold (b) Udendørsskjold

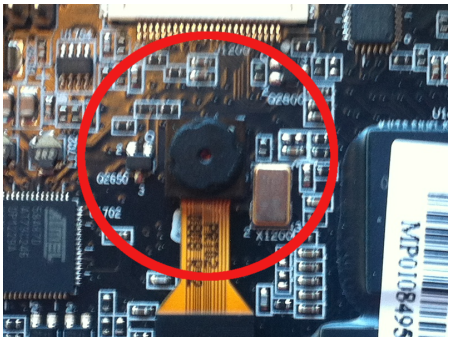
Figur 2.1: AR.Drone med indendørs- og udendørsskjold. Figurene er fra AR.Drone Developer Guide, [53]

2.1 AR.Dronens Hardware

Det er nærmest umuligt for et menneske at styre en helikopter, eller quadrokopter, ved direkte kontrol, med mindre man har trænet i mange år (hvilket er det profesionelle piloter gør, [79]). Brugerens styring af AR.Dronen er således ikke direkte, men støttet af en feedback-kontrol, hvortil der er sensorinput fra omverdenen. Herunder beskrives sensorerne, aktuatorerne, strømforsyningen og computeren ombord.

2.1.1 Sensorer

(a) AR.Dronens frontkamera



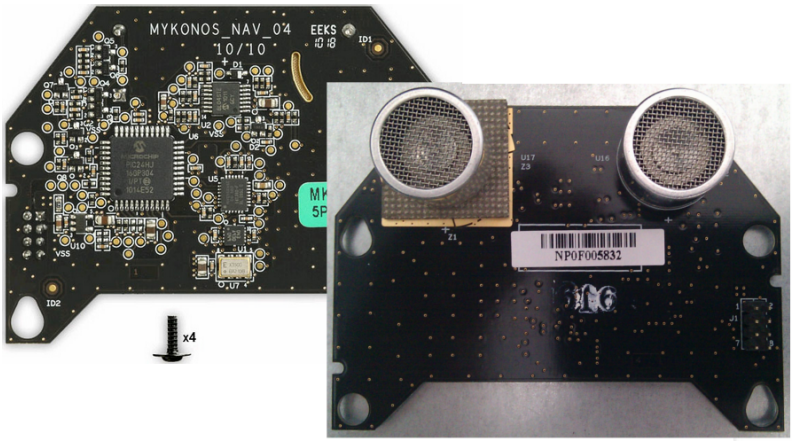
(b) AR.Dronens bundkamera

Figur 2.2: AR.Dronens to kameraer

Kameraer

AR.Dronen er som sagt udstyret med to kameraer: Et bundkamera (nedadrettet) og et frontkamera (fremadrettet). Frontkameraet har en synsvinkel på 93° og består af en CMOS[1]-billedsensor som kan levere 640 × 480 pixel billeder. AR.Dronen transmitterer dog kun med en opløsning på 320 × 240 pixels, Quarter Video Graphics Array (QVGA), med en framerate på 15 FPS. Bundkameraet er en CMOS sensor med 64° synsvinkel, der tager billeder i 176 × 144 pixels opløsning, Quarter Common Intermediate Format (QCIF), med en framerate på 60 FPS.

[1]Complimentary-Symmetry Metal Oxide Semiconductor, på dansk: en sensor af komplimentær metaloxid halvleder teknologi.



Figur 2.3: Over- og undersidebilleder af AR.Dronens navboard. Navboardet indeholder bl.a. den nedadrettede afstandssensor. Figurerne er fra to internetbutikker, [68] og, [33], hvorfra der kan bestilles AR.Drone reservedele.

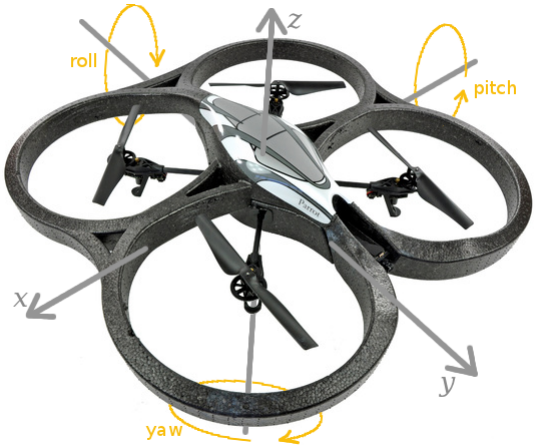
Afstandssensor

Til at bestemme AR.Dronens afstand til gulvet anvendes en ultrasonisk afstandssensor (se figur 2.3). Denne opererer ved at udsende en ultralydspuls og herefter måle tidsintervallet der går før ekkoet måles. Ultralydsmåleren er angivet til at virke op til 6 meter og er fysisk placeret på navboardet, på AR.Dronens underside.

Gyroskoper og accelerometer

AR.Dronen er udstyret med to gyroskoper: Et 2-akses gyroskop til at måle rotationsvinklerne θ og ϕ (pitch og roll) samt et 1-akses piezoelektrisk gyroskop til at måle rotationsvinklen ψ (yaw). Gyroskoperne kombineres med et 3-akses accelerometer i en samlet IMU[2]-enhed til at måle inerti. For en grafisk repræsentation se figur 2.4.

[2]Inertial Measurement Unit



Figur 2.4: AR.Dronens lokale koordinatsystem.

2.1.2 Aktuatorer

Figur 2.5: Motorreservedel, samlet motor og gear på kulfiberkryds, gear-reservedele.

AR.Dronens aktuatorer er de fire børsteløse 15 Watt motorer, [32]. Motorerne arbejder i intervallet 10350 til 41400 omdrejninger per minut (RPM) og når AR.Dronen står stille i luften, arbejder motorerne med 28000 RPM. Dette svarer, grundet gearingen, til 3300 RPM for propellerne, [32]. Motorerne har uden sammenligning det største strømforbrug af alle AR.Dronens enheder.

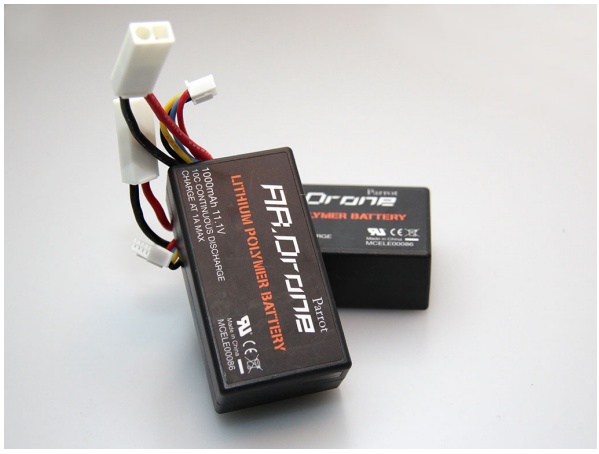
Den indlejrede computer

Den centrale computer på AR.Dronen er en ARM9 468 MHz processor (Parrot 6 ARM926EJ) med 128 MB DDR RAM til arbejdshukommelse og 128 MB NAND flash til persistent (vedvarende) hukommelse.

Tilkoblet den indlejrede computer er der et integreret trådløst netværkskort (Atheros AR6102G-BM2D) til WIFI.

På undersiden af quadrokopteren er der et 7-benet molex-stik til ekstern serialkommunikation, [65]. Porten virker bl.a. som en On-The-Go USB-port, [36], der primært anvendes ved softwareopdateringer.

2.1.3 Strømforsyning



Figur 2.6: Batterier til AR.Dronen (Figuren er fra [69]).

Batteriet der driver AR.Dronens enheder er et 3 celle lithium-polymer batteri med en open-circuit[3] spænding på 11,1 Volt. Batteriet har en ladning på 1000 milli-ampere-timer (mAh) og en afladnings hastighed på 10 Coulomb (C), [47]. Opladning af batteriet kan foregå i løbet af 90 minutter og et fuldt opladet batteri giver omking 12 minutters flyvetid, [87].

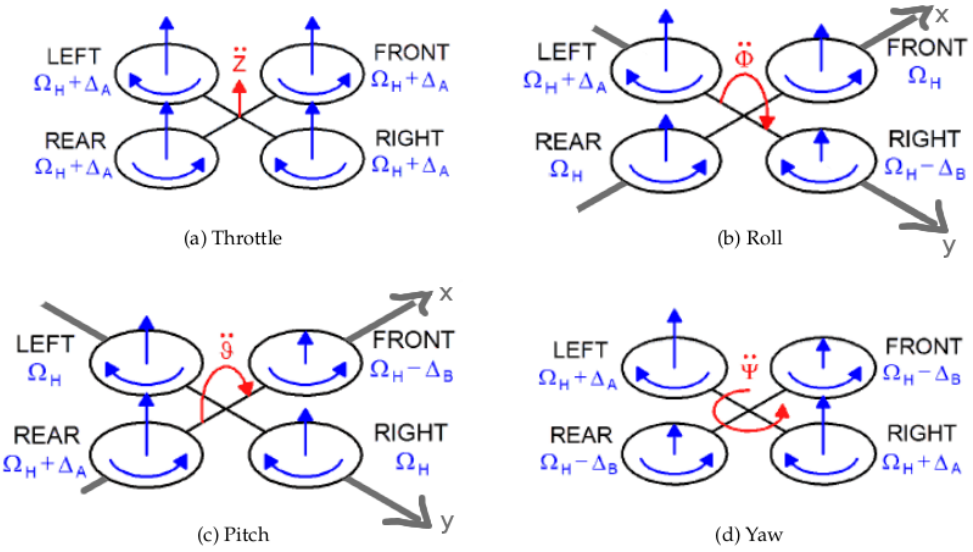
[3]Open-circuit spændingen er den spænding, der er over batteriet, når det er uden belastning og ikke oplades.

2.2 AR.Dronens fysik

AR.Dronen fåes som nævnt med to skrog eller skjold (se figur 2.1(a)), og vejer 420 gram med det indendørs skrog monteret. 420 gram er en relativ lille vægt i forhold til f.eks. [58], som beskriver en quadrokopter der skal løfte 4-5 kg.

På en AR.Drone sidder de fire propeller i hver sit hjørne af et kulfiberkryds. Når en propel roterer om sin akse flytter den luften i et areal rundt om propellen. Lufttrykket på undersiden af propellen bliver herved større end lufttrykket på oversiden af propellen. Hvis AR.Dronen ligger vandret, vil forskellen i lufttryk give en Bernoullieffekt,[3], som påvirker AR.Dronen med en kraft modsat tyngdekraften (thrust), [89]. Udover thrust påvirker den roterende bevægelse også propellens base med et moment (torque) modsat propellens retning. Eksempelvis vil en helikopter uden haleror dreje ukontrollerbart rundt om sig selv, men fordi AR.Dronen har et par rotorer som drejer med uret og et andet par som drejer modsat mod uret, bliver summen af momentpåvirkningen nul, såfremt de fire rotorer drejer med samme hastighed. Denne egenskab bruges også til at styre AR.Dronens flyvning og bevægelser i rummet, som kan beskrives med tre bevægelsesakser: x, y og z, og de tilhørende rotationsvinkler ϕ, θ og ψ (Tait-Bryan-vinklerne, [77]). På figur 2.7 forklares hvordan en quadrokopter i det generelle tilfælde kan bevæges omkring de tre akser. Det bemærkes at x og y-akserne på denne figur er forskudt 45° så de flugter med motorerne (for at lette forklaringen), hvorimod AR.Dronens akser er forskudt så y-aksens positive retning peger i samme retning som frontkameraet, se figur 2.4.

AR.Dronens bevægelser styres ved at kontrollere dens vinkel i forhold til de 3 akser, ligesom på figur 2.7. Skal AR.Dronen f.eks. flyve fremad, ændres θ ved at sænke omdrejningshastigheden på de to forreste propeller, samtidig med at omdrejningshastigheden på de to bagerste propeller øges. Dette resulterer i at AR.Dronen hælder lidt fremad. Samme princip gør sig gældende når AR.Dronen skal flytte sig sideværts. Når AR.Dronen skal bevæges om z-aksen foregår det som på figur 2.7d. Det vil sige, at man enten øger eller sænker omdrejningshastigheden på de propelpar som drejer samme vej således, at det ene pars momentpåvirkning overstiger det andets og dermed får AR.Dronen til at rotere.



Figur 2.7: (a) vinkelhastigheden Ω_H øges med Δ_A på alle 4 propeller, dermed øges quadrokopterens løftekraft (acceleration i Z-aksen), (b) Vinkelhastigheden øges på venstre og mindskes på højre propel, quadrokopteren laver et "roll" til højre (ændring i ϕ), (c) samme som b, men i pitch (ændring i θ) (d) En større vinkelhastighed på de med-uret-drejende propeller gør at torque bliver ulige og at quadrokopteren roterer (yaw) mod venstre (ændring i ψ). Ved bevægelser frem og tilbage udledes en positiv eller negativ hastighed (V_x), ligesom ved bevægelser sideværts (V_y). Figuren er fra [53].

Når AR.Dronen går fra at ligge vandret til istedet at hælde, flyttes noget thrust fra at virke lodret imod tyngdekraften til også at virke sideværts. Dette betyder, at propellernes samlede omdrejningshastighed skal øges for at opretholde den samme afstand til gulvet. Ifølge SDK DevGuide, [53], bør ϕ og θ ikke overstige 0.52 rad (30°) ellers kan flyvehøjden ikke opretholdes.

AR.Dronens styreprogram har en parameter, *Euler_angle_max* , der beskriver den øvre grænse for hvor meget quadrokopteren maximalt må krænge. *Euler_angle_max* virker som en cut-offvinkel, dvs. at AR.Dronen under flyvning løbende tester, at hverken pitch (θ) eller roll (ϕ) overskrider grænsen. Sker dette lukkes systemet ned, kraften til rotorene stoppes og AR.Dronen falder til jorden.Den maximale hældning kan sættes til f.eks. 0.25 rad via

AT-kommandoen vist i figur 2.8.

```
AT*CONFIG=605,"control:euler_angle_max","0.25"
```

Figur 2.8: Eksempel på indholdet af en AT-kommandobesked. Beskeden overføres fra klient-enheden til AR.Dronen på UDP-port 5556. Lige netop denne besked sætter en parameter i AR.Dronens styreprogram, der bestemmer den maksimalt tilladte hældning på ϕ og θ.

2.3 AR.Dronens Software

AR.Dronens computer kommer med en Linux 2.6.27 kerne installeret. Det bemærkes at 2.6.27 er den første Linuxkerne med indbygget support for UBIFS, [92], som er det filsystem der anvendes på AR.Dronens NAND flash-hukommelse.

2.3.1 Busybox Linux platformen på AR.Dronen

Udover Linuxkernen, leveres AR.Drone også med Busybox installeret, [76]. Busybox implementerer en række Unix værktøjer (`cd`, `cp`, `ls`, `mv`, osv.) optimeret til brug i indlejrede systemer. Busybox konfigureres så kun de ønskede værktøjer medtages og disse pakkes til en enkelt eksekverbar fil. Dette betyder, at størrelsen holdes på et minimum. Busybox på AR.Drone fylder 483 kbyte og udstyrer AR.Dronen med de brugerkommandoer man forventer på et standard Linuxsystem. AR.Dronens Busybox leverer DHCP-serveren, [31], som AR.Dronen anvender til at tildele IP-adresser, [55], og lave et netværk mellem en ekstern klient og AR.Dronen selv. Desuden leverer Busybox også en telnet-server, [56], der giver adgang til AR.Dronens terminal og en texteditor (Vi) der gør det muligt at redigere filer direkte på AR.Dronen.

2.3.2 AR.Dronens styreprogram

Det styreprogram der står for selve motorkontrollen, transmitteringen af data og bearbejdning af sensordata i forbindelse med flyvning, er implementeret i `/bin/program.elf` på AR.Dronen. Da der er tale om proprietært software

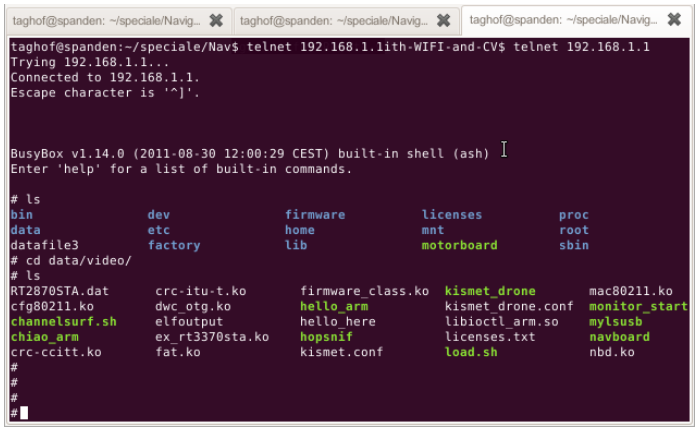
betragtes programmet som en lukket kasse, hvor implementationsdetaljer ikke er kendt. Forskellige elementer af `program.elf` kan dog udledes, efter at have observeret AR.Dronen under flyvning. `Program.elf` indeholder blandt andet:

- En PID controller, [81], [1], til at stabilisere AR.Dronen når der svæves.
- En algoritme til at udlede AR.Dronens hastighed ved hjælp af bundkameraet, [5].
- En algoritme til at detektere tags i billeder, til brug for augmented reality spil, se figur 2.9.

AR.Dronens styreprogram modtager styringskommandoer (AT-kommandoer) fra brugeren gennem en User Datagram Protocol (UDP)-port, [54]. Kommunikationsinterfacet mellem klientenhed og AR.Dronen beskrives i afsnit 2.4.

(a) Roundel (b) Hull (c) stripes

Figur 2.9: Visuelle tags der kan genkendes af AR.Dronen, billeder fra [62] og [53]



Figur 2.10: Forbindelse til AR.Dronens Telnetserver på AR.Dronens standard IP addresse 192.186.1.1.

2.4 Kommunikation mellem AR.Dronen og klienter

AR.Dronen fjernstyres oftest via en iPhone, men dette er blot en af flere interaktionsmuligheder. Herunder listes nogle af de muligheder AR.Dronens Linux-platform og AR.Dronens styreprogram åbner op for, både i form af fjernstyring og intern kommunikation på AR.Dronen.

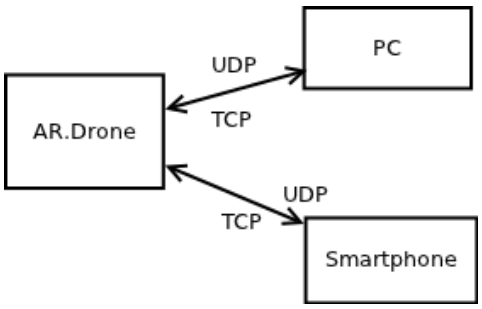
Figur 2.11: Fjernstyring af AR.Dronen med en Smartphone, her en Nokia-telefon. Figuren er oprindeligt fra [43].

2.4.1 Ekstern klientkommunikation

En ekstern klients (PC, smartphone eller lignende) interaktion med AR.Dronens Linux-platform og AR.Dronens styreprogram kan kategoriseres i 2 grupper:

- Fra en ekstern klientapplikation (FTP, [57], eller Telnet, se figur 2.10) over WIFI til services kørende i userspace på AR.Dronen.
- En ekstern klientapplikation kan sende AT-kommandoer[4] via UDP protokollen til AR.Dronens styreprogram og kan modtage sensordata fra AR.Dronens styreprogram over trådløs UDP.

[4] AT(tention)-kommandoer over trådløst netværk (gennemgåes nærmere i sektion 2.4.5)

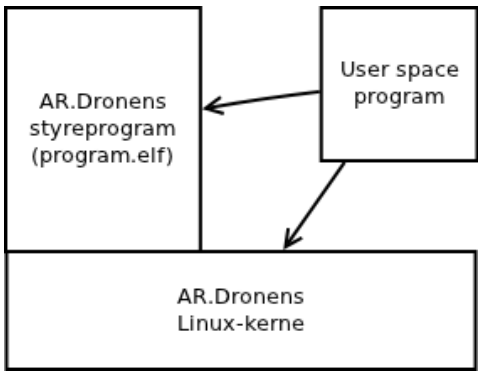


Figur 2.12: Klienter kommunikerer med AR.Dronen over netværket gennem UDP og TCP.

Interaktion mellem AR.Dronen og en ekstern klient regnes som værende fjernstyring, mens AR.Dronen regnes som værende autonom hvis kontrolsignaler til AR.Dronens styreprogram kommer fra en intern process, der kører parallelt med AR.Dronens styreprogram på AR.Dronen.

2.4.2 Intern klientkommunikation

Interne processer på AR.Drone-platformen har mulighed for at kommunikere med AR.Dronens styreprogram gennem UDP, på lige fod med de eksterne klienter til fjernstyring. Et eksempel på en intern klient gives i blogposten "UDPing ATcommands from inside the drone", [24]. En intern process vil afvikles i userspace, en nærmere beskrivelse herom gennemgåes i sektion 2.5.



Figur 2.13: En intern klient på AR.Dronen, kommunikerer med AR.Dronens styreprogram på 3 UDP-porte, men har også adgang til platformen som ethvert andet userspaceprogram.

2.4.3 FTP og Telnet baseret kommunikation

Fra en PC eller smartphone er det muligt at få adgang til AR.Dronens Linuxinstallation, ved at oprette en Telnetforbindelse til AR.Dronens faste IP-adresse: 192.168.1.1. Efter forbindelsen er etableret, får brugeren adgang til en root-terminal (se figur 2.10).

Ved at oprette en FTP-forbindelse til AR.Dronens FTP-server, fra en FTP-klient, kan man overføre filer mellem AR.Dronen og en PC med FTP-protokollens PUT og GET kommandoer. Hvis der ikke angives en sti lægges overførte filer som standard i `/data/video` mappen på AR.Dronen.

2.4.4 UDP baseret kommunikation

I modsætning til FTP- og Telnet-klienterne, som er TCP baserede, anvendes UDP-protokollen istedet til kommunikationen imellem intern og eksterne klienter og AR.Dronens styreprogram.

AR.Dronens styreprogram sender og modtager pakker på tre UDP-porte:

- På port 5556 (command port) lytter styreprogrammet efter AT-kommandoer afsendt fra klienten, AT-kommandoer gennemgåes i 2.4.5.

- på port 5555 (video port) sender AR.Dronen en videostrøm, indeholdende billeder fra èn af fire videokanaler til klienten.
- på port 5554 (navdata port) sender AR.Dronen en navdatastrøm indeholdende statusinformation om f.eks. hastigheder, eulervinkler, afstand til gulvet og generel information om navigationstilstanden.

2.4.5 AT kommando interface

En klient som skal kommunikere med AR.Dronens styreprogram, gør dette ved hjælp af såkaldte attention-kommandoer (AT-kommando). En AT-kommando er basalt set blot en tekststreng repræsenteret som 8 bits karakterer. AT-kommandoer sendes til AR.Dronens kommandoport (5556) som UDP-pakker. Formatet for AT-kommandoer ses på figur 2.14.

```
AT*[kommandonavn]=[sekvensnummer],[arg1, arg2 ... argN]<LF>
```

Figur 2.14: Syntaksen for en AT-kommando: [kommandonavn] kan f.eks. være `PCMD`. [sekvensnummer]-heltallet skal være stigende for hver ny afsendt AT-besked. Listen med argumenter: [arg1, arg2 ... argN], varierer i længde afhængig af konteksten. Ved afsending af en `PCMD`-kommando skal der medsendes 5 talværdier for hhv. flag, roll, pitch, gas og yaw. Se tabel over alle AT-kommandoer i appendiks A.1

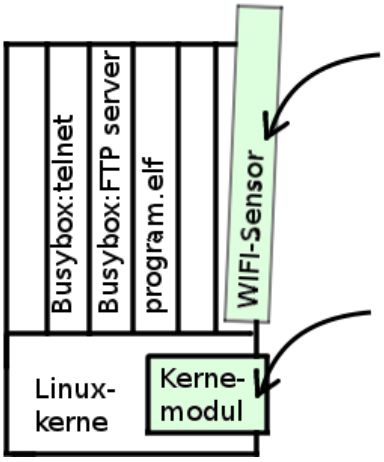
Ved at afsende kommandoen i figur 2.15 bestemmer man hvorledes AR.Dronen skal bevæge sig ved translation og rotation, som beskrevet i AR.Dronens fysik, sektion 2.2.

```
AT*PCMD=21625,1,0,0,0,0<LF>
```

Figur 2.15: Eksempel på en `PCMD` AT-kommando. Denne specifikke kommando bringer AR.Dronen i hovertilstand. [sekvensnummer]'et er 21625. Listen med argumenter er hhv. flag=1, roll=0, pitch=0, gas=0 og yaw=0.

Brugen, syntaksen og betydningen af 7 forskellige AT-kommando navne beskrives i detaljer i afsnit 6 af AR.Drone Development Guide, [53]. Det skal dog nævnes, at man i reglen skal sikre sig, at man vedbliver at sende beskeder periodisk, for at AR.Dronen ikke skal tolke forbindelsen til klientenheden som tabt.

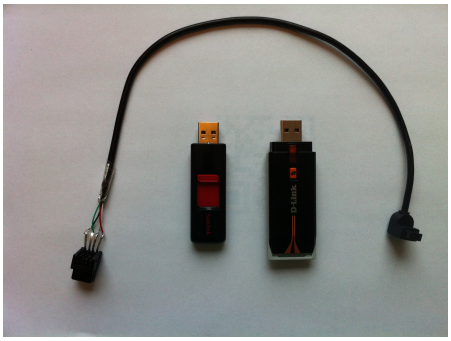
2.5 Udvidelse af AR.Dronen med USB-moduler



Figur 2.16: Platformstrukturen på AR.Dronen gør det muligt at tilføje kernemoduler, biblioteker og at køre programmer i userspace.

En robotplatform der kan udvides med nye sensorer efter behov, er klart mere anvendelig end en helt lukket platform. Derfor blev det undersøgt hvorvidt, det er muligt at anvende AR.Dronens OTG-USB-port, [36], til andet end softwareopdateringer. Dette har blandt andet indbefattet at kompilere kernemoduler til AR.Dronens linuxkerne, samt at omgå AR.Dronens styreprogram for ikke at miste strømmen til USB-porten, [15]. Det indledende proof-of-concept var, at få mounted en almindelig USB-memorystick, at skrive til den og at læse fra den. Indstallationsproceduren er forholdsvis simpel, idet Linuxdriverne til dette formål er meget generiske. Proceduren for at ak-

tivere AR.Dronens USB-port og læse fra en USB-memorystick, kan læses i blogposten "Enabling the Drone USB Port", [15].



(a) Hjemmelavet USB-kabel, en USB memorystick og en WIFI-USB-adapter til tilslutning på AR.Dronen.

(b) AR.Dronen med USB memorystick tilsluttet.

2.5.1 WIFI-sensor på AR.Dronen

Efter de indledende erfaringer med AR.Dronens USB-port, blev der implementeret en WIFI-sensor, for på sigt at kunne anvende denne som lokaliseringsenhed. WIFI-sensoren består af en USB-WIFI-adapter og et *WIFI-sensor-program* til at aflæse og videresende signalstyrkerne fra AR.Dronens omkringliggende accesspoints og andre WIFI-kilder.

Signalstyrken fra et accesspoint siger generelt noget om hvor langt modtageren er fra accesspointet. Hvis man ved på hvilken etage og i hvilket lokale et accesspunkt er placeret, og man modtager et signal med svag signalstyrke, så ved man at man er langt væk fra den placering. Hvis man istedet modtager et kraftigt signal fra samme accespoint, så ved man at man er tæt på placeringen. Lokalisering via WIFI-fingerprinting bruger radiosignalstyrker til at give specifikke lokationer unikke karakteristika (WIFI-fingerprints). Et WIFI-fingerprint kommer i form af en mængde MAC[5]-adresser med tilhørende målte signalstyrkeværdier på en given lokation, se sektion 5.1 for en detaljeret beskrivelse af WIFI-fingerprinting.

[5]Media Access Control

For at teknikken skal fungere bedst, skal WIFI-sensor-programmet modtage og behandle så mange WIFI-pakker som muligt. Derfor sættes WIFI-adapteren i monitor og promiscous mode[6], så den modtager alle pakker der sendes fra de omkringliggende WIFI-kilder. Netværksinterfacet indstilles gennem et script (load.sh, [16]) som kaldes i forlængelse af AR.Dronens oprindelige bootup-sekvens. Scriptet sørger også for at indlæse alle de nødvendige kernemoduler og starte WIFI-sensor-programmet.

WIFI-sensor hardware

AR.Dronen er blevet udstyret med en D-Link DWL-G122 WIFI-adapter, [30]. Denne adapter blev valgt fordi den bygger på et Ralink chipset, [84] og understøtter monitor mode. Desuden udgiver Ralink linuxdrivere til deres chipsets. Efter længere tids søgen og eksperimenteren, blev den korrekte driver fundet og kompileret til ARM-arkitekturen så den var kompatibel med linuxkernen der anvendes på AR.Dronen. Når driveren er indlæst, genkendes adapteren og interfacet *ra0* oprettes i `/sys/class/net/`.

WIFI-sensor software

De første eksperimenter med hensyn til at kompilere og afvikle kode på AR.Dronen, kan ses i blogposten "Compiling code for the AR.Drone", [13]. Det var simple 'hello world' eksempler, men de gav indsigt i proceduren omkring crosscompiling til AR.Drone-arkitekturen og den anvendte toolchain.

For at skrive et program der kan modtage pakker fra et netværksinterface anvender man ofte linux' pcap-bibliotek, [67]. Dette er dog ikke installeret på AR.Dronen som standard og skulle først kompileres til ARM-arkitekturen, før WIFI-sensor-programmet kunne linke til det (se blogposten [13]).

WIFI-sensor-programmet kører på AR.Dronen. Det henter WIFI-pakker som det installerede netværksinterface, gennem WIFI-adapteren, henter fra luften. De modtagne WIFI-pakker indlæses og MAC-adressen på pakkens afsender, samt signalstyrken puttes i en ny besked, som afsendes fra AR.Dronen. De afsendte pakker er del af en sensorstrøm, WIFI-strømmen, og ligner det der ellers kommer fra AR.Dronen (navdata og video). Strømmen fra WIFI-sensoren går over UDP-port 5551. Kommunikationen anvender denne port, for at holde klientens interface til AR.Dronen nogenlunde konsistent.

[6]Normalt vil et netværksinterface frasortere og ikke videresende datapakker der ikke er adresseret til det, men i promiscous mode sendes al trafik videre.

WIFI-sensor-programmet kan opstartes med et interfacenavn som input-argument, men som standard anvendes *ra0*-interfacet. Der kan desuden vælges at angive en fast kanal, eller at anvende channelhopping til at modtage pakker fra forskellige kanaler. Derudover er det muligt at få tekstuel output til terminalen. For ikke unødigt at spilde processorkrafter med at modtage og behandle pakker, hvis de alligevel ikke skal bruges, venter WIFI-sensor-programmet efter opstart på et start-signal fra klienten via en UDP-initialiseringspakke. Modtages startsignalet begynder WIFI-sensor-programmet at sende MAC-adresser og signalstyrker videre til klienten.

2.6 Vurdering af AR.Dronen som robotplatform

Under den indledende informationssøgning til dette speciale og gennem diverse forsøg og eksperimenter undervejs, er der indsamlet en del erfaringer med AR.Dronen. Nogle af de mest markante observationer er opregnet her.

2.6.1 Hastighed og inerti

Det giver mening at beskrive AR.Dronens bevægelser i 3 dimensioner, for derved nemmere direkte at kunne bruge accelerometer- og gyro-målinger fra sensorerne om bord. Under arbejdet har det kunnet konstateres, at θ og ϕ-værdierne ikke afviger fra det forventede, hvorimod ψ-værdien har en tendens til at blive mere og mere upræcis under flyvning, hvilket også bekræftes i [52].

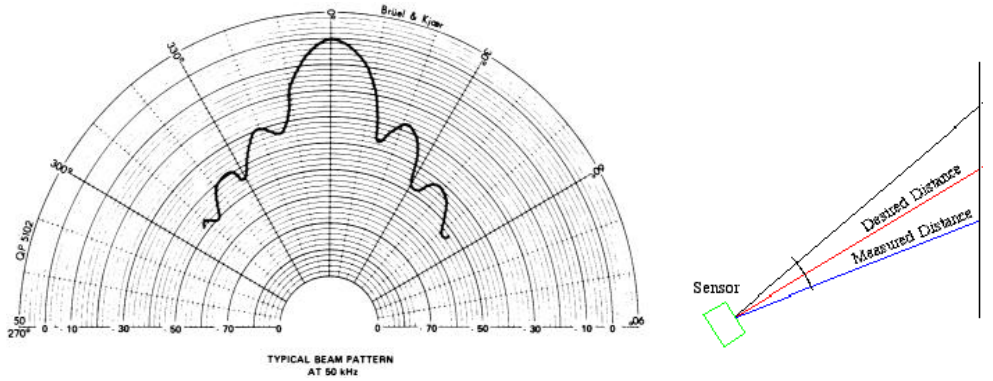
En observeret egenskab er, at man ikke kan sige noget udfra accelerometer og gyroer om *præcis* hvor langt AR.Dronen har fløjet eller *præcis* med hvilken hastighed AR.Dronen bevæger sig. Der kan dog udledes estimater for hastigheder ved at integrere over accelerationen, men hastigheden i det horisontale plan er i praksis afhængig af miljøet og kan ikke umiddelbart bestemmes. Roll og pitch værdierne kan være aflæst til 0, men AR.Dronen kan stadig godt bevæge sig sideværts i luften[7] på grund af indebåren inerti eller f.eks. en stille konstant vind. Bevæger AR.Dronen sig med en konstant hastighed, kan det ikke aflæses af accelerometeret. Parrot har derfor underbygget hastighedsestimatet med en optical-flow algoritme i styreprogrammet,

[7] AR.Dronen siges at være "trimmet" (med en konstant hastighed, evt. $0m/S$, men også forskellig fra $0m/S$) hvis den ikke påvirkes af en resulterende kraft og acceleration dermed er 0

der bearbejder data fra bundkameraets billedstrøm, således at hastighedsestimatet bliver mere troværdigt, [5].

Dette speciales egne eksperimenter med hastighedsudledning er beskrevet i afsnit 5.5. De viser at hastighedsudledningen under de rette forhold er udmærket brugbar. Udledningen af hastighed besværliggøres ved forhold med dårlig belysning og ved gulvflader uden forskelligartet tekstur, hvor bundkameraet ikke kan detektere markante features.

2.6.2 Afstandsmåleren



Figur 2.17: a) Udbredelsesområde for ultralydssensorsignal, b) man ser at afstanden ikke nødvendigvis er vinkelret på sensoren, idet der kan forekomme ekko fra et større område under sensoren (figuren er fra [26])

Den nedadrettede afstandsmåling varetages som nævnt af en ultralyds-sender og -modtager. Det vil sige højden er udledt af den tid der går fra en lydpuls afsendes, til der modtages et ekko. Ekkoet behøver ikke komme fra en stor overflade som en væg eller et gulv, men kan være en kasse inde i sensorområdet som figur 2.17a. Teknologien gør således, at afstanden der returnes ikke nødvendigvis er fra det vinkelrette punkt udfor sensoren, men derimod den mindste afstand indenfor et område som det på figur 2.17. Dette kan specielt være et problem hvis der flyves i et trangt lokale med kasser, stole og borde på gulvet, hvor de mange legemer vil give en falsk opfattelse af højden til gulvet. Fordi teknologien netop er lyd, vil der også være problemer med ekko på skrånende og bløde overflader.

De ovennævnte problematikker kan have indflydelse ved udvikling med AR.Dronen. Specielt ved flyvning i høj hastighed, kan vinklen til gulvet indføre fejl i højdemålingen, da AR.Dronen i disse tilfælde hælder meget.

2.6.3 Realtidssystem

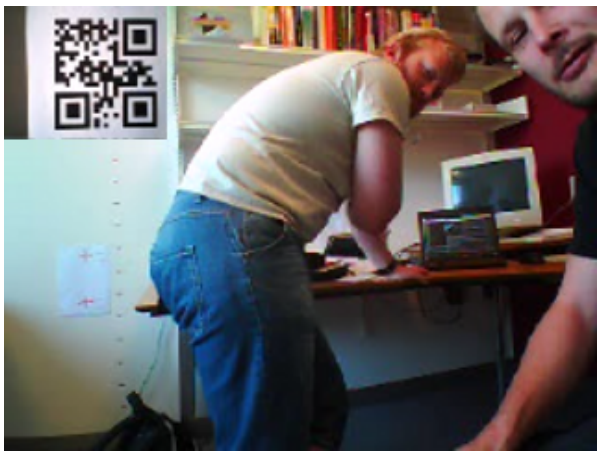
På forummet AR Drone Flyers, [66], har en bruger skrevet, at Linuxinstallationen, styreprogrammet og de eksterne processer der kører på AR.Dronen tilsammen bruger op mod 80% af CPU-kraften. Om det lige akkurat er 80% kan formentlig diskuteres. Under alle omstændigheder, skal man tage højde for den begrænsede CPU-ressource, hvis man vælger at afvikle yderligere processer på AR.Dronen, som f.eks. USB-understøttelse og en WIFI-sensor, som beskrevet i afsnit 5.1. Pointen er at man ikke kan bruge mere end de resterende f.eks. 20%, uden at det vil gå ud over kontrollen med flyvningen.

2.6.4 Trådløs båndbredde

På grund af den manglende transmissionskontrol, opererer UDP-protokollen med et forholdsvist lille kontroloverhead. Det gør, at der i princippet kan sendes flere f.eks. videopakker end med en synkroniseret kommunikationsprotokol som TCP. Det giver mest mening at anvende en asynkron protokol i realtidskommunikationen med AR.Dronen, idet det ikke kan betale sig at generhverve det fåtal af billedframes der går tabt pga. fejl i transmissionen. Det er bedre at indlæse den næste datapakke istedet. Informationen i den gamle datapakke vil alligevel være forældet og irrelevant. Erfaring har da også været, at frameraten både til video og navdatastrømmen har været passende.

2.6.5 Billedkvalitet

AR.Dronen er et produkt designet til at være billig i produktion, for dermed let at kunne erhverves af menigmand. I deres valg af design har Parrot derfor gjort et tradeoff mellem videokvalitet og framerate. Det betyder, at opdateringshastigheden af billeder og data fra AR.Dronen som nævnt er tilstrækkelig i de fleste tilfælde, men, som det fremgår af afsnit 5.2.4, er billedkvaliteten ikke tilstrækkelig til f.eks. afkodning af QR-koder, [82].



Figur 2.18: Et billede fra AR.Dronens bundkamera (88 × 72 pixels) i øverste venstre hjørne af et billede fra frontkameraet (320 × 240 pixels).

2.6.6 Skiften mellem videostrømme

I videostrømmen mellem en klient og AR.Dronen sendes kun et billede ad gangen. Man kan modtage fire forskellige slags billeder: fra frontkameraet, fra bundkameraet, fra frontkameraet med bundkameraets billede overlagt i øverste venstre hjørne og fra bundkameraet med frontkameraets billede overlagt i øverste ventre hjørne (de fire videostrømme). Skal der bruges billeder i fuld størrelse fra begge kameraer, er det nødvendigt at skifte imellem to videostrømme og skiftevis få et billede fra front- og bundkamera. Dette er ikke praktisk, idet det nedsætter opdateringshastigheden i eksempelvis en feedback-kontrol løkke. Anvender man de kombinerede billeder (se figur 2.18) skal man være opmærksom på, at det billede der er indeholdt i det andet, er af noget mindre opløsning end det originale (88 × 72 pixels for bundkameraets billede når det er lagt ovenpå frontkameraets, mod 176 × 144 pixels originalt). Denne forskel i kvalitet har tydeligt en indflydelse på resultatet af diverse billedanalysealgoritmer, se sektion 5.2.3.

Kapitel 3

Klient platform

Den grundlæggende platform er udviklet med det formål, at understøtte det videre arbejde med AR.Dronen. Platformen skal sikre let og effektiv adgang til AR.Dronens sensor output, samt give mulighed for at styre AR.Dronen, både manuelt med joystick og tastatur gennem en PC og ved hjælp af forprogrammerede sekvenser af bevægelser (tasks, beskrives nærmere i sektion 3.3).

3.1 Platformens anvendelse

Python blev, som tidligere nævnt, valgt for at lette tilgængeligheden for brugeren. Et af specialets mål er, at stille en let anvendelig og let udbyggelig platform til rådighed for udviklermiljøet omkring AR.Dronen.

Herunder vises simple eksempler på anvendelsen af dele af klientplatformen.

3.1.1 Nem fjenstyring af AR.Dronen

Et simpelt eksempel på anvendelse af det implementerede kontrolinterface gives på figur 3.1. Først startes pythons fortolker, herefter importeres examplesmodulet og metoden `square` afvikles. Det antages, at computeren som afvikler eksemplet er forbundet med AR.Dronens trådløse netværk.

```
>>> import examples as e
>>> e.square()
```

Figur 3.1: Import og afvikling af simpelt kontrolinterface-eksempel i pythonfortolkeren

Metoden `square` der kaldes i eksemplet, lader AR.Dronen lette, flyve rundt i en firkant og for herefter at lande igen. Bevægelsesmønstret opnåes ved at der kaldes en metode (`take_off`, `move` eller `land`) på kontrolinterfacet (sektion 3.2.2), hvorefter der er en pause før den næste metode kaldes. Metodekaldende oversættes af kontrolinterfacet til AT-kommandoer og sendes så som UDP-pakker til AR.Dronens kommandoport. Den konkrete implementation af `square` ses i figur 3.2. Det bemærkes, at kontrolinterfacet skal startes (ved kald af `c.start()`) før kommandoerne har nogen effekt.

```
def square():
    import controllers
    import time

    c = controllers.ControllerInterface()
    c.start()

    print 'taking off'
    c.take_off()
    time.sleep(5.0)
    print 'moving Forward'
    c.move(0.0, 0.2, 0.0, 0.0, True)
    time.sleep(2.0)
    print 'moving right'
    c.move(0.2, 0.0, 0.0, 0.0, True)
    time.sleep(2.0)
    print 'moving back'
    c.move(0.0, -0.2, 0.0, 0.0, True)
    time.sleep(2.0)
    print 'moving left'
    c.move(-0.2, 0.0, 0.0, 0.0, True)
    time.sleep(2.0)
    print 'landing'
    c.land()

    c.stop()
```

Figur 3.2: Python implementation der importerer og anvender kontrolinterfacet til en simpel flyvning. De fire første numeriske parametre der gives til move-metoden repræsenterer henholdsvis roll, pitch, thust og yaw. Den boolske parameter indentificerer overfor kontrolinterfacet, om der tale om værdier fra en fysisk kontrolenhed (XBox360 controller)

3.1.2 Modtagelse af sensordata fra AR.Dronen

I det følgende gennemgåes mulighederne for at modtage datapakker med video, navigation- og WIFI-data fra AR.Dronen. For at kunne anvende AR.Dronen som en robot, er det nødvendigt at kunne indsamle data fra de ombordværende sensorer, samt at kunne stille disse data til rådighed for andre dele af systemet. Til dette formål er der designet og implementeret en receiverstruktur bestående af tre receiver-moduler. Disse moduler eksekveres i hver deres process og sørger selv for at initiere forbindelsen med AR.Dronen. Efter initialiseringen lytter hver receiver efter enten video-, navigations- eller WIFI-pakker, når en sådan modtages udføres en passende afkodning af pakken og den afkodede data deles med processen der startede receiveren. De tre receivere er ikke afhængige af hinanden og kan afvikles hver for sig, således kan klientplatformen afvikles med en eller flere receivere tilkoblet. Sektion 3.1.3 giver et eksempel på hvordan videoreceiveren kan anvendes, til at modtage og vise et enkelt billede fra AR.Dronens videostrøm. Anvendelsen af de to andre receivere foregår på fuldstændig samme måde som videoreceiveren, den eneste forskel er typen af data der modtages.

3.1.3 Eksempel på modtagelse af videostream

Eftersom videoreceiveren kan anvendes for sig selv, vises her et eksempel på hvor få linier kode der behøves for at modtage og vise et billede fra AR.Dronens kamera. Eksemplet antager at der allerede er oprettet en virkende forbindelse med AR.Dronens trådløse netværk. Eksemplet startes fra Pythons kommandopromt.

```
>>> import examples as e
>>> e.receive_and_show_picture()
```

Figur 3.3: Import og afvikling af simpelt videoreceiver-eksempel i Pythonfortolkeren

Som det ses på figur 3.4, kræver det tre liniers kode for at modtage et billede (og endnu fire for at vise det). Dette illustrerer fint, hvordan dette

abstraktionslag tilbyder et let anvendeligt interface til AR.Dronens forskellige sensoroutput.

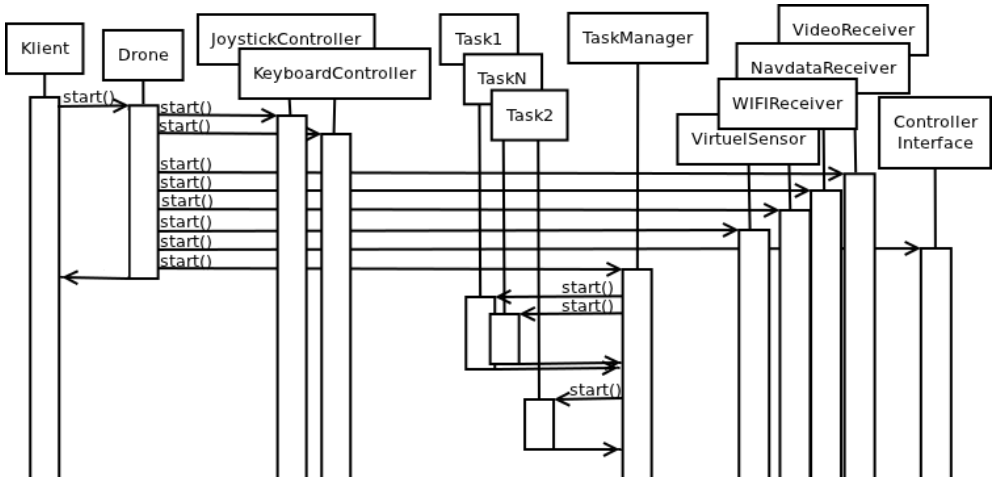
```
def receive_and_show_picture():
    import receivers, settings, time
    import cv2.cv as cv

    video_sensor = receivers.VideoReceiver(VIDEO_PORT)
    video_sensor.start()
    time.sleep(1)
    pic = video_sensor.get_data()

    cv.StartWindowThread()
    win = cv.NamedWindow('win')
    cv.ShowImage('win', cv.fromarray(pic))
    cv.WaitKey()
    cv.DestroyWindow('win')

    video_sensor.stop()
```

Figur 3.4: Python implementation der importerer og anvender videoreceiver til at modtage og vise et billede

3.2 Platformens opbygning

Platformen består af et antal klasser placeret i et antal Python-moduler som beskrives nærmere i det følgende. Her er indledningsvis en kort introduktion for overblikkets skyld. Der er udviklet tre forskellige receiverklasser til at modtage henholdvis video, navigationsdata (navdata) og WIFI-data. Disse beskrives i sektion 3.2.1. Der er udviklet to forskellige controllerklasser til joystick og keyboard som beskrives i sektion 3.2.3. Der er kontrueret en taskmanager (sektion 3.2.5) med ansvar for at opstarte autonome delopgaver kaldet tasks (tasks beskrives i sektion 3.3). For at styre AR.Dronen anvendes et kontrolinterface. Et simpelt eksempel herpå er allerede vist i figur 3.2. Kontrolinterfacet gennemgåes i sektion 3.2.2. Med inspiration fra [40] og [29] er der, udover de allerede nævnte grundreceivere, implementeret to virtuelle sensorer (sektion 3.2.1) som bruges til at fortolke datastrømmene fra de almindelige receivere.

Platformens klasser instantieres, startes og tilgåes som udgangspunkt gennem droneklassen (dronemodulet). På denne måde kan udviklere der ønsker at anvende platformen blot importere dronemodulet og instantiere droneklassen, hvorefter der er let adgang til diverse sensordata, se sekvensdiagrammet i figur 3.5.

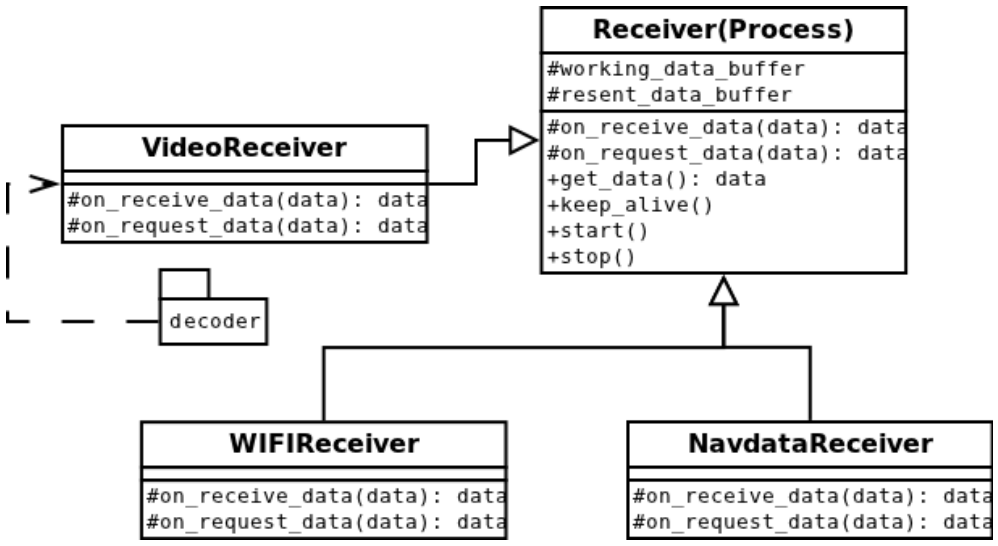


Figur 3.5: Klientapplikationens entrypoint til platformen er gennem instantiering af drone-klassen, som sørger for at instantiere resten af objekterne og processerne i platformen.

Udover ovenstående er der lavet et testdevice, som anvendes i forbindelse med udvikling, når det ikke er muligt at have en AR.Drone fysisk tilstede. Sektion 3.2.4 beskriver muligheden for at simulere output fra AR.Dronen gennem testdevicet.

3.2.1 Datastrømme-modtagere på klienten

Subklasser af *Receiver* er abstraktioner over de fysiske hardwaresensorer på AR.Dronen. Receivernes opgaver er at initialisere UDP-kommunikationen med AR.Dronen ved at sende en startbesked og herefter modtage de datapakker AR.Dronen sender. Desuden skal datastrømmen holdes i live ved at sende et watchdog-signal i faste intervaller.



Figur 3.6: Klassehierakiet for receiverklasser. Nedarvende klasser kan specialisere sig ved at overskrive de to metoder `on_receive_data` og `on_request_data`.

Receiverklasser nedarver fra baseklassen Receiver (se figur 3.6), der indkapsler den basale receiverfunktionalitet. For bedre at kunne udnytte en multicore processor, er denne baseklasse implementeret som en selvstændig proces ved hjælp af Pythons multiprocessing-modul. Receiverprocessen kommunikerer ved hjælp af en delt liste, som anvendes både til at overføre signaler og data. For at tilpasse de enkelte receivere implementerer disse specifikke metoder (for eksempel til afkodning af den specifikke type data) der kaldes på bestemte tidspunkter i det generelle receiverloop.

Navdata-modtager

NavdataReceiveren modtager AR.Dronens navdatastrøm (AR.Dronens afstand til jorden, flyvehastighed i x- og y-retning m.m.) fra AR.Dronen og decoder denne, i forhold til de structs der findes beskrevet i headerfilen navdata_common.h, [37], fra Parrots SDK.

Video-modtager og billedafkoding via Psyco

VideoReceiveren modtager og afkoder AR.Dronens videostrøm. Afkodningen af billederne foregår på baggrund af den process der nævnes i [53] på side 43.

JIT-kompileret afkodning af billeder

Da Python er et fortolket højniveau sprog er det ikke muligt at opnå samme afviklingshastighed som ved afvikling af C eller andre maskinnære sprog. I forbindelse med afkodningen af video fra AR.Dronen anvendes biblioteket Psyco, [61], til at øge afviklingshastigheden. Psyco er en form for JIT compiler, hvormed umodificeret Pythonkode afvikles hurtigere. Til sammenligning, vil almindelig afviklet Pythonkode til afkodning af 100 billeder fra AR.Dronens billedstrøm tage omkring 37.569 sekunder, men hvis det samme Python-kode afvikles med hjælp fra Psyco er afviklingstiden kun 7.075 sekunder. Ved brug af Psyco kan der altså hentes og afkodes mere end 5 gange så mange billeder. Psyco er lavet til 32 bit maskiner og virker fint der, men desværre har udvikleren af Psyco valgt ikke at lave en 64-bit version af biblioteket.

Videoformatet der sendes fra AR.Dronen minder meget om JPEG-formatet og forfatteren til den algoritme vi har tilpasset, spekulerer på forumsiden, [74], om det ville være muligt at anvende en decideret JPEG decoder til afkodningen. Dette kunne gøre det muligt at anvende et optimeret C bibliotek til formålet for derved at øge afkodningshastigheden og samtidig eliminere afhængigheden af Psyco.

WIFI-modtager

Som de andre receivere modtager WIFIReceiveren en datastrøm fra AR.Dronen. WIFI-strømmen er dog en tilføjelse vi har lavet og formatet der skal afkodes er blot en tekststreng bestående af en MAC-adresse og en signalstyrke. Der modtages således en pakke for hver pakke det indlejrede WIFI-sensorprogram (beskrevet i sektion 2.5.1) modtager. WIFI-receiveren skiller sig ud fra de andre receivere, ved at placere de modtagne adresse/signal-par i en liste og holde denne liste opdateret med de nyeste signalværdier for hver adresse. For hver adresse gemmes desuden de sidste 20 modtagne værdier og deres gennemsnit, varians og middelværdi. Når WIFIReceiverens `get_data` metode kaldes, er det denne liste der returneres.

Virtuelle sensorer

De virtuelle sensorer er en sen tilføjelse til platformen og er, som tidligere nævnt, inspireret af sensorhierakierne beskrevet i [40] og [29], hvor sensorer opererer på forskellige abstraktionsniveauer og derfor leverer sensor output på forskellige abstraktionsniveauer. I modsætning til receiverne, modtager de virtuelle sensorer ikke data direkte fra AR.Dronen, men derimod netop fra receiverne. De virtuelle sensorer leverer således information om AR.Dronens omgivelser på et højere abstraktionsniveau, end de rå værdier fra receiver-strømmene.

Den ene virtuelle sensor, PositionSilhouetDetector, [25], bearbejder kontinuerligt videodata for at genkende henholdsvis markører på jorden og silhuetter umiddelbart foran AR.Dronen. For at kunne detektere både markører på gulvet og silhuetter er det nødvendigt for sensoren hele tiden at skifte mellem bund- og frontkamera. Dette giver imidlertid problemer for AR.Dronens styreprogram der ved konstante kameraskift løbende øger sit hukommelsesforbrug indtil Linuxkernen lukker styreprogrammet ned. Målinger viser, at AR.Dronens program kun kan køre mellem ti og tolv minutter, såfremt der samtidig skiftes kamera med cirka 5 millisekunders interval. Selvom om dette problem ikke beskrives direkte nogen steder, antydes det dog af en udvikler på Parrots forum, at denne funktionalitet ikke er ment anvendt til hurtige kameraskift i software og at der derfor er stor sandsynlighed for, at metoden ikke virker pålideligt, [51].

Den anden virtuelle sensor, DistanceTracker, [25], benytter hastighedsværdierne fra navdatastrømmen. Sensoren udfører løbende integration over hastighederne i x og y-retning. Dette giver et estimat over den distance AR.Dronen har tilbagelagt. Se sektion 5.5, for resultaterne af de eksperimenter der er udført med DistanceTrackeren.

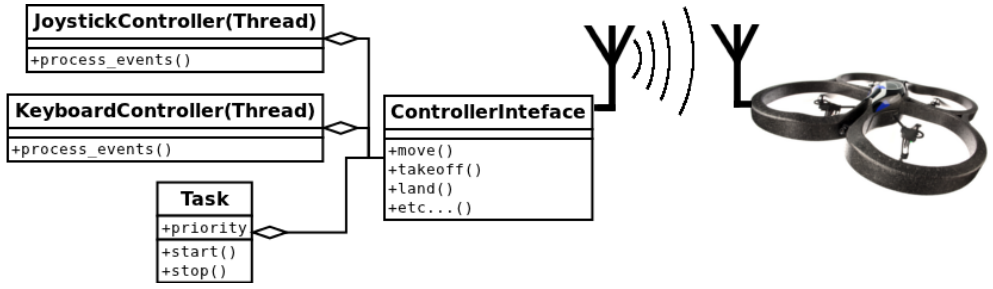
3.2.2 Kontrolinterface til fjernstyring af AR.Dronen

Ved hjælp af AT-kommandoerne definerer AR.Dronen et AT-interface til enheder der er indenfor rækkevidde af dens trådløse netværk. For at gøre dette AT-interface mere praktisk anvendeligt, er der, med basis i arbejdet i [75], udviklet en Python-wrapper, til AT-interfacet.

Kontrolinterfacet er implementeret som en tråd, der løbende sender kommandoer med de aktuelle kontrolværdier til AR.Dronen. Dette betyder, at platformen ikke afhænger af at alle kontrolpakker når deres destination, hvil-

ket er praktisk, da UDP-protokollen netop ikke garanterer at en pakke når sin destination. Ved at sende al kontrolkommunikation gennem kontrolinterfacet, skal der ikke tænkes på detaljer som sekvensnumre og watchdog-beskeder, da disse håndteres transparent af systemet.

Kontrolinterfacet (implementeret i ControllerInterface-klassen, [14]) definerer en række metoder til at fjernstyre AR.Dronens opførsel og bevægelser. Al styring af AR.Dronen fra platformens tråde, går gennem den samme ene instans af kontrolinterfacet se figur 3.7. Udover at virke i platformen, kan interfacet til simple formål også anvendes alene, direkte fra Pythons interpreter (som beskrevet i sektion 3.1.1).

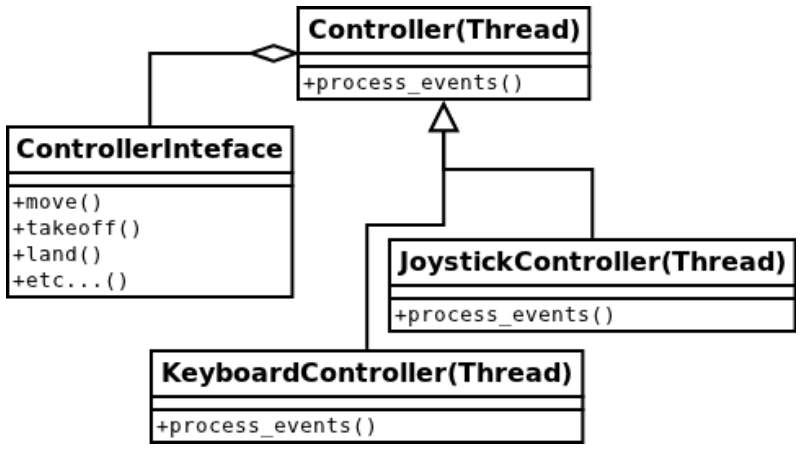


Figur 3.7: Singeltonklassen ControllerInterface til fjernstyring af AR.Dronen.

3.2.3 Controllers

Selvom det er muligt, er det ikke praktisk anvendeligt at brugeren skal afvikle scripts, for hver ønsket bevægelse (som det der er beskrevet i eksemplet i figur 3.2). I stedet er der implementeret to controllere til at tage mod input fra brugeren: En keyboardcontroller der lytter efter input fra tastaturet og en joystickcontroller der er tilknyttet en fysisk Xbox360-controller (se figur 3.9).

Platformen kan startes med begge controllere på samme tid, men det er også muligt at udelade en eller begge. Under udviklingen af platformen er begge controllere oftest anvendt sideløbende, da de komplimenterer hinanden. Man kan som sagt anvende platformen helt uden controllere og udelukkende anvende taskmanageren til at kontrollere AR.Dronen, men skulle der i det tilfælde ske noget uforudset under udførslen af en task, vil der ikke være nogen mulighed for at bringe AR.Dronen manuelt (og sikkert) ned.



Figur 3.8: Klassehieraki for klasser der nedarver fra Controller-klassen. Subklasser af Controller er kendetegnet ved at have adgang til styring af AR.Dronen gennem kontrolinterfacet.

Begge controllere nedarver fra en basecontrollerklasse: `Controller` (se figur 3.8). Denne er implementeret som en tråd der med et givent tidsinterval kalder en kontrolmetode (`process_events`). Den simple opbygning betyder, at hvis man ønsker at implementere en anden type controller, er det nok at nedarve Controller og implementere `process_events`-metoden. Da Python behandler metoder som førsteklasses objekter, er det let at lade en controller skifte mellem forskellige tilstande. Dette gøres ved at kalde `set_control_method` med en ny kontrolmetode som argument.

Joystickcontroller

Platformen understøtter XBox360 controlleren gennem den implementerede Joystickcontroller. XBox360 controlleren anvendes istedet for en iPhone eller iPad, til at fjernstyre AR.Drone manuelt i forbindelse med testflyvninger. Der var flere praktiske grunde til at implementere en alternativ fjernstyring. Blandt andet var der et behov for at kunne interagere både med platformen og AR.Dronen på samme tid, mens der til stadighed var øjne på AR.Dronen. Desuden blev iPhonestyringen hverken fundet tilstrækkelig præcis eller særlig intuitiv. Xbox360 controlleren derimod omtales som en af de bedst designede controllere og har endda fundet anvendelse af militære styrker i USA og

storbritannien, [95], [60], [94]. Fjernstyringsdelen er meget simpel og fungerer ved at oversætte værdier fra Xbox360 controlleren (som gives af pygames joystickmodul, [64]) og så sende disse til kontrolinterfacet. Xbox360 Joystickcontrolleren bruges udover manuel fjernstyring af AR.Dronen også til at starte tasks (sektion 3.3). Således kan piloten manuelt bringe AR.Dronen i en ønsket position og så herefter starte afviklingen af en task. På figur 3.9 ses den anvendte standardopsætning af Xbox360 controlleren.

Figur 3.9: Standardopsætning af Xbox360 controlleren

Keyboardcontroller

Hvis man ønsker at starte platformen uden grafisk brugerflade (og dermed uden tilhørende funktionalitet til at håndtere input fra brugeren), er det praktisk stadig at kunne anvende tastaturet som input (evt. i kombination med Xbox360 controlleren). Der er derfor implementeret en simpel keybo-

ardcontroller, der læser input fra terminalen. Keyboardet anvendes ikke til direkte styring af AR.Dronen (da Xbox360 controlleren er mere intuitiv og ligeledes tilgængelig), men i stedet til ting såsom: At få udskrevet nuværende batteriniveau ('b'), at skifte kamera ('z'), at få udskrevet aktive tasks ('c'), at få udskrevet en oversigt over tråde i programmet ('t'), samt at starte tasks ('1', '2' og '3').

3.2.4 Testdevice

Dette modul muliggør gennemførelsen af simple simuleringseksperimenter i et test miljø. Tanken bag modulet er, at man skal kunne optage, ikke bare video, men alle sensoroutput fra en flyvning med AR.Dronen, for så senere at kunne afspille hele flyvningen igen. Selve afspilningen foregår næsten transparent for resten af platformen. Receiverne skal stadig initiere kommunikationen (nu blot med testdevicet og ikke den reelle AR.Drone). Forbindelsen skal stadig holdes i live og data der modtages skal stadig afkodes. De eneste synlige forskelle der er mellem test og live-kørsler er timing og hastighed, den første fordi det ikke har været en prioritet at afsende testdataen i de intervaller de blev modtaget i og den sidste fordi samme computer nu står både for afsendelse, modtagelse og afkodning. Testdevicemodulet har ikke noget at gøre med optagelsen af sensoroutputtet. Dette foregår ved hjælp af sensordisplay-værktøjet, beskrevet i sektion 4.1.

3.2.5 Taskmanager

Taskmanagerklassen startede ud som en almindelig controller, men er endt som en mere selvstændig del af platformen. Taskmanageren skiller sig ud ved ikke, i modsætning til joystick- og keyboardcontrollerne, at arbejde direkte med kontrolinterfacet eller med output fra receiverne. Taskmanagerens opgave er at starte task (gennemgåes i sektion 3.3) og stoppe dem igen, samt at facilitere beslutningen, om hvilken task der har lov til at bevæge AR.Dronen.

3.3 Tasks

For at indkapsle forskellige bevægemønstre og for at kunne koordinere en parallel afvikling af disse, er der implementeret et hierakisk opbygget tasksystem. Enhederne i dette system kaldes tasks og ikke behaviors, der ellers både bruges i artiklen 'A Modular Hierarchical Behavior-Based Architecture', [40], og i Lejos' subsumption-pakke, [10]. Grunden til dette er, at flere af de implementerede tasks er forholdsvis kortlivede (for eksempel take-off og landtasks) og derfor mere minder om en opgave der skal forberedes, udføres og afsluttes. Dette ses i modsætning til en behavior, som kan opfattes som en længerevarende opførsel. Der er også implementeret højniveau bevægemønstre, der minder mere om traditionelle behaviors (for eksempel FollowTourTask som gennemgåes i sektion 3.3.1), men for at være konsistente, kaldes disse også tasks.

De første udgaver af tasksystemet indbefattede tasks der indsamlede og behandlede data fra AR.Dronens sensorer. De behandlede data blev herefter stillet til rådighed for andre tasks, gennem en delt kontekstliste. I de endelige udgaver er disse tasks udskilt som virtuelle sensorer. Tasks skal således ikke behandle sensordata, men kun reagere og formidle bevægelse på baggrund af højniveau-sensordata.

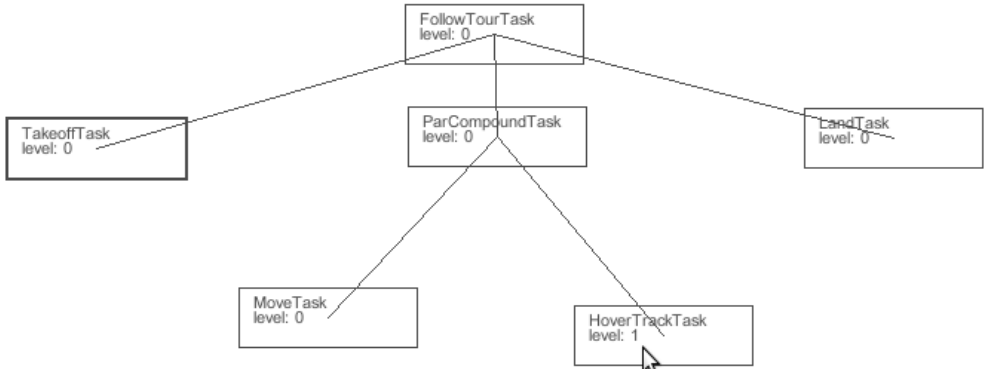
3.3.1 Task typer

Der er to hoved-tasktyper, simple og compoundtasks. Simple tasks nedarver fra basetaskklassen Task. De repræsenterer oftest en simpel bevægelse, men kan være vilkårligt komplekse. For eksempel implementerer HoverTrackTask, blandt andet, en PID-controller.

Compoundtasks nedarver fra CompoundTask-klassen og indeholder en liste af subtasks. Compoundtasks opdeles yderligere i to typer, sekventielle og parallelle (SeqCompoundTask og ParCompoundTask). Sekventielle compoundtasks starter deres subtasks en efter en og venter med at starte en ny subtask, indtil den forrige er afsluttet. En sekventiel compoundtask stopper, når den sidste subtask er afsluttet. Parallelle compoundtasks starter alle deres subtasks på en gang og stopper, når den sidste subtask er afsluttet. Herunder gives eksempler på henholdsvis én compoundtask og to simple tasks.

FollowTourTask

Et eksempel på en compoundtask er FollowTourTasken, hvis tasktræ kan ses på figur 3.10. FollowTourTask nedarver fra SeqCompoundTask og indeholder, udover takeoff og land-tasks, en parallel compoundtask der kan veksle mellem en MoveTask og en HoverTrackTask (det vil sige, at der veksles mellem bevægelse og fastholdelse af en erkendt position). Selve FollowTourTask enheden holder, ved hjælp af et map-objekt (se sektion 4.2), styr på den tour der følges og sørger for at opdatere HoverTrackTasken med en ny målposition, når det er nødvendigt. Hvis map-objektet anskues som en relationel graf, hvor alle knuder er forbundne, minder den implementerede opførsel meget om Kuipers og Byuns metode der beskrives i [42]. Specielt lægges mærke til, at der i begge systemer udføres fejlkorrigering, hver gang en ny position detekteres. Under afviklingen af en FollowTourTask, sker dette ved at HoverTrackTasken bringer AR.Dronen indenfor en maksimum afstand af den præcise position, før den sendes videre.



Figur 3.10: Opbygningen af en FollowTourTask.

HoverTrackTask

HoverTrackTasken, som er en simpel task, implementerer en PID-controller,[81], der kan holde AR.Dronen over en erkendt position.

Når en markør (se 5.4(c) side 70) er synlig for AR.Dronens bundkamera, kan markørens billedkoordinat findes via af PositionSilhouetDetectoren (se sektion 3.2.1). Med dette billedkoordinat udregnes en error-værdi i x- og

y-retningen, i forhold til centrum af billedet. For at kunne bruge errorværdierne, skal de konverteres fra pixels til millimeter og der skal tages højde for AR.Dronens øjeblikkelige hældning. Denne konvertering og korrigering foregår, efter den metode der beskrives og anvendes af Michaël Ludmann og Guillaume Depoyant i deres speciale LUDEP, [27], [28]. Efter udregningen af errorværdierne, anvendes de som i en standard PID-controller, til at bestemme de endelige værdier der sendes til kontrolinterfacet, for at bevæge AR.Dronen. PID-funktionaliteten er implementeret i HoverTrackTaskens update-metode, [21].

I forbindelse med deltagelse i en FollowTourTask, kan HoverTrackTasken tildeles information om den specifikke position der skal trackes. Den kan således lade andre tasks bevæge AR.Dronen, hvis den rigtige position ikke er fundet. HoverTrackTasken er, i forbindelse med en FollowTourTask, også ansvarlig for at dreje AR.Dronen i retning af den næste position, mens nuværende position fastholdes.

AvoidTask

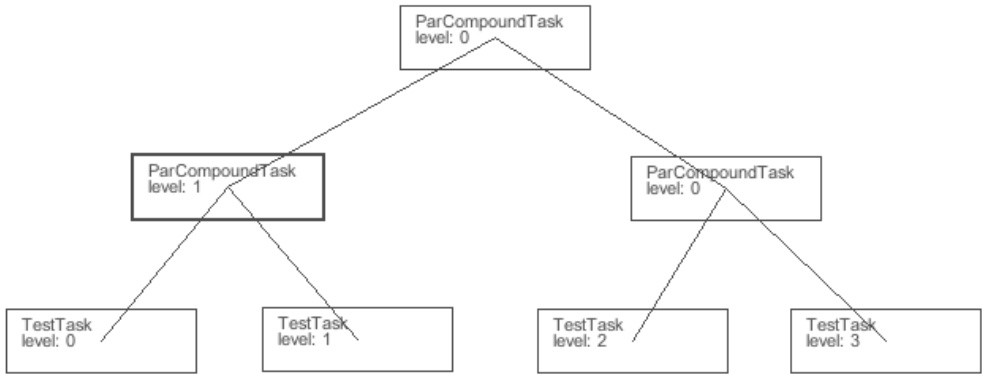
AvoidTasken er også et eksempel på en simpel task. Den anvender PositionSilhouetDetectoren (se sektion 3.2.1), til at detektere personer umiddelbart foran AR.Dronen. Hvis en person detekteres, vil AvoidTasken bevæge AR.Dronen op, indtil personen ikke længere er synlig for AR.Dronens frontkamera. Den forholdsvis simple task udnytter AR.Dronens muligheder for at bevæge sig i rummet. Grunden til at AR.Dronen bevæges op og ikke til siden eller bagud er, at AR.Dronen ikke kan orientere sig i disse retninger uden at rotere. Risikoen forbundet med at bevæge sig op er lille (da loftshøjden er kendt), i forhold til risikoen ved at bevæge sig sidelæns eller baglæns i ikke-erkendt luftrum. Desuden er det let for de andre tasks, at fortsætte deres udførelse efter AvoidTasken stopper, idet AR.Dronen bibeholder samme orientering mens AvoidTasken er aktiv. AvoidTasken kombineres let med andre tasks, f.eks. en FollowTourTask eller en simpel MoveTask, ved hjælp af taskcreator-værktøjet (sektion 4.3).

3.3.2 Task hieraki

Ved hjælp af de ovenfor beskrevne tasktyper, og specialiseringer af disse, kan der dannes en task-træstruktur med parents og children. Strukturen minder om den beskrevet i [40]. Det grundliggende formål med denne træstruktur er

at danne et task-hieraki, for at understøtte dette hieraki, tildeles hver task en priotitet (level) i forhold til andre tasks med samme parent. Taskmanageren skal ved hjælp af task-hierakiet sørge for, at der aldrig er mere end én task der bevæger AR.Dronen. En given task kan få tilladelse til at bevæge AR.Dronen, på to betingelser: 1) Ingen task med højere level og samme parent ønsker at bevæge AR.Dronen og 2) Taskens parent har tilladelse til at flytte AR.Dronen.

På figur 3.11 ses et eksempel på et tasktræ konstrueret med taskcreator-værktøjet (sektion 4.3). Under antagelse af, at hver TestTask ønsker at udføre samme antal bevægelser med AR.Dronen og ønsker at udføre disse bevægelser uden pause, vil TestTask med level 1 slutte først og TestTask med level 2 slutte sidst.



Figur 3.11: Eksempel på et taskhieraki. Hver task er annoteret med deres level. Dette tal beskriver taskens prioritet i forhold til andre tasks med samme parent.

3.3.3 Task implementationer

Der er implementeret to forskellige udgaver af task-systemet (taskmanageren og alle tasks), begge implementerer den opførsel og struktur der er beskrevet ovenfor.

Den første, trådbaserede, implementation er inspireret af Lejos' subsumption-pakke, [10]. Lejos er en firmwareerstatning til LEGO Mindstorms RCX- og NXT-platform, der tillader afviklingen af Java på disse platforme, [9]. Den

anden, ikke-trådbaserede, implementation er primært inspireret af artiklen 'A Modular Hierarchical Behavior-Based Architecture', [40].

Trådbaseret

I den første trådbaserede løsning er hver task implementeret som en individuel tråd. Alle tasks kører samtidig. Når en task ønsker at bevæge AR.Dronen, får den tilladelse til dette ved at spørge sin parent (som muligvis skal spørge videre op i træet).

Denne tilgang minder meget om arkitekturen der anvendes af Lejos' subsumption pakke, [10]. Taskmanageren vil, som tasktræets rod, have samme rolle som Arbitratoren i Lejos system. Denne arkitektur har primært et 'bottom-up'-flow, da det er de individuelle tasks der initierer bevægelsen af AR.Dronen. En ulempe ved designet er dog, at flowet ikke kun går mod taskmanageren. En del kommunikation er nødvendig for sikre, at lavere-level tasks bliver deaktiveret, når en højere-level task tager kontrollen. Erfaringen med denne arkitektur er desuden, at den på grund af overheadet ved at have en tråd per task, ikke skalerer særligt godt.

Ikke-trådbaseret

Tilgangen i 'A Modular Hierarchical Behavior-Based Architecture', [40], er at en aktiveret behavior skal bestemme hvilke af sine subbehaviors der skal aktiveres. I artiklen er dette praktisk, da deres robot kan udføre forskellige bevægelser på samme tid. Tilgangen sikrer, at det bedste sæt af parallelle behaviors kan bestemmes og herefter aktiveres.

I den ikke-trådbaserede tilgang forsimples ovennævnte princip. En Compoundtask skal på opfordring aktivere *den* bedst egnede subtask, da der ikke er mere end én aktiv tasks samtidig[1]. Den bedst egnede subtask defineres altid, som den med det højeste level og et ønske om at bevæge AR.Dronen.

Den eneste tråd i denne implementation er taskmanageren. Denne lader kontinuerligt den bedst egnede task bevæge AR.Dronen. I Praksis sker dette ved at taskmanageren vedligeholder en sorteret liste over de aktive tasks. Under hver iteration kalder taskmanageren `domove`-metoden på den første task i listen (tasken med det højeste level). Kaldet til `domove`-metoden propageres rekursivt ned gennem tasktræet og returnerer True hvis AR.Dronen blev bevæget og False hvis ingen task havde interesse i at bevæge den. Hvis metoden

[1]De bevægelser der er til rådighed kan ikke udføres uden indflydelse på hinanden

returner False forsøges med den næste task i listen, hvis der returneres True afbrydes iterationen og der startes forfra. Sammenlignet med den trådede udgave, er flowet 'top-down', da bevægelsen altid initieres af taskmanageren.

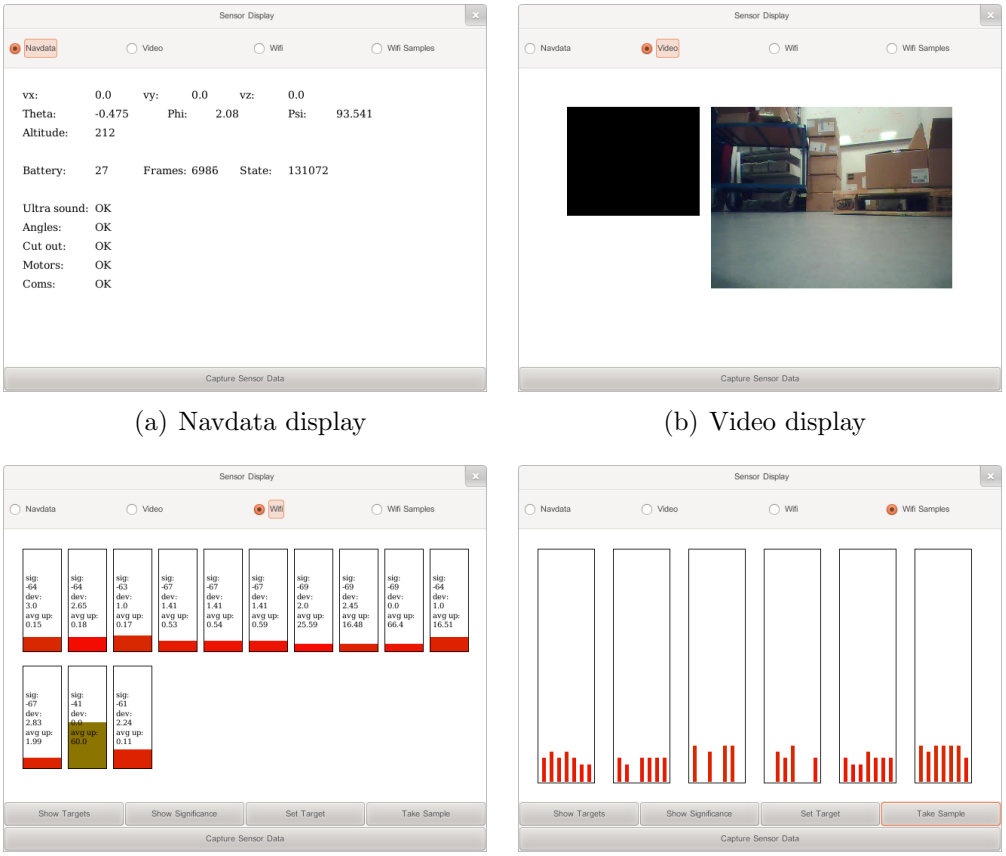
Den ikke-trådbaserede tilgang virker mere velegnet til en træbaseret taskstruktur. Kommunikationen mellem tasks er simplere og overheadet pga. de mange tråde undgåes.

Kapitel 4

Klientværktøjer

I dette kapitel gennemgåes tre værktøjer, som er udviklet i forbindelse med specialearbejdet. Der er tale om et program til at præsentere og visualisere datastrømmene fra AR.Drone (Sensordisplay), et program til at bygge de map-objekter der anvendes af FollowTourTasken (Mapcreator) og et program til at instantiere, kombinere og afvikle Tasks (Taskcreator). Alle tre værktøjer anvender et drone-objekt som vist på figur 3.5 og kan startes i henholdsvis normal- og testtilstand (se figur 4.2, 4.4 og 4.6). Hvis brugeren vælger at starte et værktøj i testtilstand, vil drone-objektet oprette forbindelse til et testdevice (beskrevet i sektion 3.2.4) istedet for AR.Dronen.

4.1 Sensor display



(a) Navdata display

(b) Video display

(c) WIFI display

(d) WIFI samples display

Figur 4.1: Sensordisplay værktøjet har fire præsentationstilstande. En til hver af de tre sensorstrømme(navdata, video og WIFI), samt en til at sammenligne WIFI-signalstyrker.

Sensordisplay-værktøjet er implementeret i sensordisplay.py, [22]. Programmet anvender et drone-objekt til at få adgang til AR.Dronens sensorstrømme. Brugeren vælger hvilken sensorstrøm der ønskes præsenteret, ved at trykke på en af de fire radio buttons (Navdata, Video, WIFI eller WIFI samples),

som aktiverer hver deres præsentationstilstand. Hvis brugeren ønsker at indsamle datapakker til anvendelse i testtilstand, kan dette gøres ved at trykke på knappen 'Capture Sensor Data'. De rå datapakker vil så blive gemt ved programmets afslutning og kan herefter bruges af det implementerede testdevice, se sektion 3.2.4.

Når navdatastrømmen præsenteres, ser brugeren et skærmbillede som det på figur 4.1(a). På dette billede ses værdier såsom højde, hastighed, samt vinklerne ϕ, θ og ψ. Desuden vises oplysninger om AR.Dronens umiddelbare tilstand, såsom motortilstand, ultralydstilstand og om hvorvidt førnævnte Tait-Bryan-vinkler er indenfor de tilladte grænser.

Når videostrømmen præsenteres, vises et billede som det på figur 4.1(b). Afhængig af hvilken videostrøm der modtages, eller om der løbende skiftes videostrøm, vil en af billedrammerne muligvis være sort. Hvis der løbende skiftes videostrøm, vises det nyeste billede fra hver videostrøm i de to billedrammer.

Hvis WIFI-sensorprogrammet er startet på AR.Dronen (beskrevet i sektion 2.5.1) og WIFI-strømmen præsenteres, vil brugeren blive præsenteret for et billede som det på figur 4.1(c). Her ses en søjle der repræsenterer den sidst kendte signalstyrke, for hver erkendt WIFI-kilde. I denne præsentationstilstand er der desuden mulighed for at indsamle signalstyrkesamples og sætte en mål-signalstyrkesample.

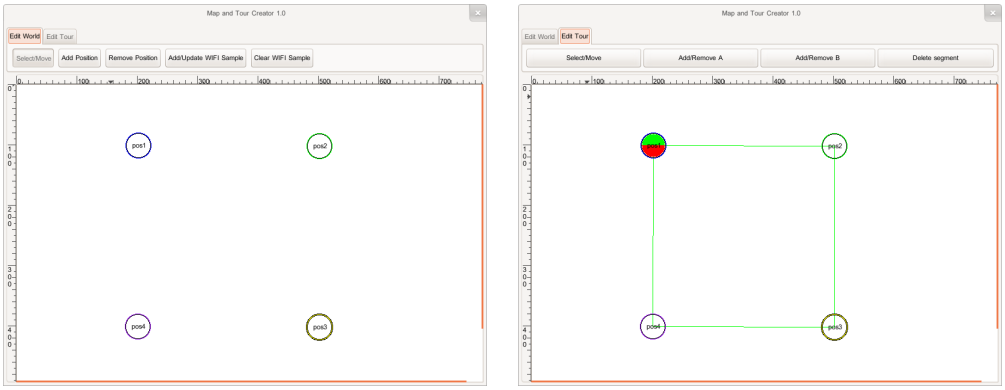
Under udviklingen af WIFI-fingerprintingalgoritmen beskrevet i 5.1.1, har det været nødvendigt, at kunne følge udviklingen i WIFI-signalstyrker mellem forskellige lokationer. Dette kan gøres i den fjerde og sidste præsentationstilstand. En signalstyrkesample registreres ved at trykke på knappen 'Take Sample'. De indsamlede signalstyrkesamples kan herefter sammenlignes. Skærmbilledet, figur 4.1(d), viser en ramme for hver erkendt WIFI-kilde. I rammerne angives WIFI-kildens signalstyrke for hver registreret signalstyrkesample med en søjle. Søjlerne viser således WIFI-kildens signalstyrkeudvikling.

Sensordisplay-værktøjet startes fra en linuxkommandolinie, som vist på figur 4.2

```
$ ./sensordisplay.py

  eller hvis programmet ønskes anvendt i testtilstand

$ ./sensordisplay.py -t
```

Figur 4.2: Afvikling af sensordisplay-værktøjet fra en linuxkommandolinie.

4.2 Mapcreator



(a) World editor (b) Tour editor

Figur 4.3: Mapcreator-værktøjet, som anvendes til at redigere 2D kort, har to redigeringstilstande, world(a) og tour(b).

Mapcreator er et værktøj til at oprette og redigere map-objekter. Map-objekter instantieres fra klassen PosMap (placeret i map.py, [19]) og bruges af FollowTourTasks til at navigere efter. Mapcreator har to redigeringstilstande, world og tour, som kan vælges ved at trykke på det tilsvarende faneblad øverst i billedet. De to tilstande ses på figur 4.3.

I world-redigeringstilstanden kan brugeren tilføje, flytte eller fjerne punkter på et 2D kort. Meningen med punkterne er, at de skal matche en visuel markør i den fysiske opstilling, som vist på figur 5.6. Internt i map-objektet vedligeholdes en liste med vinklerne mellem alle punkter. Når punkter tilføjes, flyttes og slettes opdateres denne liste også automatisk. Udover punkternes fysiske position i forhold til hinanden, kan man også tilføje WIFI-signalstyrkemålinger (WIFI-fingerprint) til punkterne ved at trykke på 'Add-/update WIFI sample'-knappen og herefter trykke med musen på det punkt der ønskes markeret. Hvis WIFI-signalstyrkemålingen ønskes slettet, trykkes først på 'clear WIFI sample' og derefter på punktet. World-skærmbilledet kan ses på figur 4.3(a).

I tour-redigeringstilstanden kan brugeren ikke længere redigere punkterne, til gengæld kan disse forbindes til en tour. Dette gøres ved at holde 'ctrl' nede og så trække en linie mellem to punkter med musen. Det er kun muligt, at forbinde til det første eller sidste punkt i en eksisterende tour. Det er muligt at fjerne liniesegmenter fra en tour ved at markere segmentet med musen og herefter trykke på knappen 'Delete Segment'. Ved at markere et liniesegment og trykke enten 'Add/remove A' eller 'Add/remove B' kan man markere, at dette liniesegment tilhører en specifik type. Denne annotering af liniesegmenter anvendes endnu ikke til noget. På sigt er det dog meningen, at FollowTourTasken skal kunne vælge en bevæge-algoritme på baggrund af denne information. Tour-skærmbilledet kan ses på figur 4.3(b).

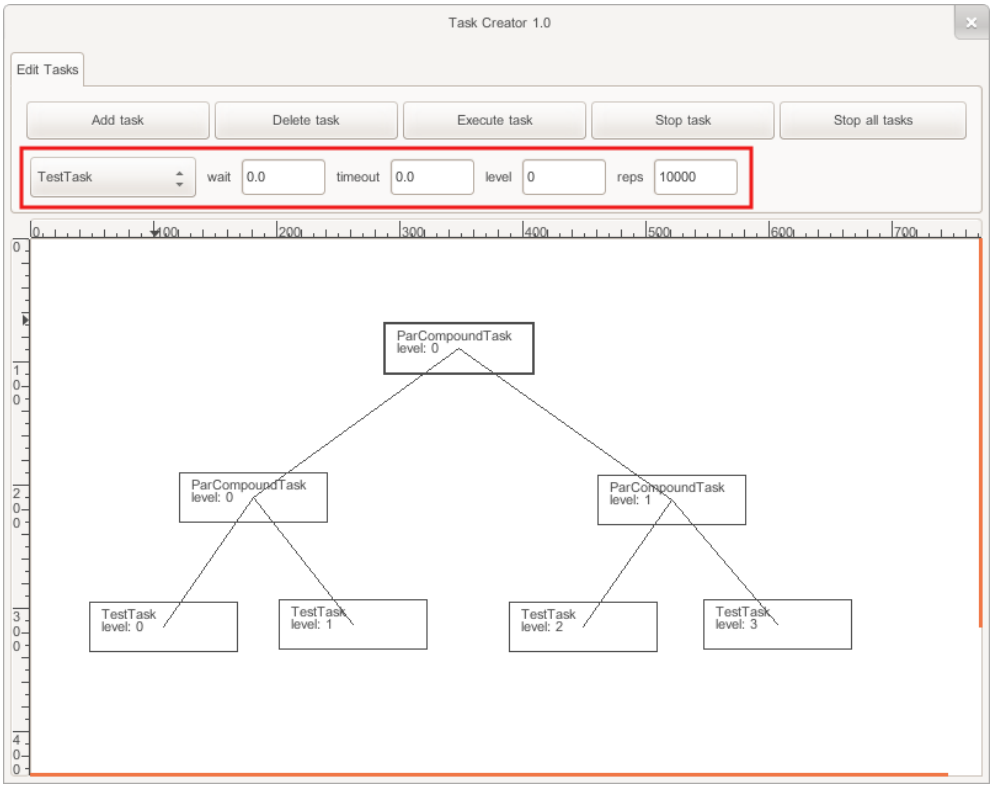
Når programmet sluttes, gemmes map-objektet i filen map.data . I map-klassens constructor søges efter denne fil og hvis den findes, indlæses map-objektets state fra denne.

Som sensordisplay-værktøjet, startes mapcreator-værktøjet fra en linuxkommandolinie, som vist på figur 4.4.

```
$ ./mapcreator.py

  eller hvis programmet ønskes anvendt i testtilstand

$ ./mapcreator.py -t
```

Figur 4.4: Afvikling af mapcreator-værktøjet fra en linuxkommandolinie.

4.3 Taskcreator



Figur 4.5: Taskcreator-skærmbillede. Dropdown-listen med tilgængelige Tasks og parameterinputfelterne til den valgte Task, er markeret med rødt.

Taskcreator-værktøjet bruges til at instantiere, kombinere og afvikle tasks (beskrevet i sektion 3.3). Brugeren kan vælge mellem de tilgængelige tasks ved at trykke på dropdown-listen, som ses yderst til venstre i den røde firkant på figur 4.5. Herefter kan standardparametrene for den valgte task tilpasses i inputfelterne, bl.a. kan taskens level angives. En tasks level beskriver dens prioritet i forhold til andre tasks med samme parent. Den valgte task instantieres ved at trykke 'Add task' og kan fjernes igen, ved at markere den med musen og trykke 'Delete task'. Tasks der nedarver fra klassen Compo-

undTask (ParCompoundTask, SeqCompoundTask og FollowTourTask) kan indeholde subtasks. Subtasks tilføjes ved at holde 'ctrl' nede og trække en linie fra compoundtasken til den ønskede subtask med musen. Hvis brugeren ønsker afkoble en subtask fra en compoundtask, gøres det ved at gentage ovennævnte procedure. Figur 4.5 viser et opbygget tasktræ bestående af tre ParCompundTasks og fire TestTasks.

Når brugeren har instantieret og kombineret de ønskede tasks, kan disse afvikles og dermed udføres af AR.Dronen. Dette gøres ved at markere den ønskede task og trykke på 'Execute task'. Afviklingen af en task eller alle tasks kan stoppes ved at trykke på henholdsvis 'Stop task' eller 'Stop all tasks'.

Taskcreatoren startes som de andre værktøjer fra en linuxkommandolinie, se figur 4.6

```
$ ./taskcreator.py

  eller hvis programmet ønskes anvendt i testtilstand

$ ./taskcreator.py -t
```

Figur 4.6: Afvikling af taskcreator-værktøjet fra en linuxkommandolinie.

Kapitel 5

Eksperimenter

En vigtig forudsætning for at en robot kan navigere i et miljø er, at den er i stand til at lokalisere sig selv i dette miljø. Der er derfor blevet ekperimenteret med forskellige former for lokalisering. Mere specifikt er der arbejdet med lokalisering ved hjælp af WIFI-signalstyrker og lokalisering ved hjælp af visuelle markører. I eksperimenterne er der gjort brug af et map-objekt med beskrivelser af lokationer (bestående af indsamlede WIFI-signalstyrker, markørbeskrivelse og koordinater) og en rutebeskrivelse bestående af en liste med lokationer. Map-objektet indeholder desuden en fortegnelse over vinkelretninger mellem alle lokationspar. Map-objektet oprettes let ved hjælp af Mapcreator-værktøjet beskrevet i sektion 4.2. For at teste de forskellige lokaliseringsmetoder anvendes HoverTrackTasks (se section 3.3.1) og FollowTourTasks (se section 3.3.1).

Udover lokalisering, er der også undersøgt andre muligheder for at udlede ekstra informationer om AR.Dronens miljø. F.eks. hvor langt AR.Dronen har bevæget sig, hvorvidt den befinder sig i en korridor og afstandsmåling til genstande foran den.

5.1 Lokalisering via WIFI-fingerprinting

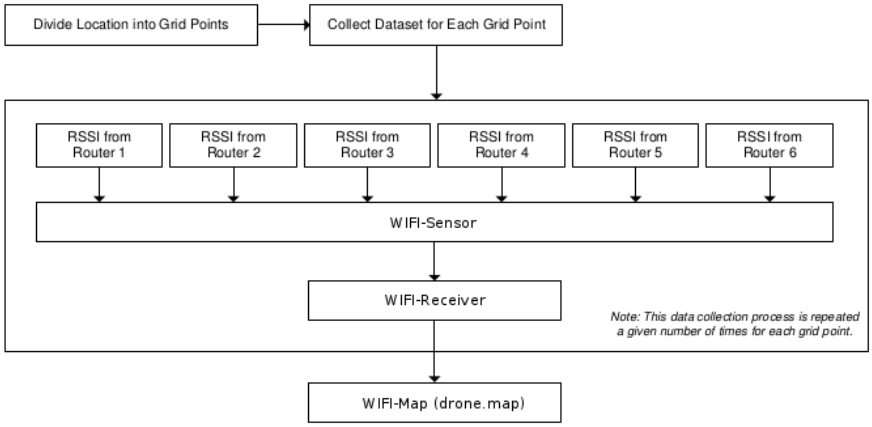
Som nævnt i sektion 2.5, er AR.Dronen blevet udvidet med en WIFI-sensor. Denne sender løbende information om de omkringliggende WIFI-kilders signalstyrker. I de følgende eksperimenter anvendes en teknik kaldet WIFI-fingerprinting. WIFI-fingerprinting fungerer i hovedtræk ved at der opbygges et WIFI-map under en initiel offline-fase, hvorefter livesignalstyrkemålinger

kan sammenlignes med WIFI-mappet. WIFI-mappet opbygges ved at foretage signalstykemålinger på et antal faste punkter. Generelt kan det siges, at jo mere offlinedata der indsamles, jo bedre fungerer lokaliseringen. WIFI-fingerprintingteknikken beskrives i [2].

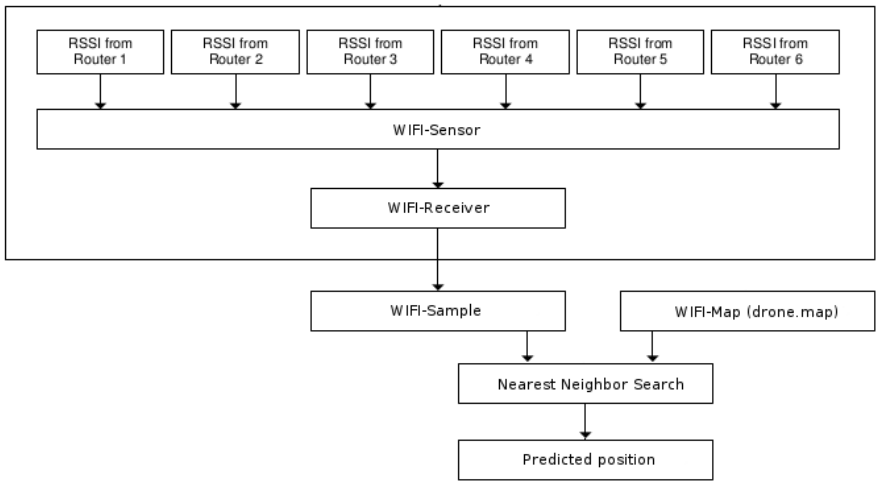
Under eksperimenterne anvendes kun den allerede tilstedeværende WIFI-infrastruktur i testmiljøet i Zuse-bygningen.

5.1.1 WIFI-fingerprinting-algoritmen

I de følgende eksperimenter anvendes en fingerprintingalgoritme, der udfører en simpel nearest-neighbor søgning i et WIFI-map. WIFI-mappet opbygges under offline-proceduren umiddelbart før eksperimentets udførelse og er en integreret del af det map-objekt der anvendes af FollowTourTasken. Et overblik over offline-proceduren kan ses på figur 5.1. Nearest-neighbor søgningen udføres ved at sammenligne hvert grid-punkt i WIFI-mappet med de nuværende signalstyrker. To sæt af signalstyrker sammenlignes ved først at finde en fællesmængde af WIFI-kilder i de to sæt og herefter udregne en afstand mellem de to sæt, baseret på denne fællesmængde. Lokaliseringsproceduren er illustreret på figur 5.2 Algoritmen er implementeret i virtualsensors.py, [25] og WIFI-mappet er implementeret som en del af map.py, [19].



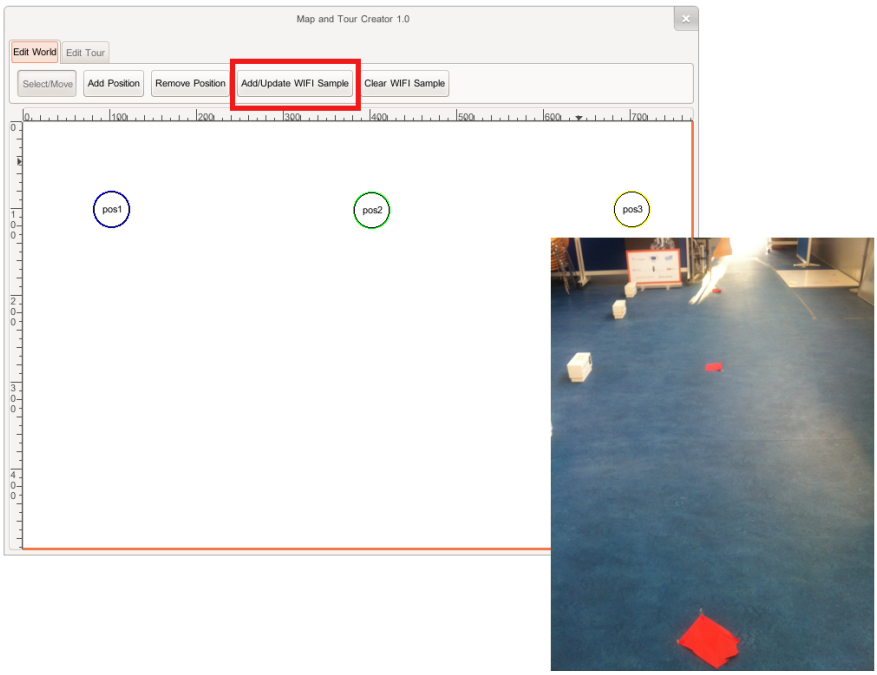
Figur 5.1: Diagrammet viser fingerprintingteknikkens offlinefase. WIFI-sensoren sender løbende signalstyrker til WIFI-receiveren, der stiller disse tilrådighed for det program der bruges til at opbygge WIFI-mappet (mapcreator.py). Figuren er tilpasset fra [59].



Figur 5.2: Diagrammet viser lokaliseringsfasen. Outputtet fra WIFI-receiveren sammenlignes med indholdet i WIFI-mappet og resultatet er den mest sandsynlige position. Figuren er tilpasset fra [59].

5.1.2 Første Eksperiment med WIFI

Opstillingen til dette eksperiment består af 3 visuelle markører med 3 meters mellemrum som vist på figur 5.3. Inden eksperimentet, udførtes en indsamling af WIFI-signalstyrker (beskrevet i sektion 5.1), til indsamlingen blev mapcreator-værktøjet (sektion 4.2) anvendt. Indsamlingen blev udført i en højde af 1 meter, med AR.Dronen placeret på et fast stativ.



Figur 5.3: Setup til første WIFI-eksperiment, knappen "Add/update WIFI sample"anvendes til at associere en WIFI-signalstyrkemåling med en lokation.

Efter den indledende fase, blev AR.Dronen skiftevis placeret over de tre markører og WIFI-fingerprinting-algoritmen afviklet 200 gange. Dette blev gentaget 3 gange for hver position. På tabel 5.1 ses detektionsraten for hver udførelse. Detektionsraten defineres som antal gentagelser (200) divideret med antal korrekte lokaliseringer.

Position	1. gentagelse	2. gentagelse	3. gentagelse
1	0.55	0.26	0.685
2	0.67	0.81	0.735
3	0.925	0.915	0.93

Tabel 5.1: Detektionsrater for WIFI-fingerprinting, 3 positioner med 3.0 meters afstand

Som det ses er resultaterne meget svingende. Med den implementerede fingerprinting algoritme og afstande mellem lokationerne på 3 meter eller derunder, kan en konsistent lokalisering ikke garanteres.

5.1.3 Andet Eksperiment med WIFI

Det andet eksperiment med WIFI-lokalisering er direkte afledt af de dårlige resultater i det første eksperiment. Det blev besluttet, at øge afstanden mellem punkterne til 6 meter, for herefter at gentage eksperimentet som beskrevet i det første eksperiment. Resultatet af dette eksperiment kan ses på tabel 5.2. De opnåede detektionsrater under denne gennemførsel er markant bedre, end under det første eksperiment.

Position	1. gentagelse	2. gentagelse	3. gentagelse
1	1.0	1.0	0.995
2	0.725	0.9	0.855
3	1.0	0.97	0.955

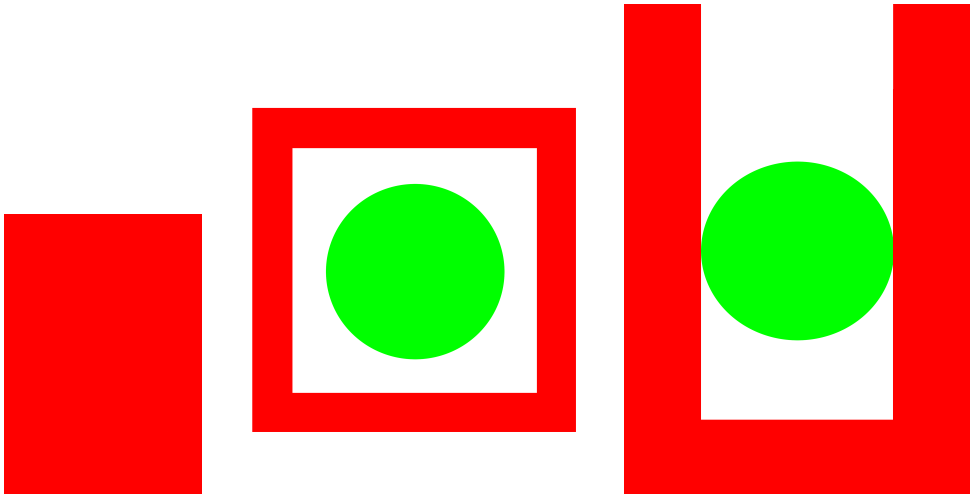
Tabel 5.2: Detektionsrater for WIFI-fingerprinting, 3 positioner med 6.0 meters afstand

5.1.4 Resultater

Med denne opstilling, kan man kun forvente en konsistent lokalisering, hvis lokationerne ligger med 6 meters afstand eller mere. Dette resultat gør, at

denne eksperimentrække ikke føres videre. WIFI-fingerprintingalgoritmen, eller en videreudvikling, kunne muligvis fungere som støtte til en primær visuel lokaliseringsmetode. Bedre resultater kunne muligvis være opnået hvis der var opstillet en WIFI-infrastruktur specifikt til eksperimentet.

5.2 Visuel lokalisering



(a) Simpel markør (b) semiavanceret markør (c) avanceret markør

Figur 5.4: Tre markører som er blevet testet i forbindelse med visuel lokalisering af AR.Dronen

5.2.1 Lokalisering via simple markører

Første eksperiment med simple markører

Det første forsøg med visuel lokalisering anvendte en simpel rød markør i A5 størrelse, se figur 5.4(a). Til eksperimentet var der implementeret en blobdetekteringsalgoritme i Python, [17], som for hver pixel vurderer om denne er rød nok til at være en del af markøren. Algoritmen antager at der i billedet kun findes eet rødt objekt. På grund af Pythons afviklingshastighed er de-

tektoren ikke videre hurtig, men til denne test, hvor detekterings-hastighed ikke var afgørende, fungerede den simple implementation udemærket.

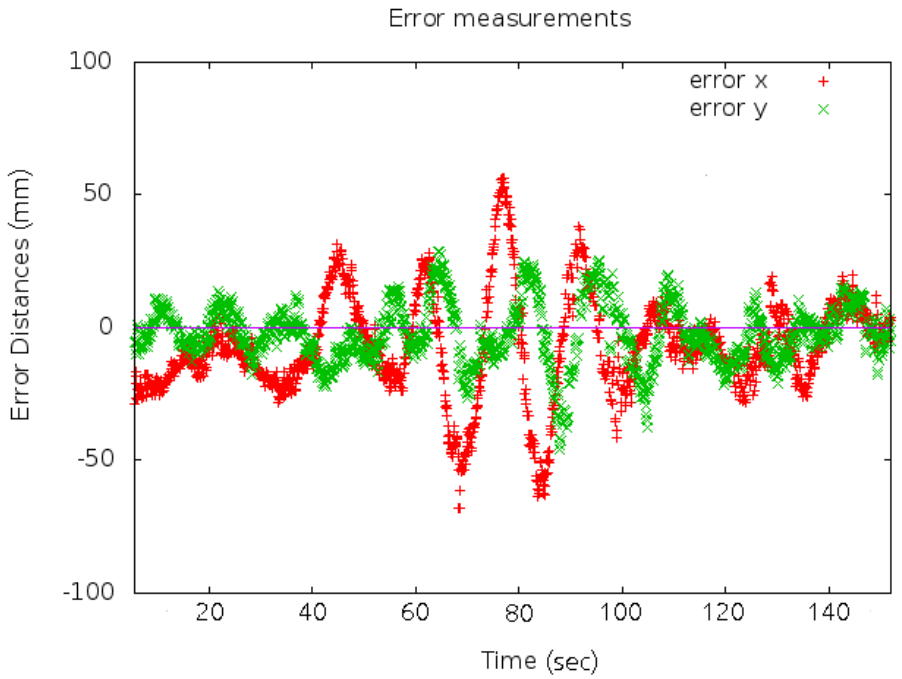
Eksperimentet udførtes ved at markøren blev placeret på gulvet og AR.Dronen placeret i position over markøren, hvorefter blobdetectionalgoritmen blev startet. Detektionsraten defineres som antal succesfulde detektioner delt med antal gentagelser af algoritmen. Algoritmen kørtes 200 gange i henholdsvis 0.5, 1.0, 1.5 og 2.0 meters højde og resultaterne kan ses i tabel 5.3 på side 78, sammen med resultaterne af de andre markør-tests.

Det første forsøg viste, at den simple markør, i højder fra 0.5 til 2.0 meter, altid kunne genkendes med AR.Dronen bundkamera.

Andet eksperiment med simple markører

Næste skridt var at anvende ovennævnte lokaliseringsmetode i forbindelse med afviklingen af en HoverTrackTask (sektion 3.3.1). Dette blev gjort, for at få en idé om metodens anvendelighed i en kontekst hvor effektivitet er afgørende.

Til dette forsøg anvendtes også kun én markør. Forsøget afvikledes ved at AR.Dronen blev fløjet i position over Markøren, hvorefter HoverTrackTasken blev aktiveret. Det viste sig her tydeligt, at blobdetektoren ikke var effektiv nok. Der blev ikke opdateret hurtigt nok og som resultat nåede AR.Dronen ofte at bevæge sig væk fra markøren. Af denne grund blev HoverTrackTasken reimplementeret således, at den anvendte den implementerede blobdetektor til at opdage markøren og herefter anvendte OpenCVs opticalflow-algoritme (`calcOpticalFlowPyrLK`, [44]) til at tracke markøren i videostrømmen. Afviklingen af `calcOpticalFlowPyrLK` er omtrent 10 gange hurtigere end blobdetectoren. Ved gentagelse af forsøget med denne tilføjelse, var AR.Dronen meget mere responsiv og efter at have fintunet HoverTrackTaskens PID-parametre, var AR.Dronen i stand til at forblive over markøren. På fig 5.5 ses HoverTrackTraskens PID error-værdier i forhold til markøren på gulvet, over en periode på mere end 2 minutter. Errorværdierne er omregnet fra pixels til millimeter, ved hjælp af den højdeværdi der løbende modtages fra navdatastrømmen. Det bemærkes, at der er et udsving omtrent midt i forløbet. Dette er ikke usædvanligt, da AR.Dronen er meget påvirkelig af turbulens og dette let forekommer, især i mindre lokaler.

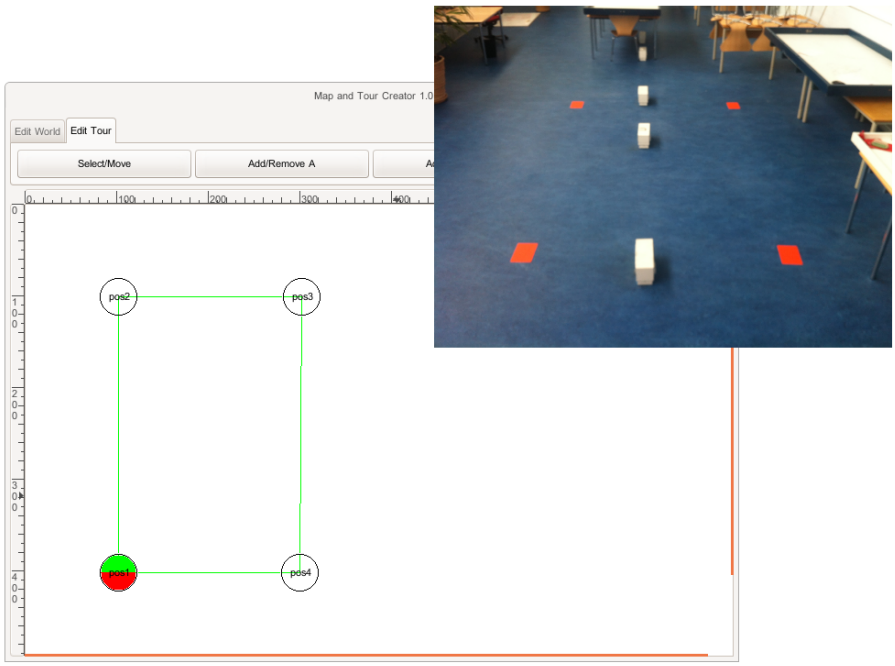


Figur 5.5: HoverTrackTaskens errorværdier over en længere periode. Errorværdierne angiver hvor mange millimeter AR.Dronen befinder sig fra markørens centrum. Errorværdierne er indsamlet ved at gemme disse for hvert gennemløb af HoverTrackTaskens `update`-metode.

Tredje eksperiment med simpel markør

Sidste eksperiment med simple markører anvendte en FollowTourTask der fulgte en rute mellem fire markører på gulvet. Eksperimenterne startede denne gang fra jorden, da en FollowTourTask selv letter og lander. Eksperimentets setup, både det konstruerede map-objekt og den fysiske opstilling ses på figur 5.6.

[H]



Figur 5.6: Repræsentation af det map-objekt FollowTourTasken anvender til at Navigere og den fysiske opstilling map-objektet afspejler.

Det var forventet, at udfordringen ville være at opdage markøren tidsnok, til at stoppe AR.Dronen før den var fløjet forbi. Dette problem kan dog løses ved at lade AR.Dronen flyve i en passende højde, en højde på 1800 mm blev fundet passende til formålet. Til gengæld afdækkede eksperimentet to andre problemer. Det første problem relaterede til, at de anvendte markører var ens. Dette betød, at der var tilfælde hvor AR.Dronen fandt samme lokation to gange i træk, og fejlagtigt antog, at den genfundne lokation var den næste i rutebeskrivelsen. Problemet lagde grund til tanken om at anvende mere avancerede lokationsmarkører, for at opnå en bedre lokalisering. Det andet problem var, at hvis AR.Dronen først mistede en markør fra synsfeltet, så var det umuligt at fortsætte. Dette problem blev løst ved at tilføje en recover-mode til HoverTrackTasken, således at denne kunne bevæge AR.Dronen opad indtil enten markøren igen kunne genkendes eller en højdegrænse blev nået,

hvorefter experimentet alligevel måtte afsluttes. Denne simple tilføjelse gjorde FollowTourTasken væsentligt mere robust.

Resultater

De overordnede resultater af eksperimenterne med simple lokationsmarkører var, at AR.Dronens evne til at genkende simple markører blev bekræftet. Desuden blev både HoverTrackTasken og FollowTourTasken udviklet i løbet af eksperimentet. I løbet af tredje eksperiment viste det sig, at markørkonceptet var for simpelt til at være praktisk anvendeligt. Dette medførte udviklingen af mere avancerede markører (næste sektion) for bedre understøttelse af FollowTourTasken. [12](youtube link) viser en video hvor AR.Dronen flyver frem og tilbage mellem to simple markører og demonstrerer at lokaliseringen, på trods af de to nævnte problemer, oftest fungerer.

5.2.2 Lokalisering via avancerede markører

Efter de indledende eksperimenter, blev det besluttet at designe og anvende en mere avanceret markør. Det første forslag var markøren der ses på figur 5.4(b).

Detekteringsalgoritme

Grundideen er at man først detekterer den røde firkant, hvorefter det indkransede område afsøges for en farvet blob der relaterer markøren til en specifik lokation.

For at sikre en hurtig detektering (og dermed lokalisering) gør algoritmen omfattende brug af OpenCVs, [45], optimerede billedbehandlingsalgoritmer. Detekteringsalgoritmen er implementeret i den virtuelle sensor PositionSilhouetDetector og afvikles således konstant for at give et så nøjagtigt billede af det øjeblikkelige miljø som muligt.

Første skridt i algoritmen er at udføre thresholding ved hjælp af OpenCVs `inRange` algoritme. Dette giver et billede, hvor alle de oprindelige røde pixels nu er hvide og resten er sorte. Andet skridt er at finde konturer i det binære billede. Dette gøres med OpenCVs `findContours` algoritme. Tredje skridt indbefatter at udvælge de konturer der har et stort nok areal til at være interessante. Dette gøres med OpenCVs `contourArea` algoritme. Fjerde skridt er at undersøge områderne der dækkes af de udvalgte konturer, for at

finde en farvet blob. Denne undersøgelse anvender igen OpenCVs `inRange` algoritme. Hvis en sådan farvet blob findes, kan farven sammenlignes med beskrivelsen af de forskellige lokationer i map-objektet og dermed lokalisere AR.Dronen præcist.

Første eksperiment med avancerede markører

Dette eksperiment udførtes på samme måde som 5.2.1, blot var den simple markør skiftet ud med en semiavanceret markør, vist på figur 5.4(b). Forventningen var fra start, at det ville være svært at identificere markøren over en hvis højde, da den røde markering ikke er lige så koncentreret som på den simple markør. Efter at have tilpasset algoritmens thresholdværdier og udført eksperimentet i højder fra 0.5 til 2.0 meter, blev forventningerne bekræftet. Den semiavancerede markør kunne med en hvis sikkerhed genkendes op til 1.5 meters højde, men herefter faldt detektionsraten betydeligt, resultaterne kan ses i tabel 5.3.

En detektionshøjde på 1.5 meter ligger lidt under 1.8 meter, som tidligere var bestemt som en passende flyvehøjde. En markør med større rød overflade blev derfor udviklet, for at sikre bedre detektion, se figur 5.4(c). En gentagelse af eksperimentet med den avancerede markør bekræftede, at denne nu kunne detekteres i op til 2 meters højde. Den avancerede markør er desuden designet sådan, at den på sigt kan bruges til at udlede AR.Dronens retning. Dette vil gøre FollowTourTasken mindre afhængig af de upræcise ψ-målinger fra AR.Dronens piezo-sensor (se sektion 2.6).

Andet eksperiment med avancerede markører

Dette eksperiment er næsten identisk, med det andet eksperiment der blev udført for de simple markører. Det skal dog nævnes, at HoverTrackTasken i mellemtiden blev redigeret til at anvende den virtuelle sensors detekteringsresultater, istedet for selv at udføre en detekteringsalgoritme. Under dette eksperiment var den virtuelle sensor indstillet til ikke at skifte mellem de to kamerastrømme. Som forventet var AR.Dronen også i stand til at holde positionen over den avancerede markør.

Tredje eksperiment med avancerede markører

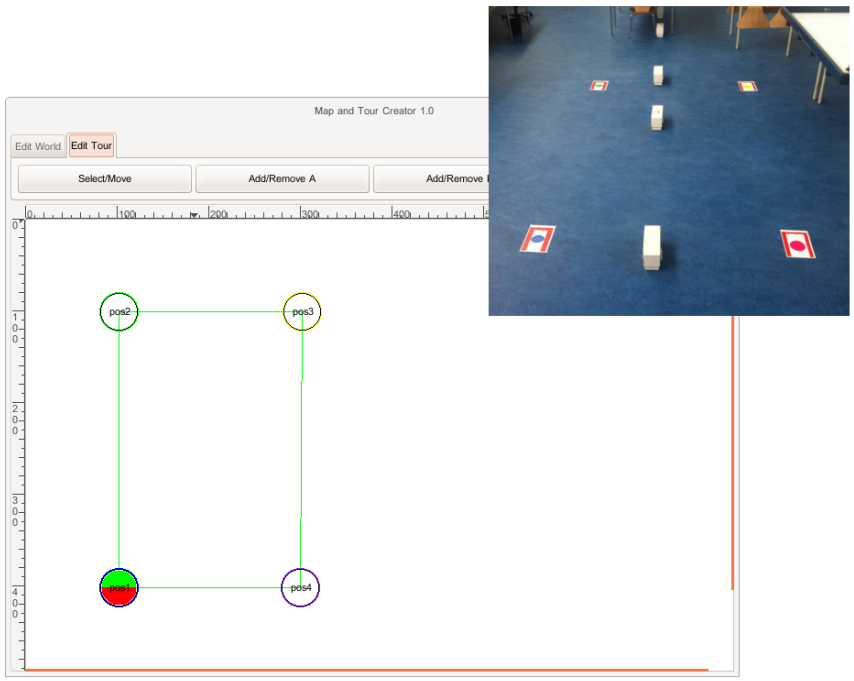
Dette eksperiment varierer fra det ovenstående, ved at den virtuelle sensor her løbende skifter mellem de to kamerastrømme. Det blev undersøgt om

HoverTrackTasken stadig var i stand til at stabilisere AR.Dronen over en erkendt lokation, når bundkameraets framerate blev begrænset. Forsøget viste, at AR.Dronen stadig forblev stabil over lokationen. Til gengæld stoppede den pludseligt da AR.Dronens originale styringssoftware lukkede ned efter omtrent ti minutter. Nærmere undersøgelser viser, at AR.Dronens software muligvis har en memoryleak. Dette viser sig ved at softwarens hukommelsesforbrug stiger kraftigt, så snart de hurtige kameraskift iværksættes ved brug af AT-kommandoen *set_config*. Denne opførsel beskrives også i sektion 3.2.1.

Fjerde eksperiment med avancerede markører

Ligesom i sektion 5.2.1, anvendes der også til dette eksperiment en opsætning med fire markører. Formålet var at bekræfte, at der stadig med de nye markører og den ændrede HoverTrackTask, kunne navigeres mellem de fire punkter med FollowTourTasken. Opstillingen kan ses på figur 5.7.

Eksperimentet bekræftede, at FollowTourTasken konsistent kunne lokalisere AR.Dronen og følge den fastlagte tour.



Figur 5.7: Billede af opstilling med avancerede markører til brug for det fjerde eksperiment med disse markører

5.2.3 Detektering af markører på små billeder

I de ovenstående eksperimenter anvendes AR.Dronens bundkamera. Dette har en opløsning på 176x144. Der er dog andre tasks der skal bruge data fra frontkameraet. Af den grund kan det være nødvendigt kontinuerligt at zappe mellem de to kameraer og dette fungererer som nævnt i sektion 5.2.2 ikke særligt godt. Derfor undersøges mulighederne for at anvende den videostrøm hvor bundkameraets billeder er lagt oven på frontkameraets (se figur 2.18, side 36). Når bundkameraets billeder udtrækkes fra denne strøm, er disse billeder i en opløsning på 88x72 pixels. Eksperimenterne er fuldstændig magen til dem der udførtes i sektion 5.2.1 og sektion 5.2.2. De er ligeledes udført i højderne 0.5, 1.0, 1.5 og 2.0 meter og er udført både med de simple, semi-avancerede og avancerede markører. Resultaterne ses i tabel 5.4, de viser

tydeligt at det ikke er muligt at anvende de små billeder, da de selv for de simple markører kun giver et pålideligt resultat op til 1.5 meter, hvorefter detektionsraten falder betydeligt.

Højde	Simpel	Semiavanceret	avanceret
0.5	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.5	1.0	0.805	1.0
2.0	1.0	0.045	1.0

Tabel 5.3: Detektionsrater for de forskellige markører i højder op til 2.0 meter

Højde	Simpel(88x72)	Semiavanceret(88x72)	Avanceret(88x72)
0.5	1.0	0.615	1.0
1.0	1.0	0.065	0.935
1.5	1.0	0.0	0.015
2.0	0.17	0.0	0.015

Tabel 5.4: Detektionsrater for de forskellige markører i højder op til 2.0 meter når der anvendes 88x72 pixels billeder

5.2.4 Lokalisering via QR-koder

I det følgende afsnit undersøges muligheden for at anvende QR-koder til lokalisering. Man er begyndt at se QR-koder mange steder i det offentlige rum, hvor de ofte bruges i forbindelse med reklamekampagner. QR-koder bruges ofte til at indkode en webadresse, men kan i princippet anvendes til at indkode en arbitrær streng. QR-koder er blevet mere populære, i takt med at opløsningen på mobiltelefonkameraer er steget. De fleste moderne mobiltelefoner har således et kamera med en opløsning flere gange højere end AR.Dronens kamera.

Til dette eksperiment anvendtes bibliotekerne pyqrcode, [35], og Zbar, [6], til henholdsvis indkodning og afkodning af QR-koder. Zbar er det bibliotek der også bruges i iPhone-app'en til afkodning af QR-koder. Zbar har pythonbindinger så det kan bruges på billeder taget med AR.Dronen.

Figur 5.8: Teksten "woah" som QR-kode

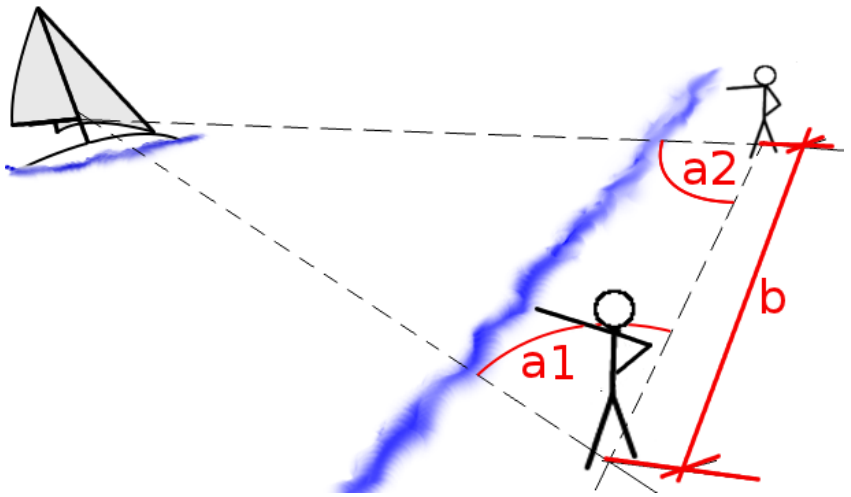
Formålet med eksperimentet var primært, at undersøge om AR.Dronens kameras opløsning er høj nok til at afkode QR-koder. Til formålet fremstilledes QR-koden der ses på figur 5.8. Den blev printet på et A4 ark og det blev kontrolleret at koden kunne afkodes ved hjælp af en iPhone. Herefter konstrueredes et lille testprogram, der henter billeder fra AR.Dronens frontkamera (320x240 pixels) og scanner disse for QR-koder ved hjælp af Zbar.

Afviklingen af eksperimentet foregik ved at QR-koden blev placeret på en væg og AR.Dronen placeret i skiftende afstande til væggen. QR-koden blev forsøgt afkodet på afstande op til en meter, men der var på intet tidpunkt en succesfuld afkodning. IPhonen kunne samtidig afkode QR-koden på alle afstande op til en meter. Resultatet viser med al tydelighed, at QR-koder ikke kan anvendes med AR.Dronens kamera og at andre markører derfor må anvendes til lokalisering.

5.3 Afstandsbedømmelse vha. triangulering

For at lette formidlingen af ideen bag dette eksperiment, er det her forsøgt at introducere problemstillingen via en lignende, men noget simplere testopstilling, helt uden quadrokoptere.

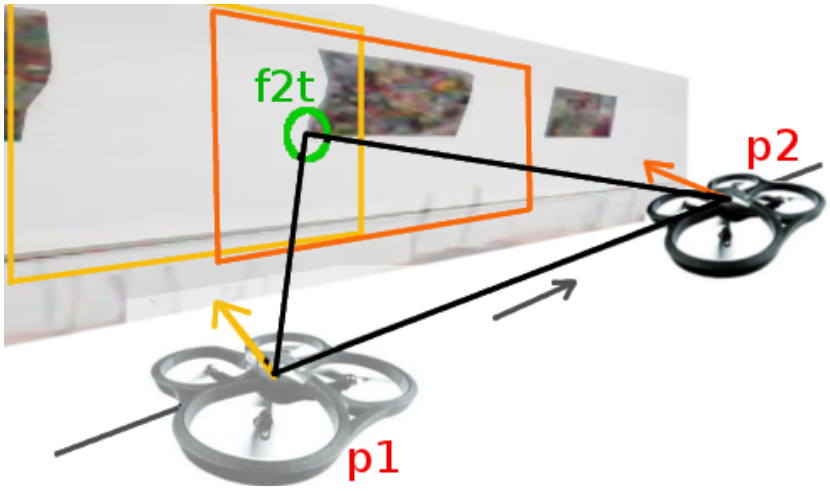
Figur 5.9 viser hvordan to mennesker på to forskellige positioner langs en kyst kan udregne afstanden ud til et skib på havet ved hjælp af triangulering.



Figur 5.9: To mennesker på en kyst kan i fælleskab bedømme afstanden ud til en båd på vandet, ved at kende afstanden imellem sig (b) og vinklerne imellem hinanden og båden (hhv. $a1$ og $a2$).

Hvis b, $a1$ og $a2$ er kendt, så kan de to på stranden i fællesskab udregne afstanden ud til båden. Afstanden imellem personen bagerst og båden i figuren vil eksempelvis kunne udregnes ved hjælp af, [90]: $A1 = \frac{\sin(a2)*b}{\sin(180-(a2+a1))}$.

Ideen med eksperimentet er så, at få AR.Dronen til alene at agere de to mennesker i eksemplet herover og derigennem finde afstande til objekter foran frontkameraet. Hvis AR.Dronen sættes til at flyve *sidelæns* over en afstand, svarende til strandkanten herover, vil den til to tider $t1$ og $t2$ være ved to positioner $p1$ og $p2$. Hvis tiden imellem de to tider er tilpas lille, eller hvis AR.Dronen flyver tilpas langsomt, vil der være et overlap i billederne taget med frontkameraet ud for de to positioner (figur 5.11).

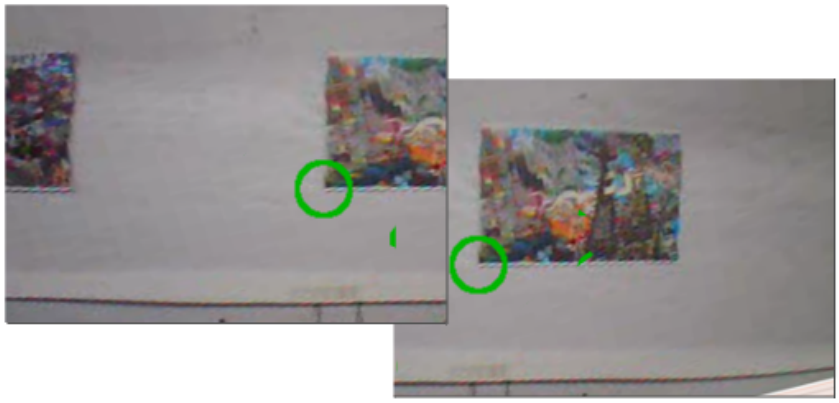


Figur 5.10: AR.Drone flyver sidelæns mens frontkameraet filmer væggen. Den grønne cirkel ($f2t$) er en genkendelig feature (uddybes i sektion 5.3.1) som optræder i begge billeder. Billederne er taget ud for position $p1$ og $p2$. Det er afstanden mellem $f2t$ og $p2$, i planet, der ønskes fundet. Den grå pil i figuren viser AR.Dronens sidelæns flyveretning. De orange pile viser hvilken retning AR.Dronens frontkamera filmer.

I figur 5.10 får den genkendelige feature (i den grønne cirkel) den samme rolle som båden i figur 5.9, mens AR.Dronens to positioner har menneskenes rolle. Implementationen er ikke begrænset til at holde styr på afstanden til et enkelt punkt, men kan tage vare på en hel liste af features i billederne fra AR.Dronens frontkamera. For hver feature er det muligt at udregne afstanden til AR.Dronens position. I sektionen herunder (sektion 5.3.1) beskrives den valgte metode til at finde, genfinde og genkende features i en videosekvens. I de næste to sektioner (5.3.2 og 5.3.3) forsøges forklaret hvorledes vinklerne $a1$ og $a2$ i figur 5.9 udledes af pixel-positioner i billedet fra frontkameraet, til brug i beregningen. I sektion 5.3.4 forklares udregningen af liniestykket b imellem de to positioner $p1$ og $p2$, for til sidst at kunne udregne afstanden $|ft2\ p2|$ i sektion 5.3.5. Sektion 5.3.6 fremlægger resultatet af det eksperiment der blev udført.

5.3.1 Optisk feature-tracking

I det følgende bruges OpenCV-funktionerne `goodFeaturesToTrack` og `calcOpticalFlowPyrLK`. Programmet sættes til først at finde en liste af egnede features[1] ($f2t_i$) een gang i billedramme 0 med `goodFeaturesToTrack`. Hvorefter hver enkelt $f2t$-position følges fra billedramme til billedramme med opticalflow-funktionen `calcOpticalFlowPyrLK`.



Figur 5.11: Et typisk egnet good-feature-to-track er som den grønne ring på de to billedrammer: Et hjørne på en billedramme af et billede i billedet. Punktet bag den grønne cirkel i billedet til venstre er genfundet i billedet til højre vha. funktionen *calcOpticalFlowPyrLK*. Funktionen returnere en liste med nye positioner for genkendte features (grønne cirkler).

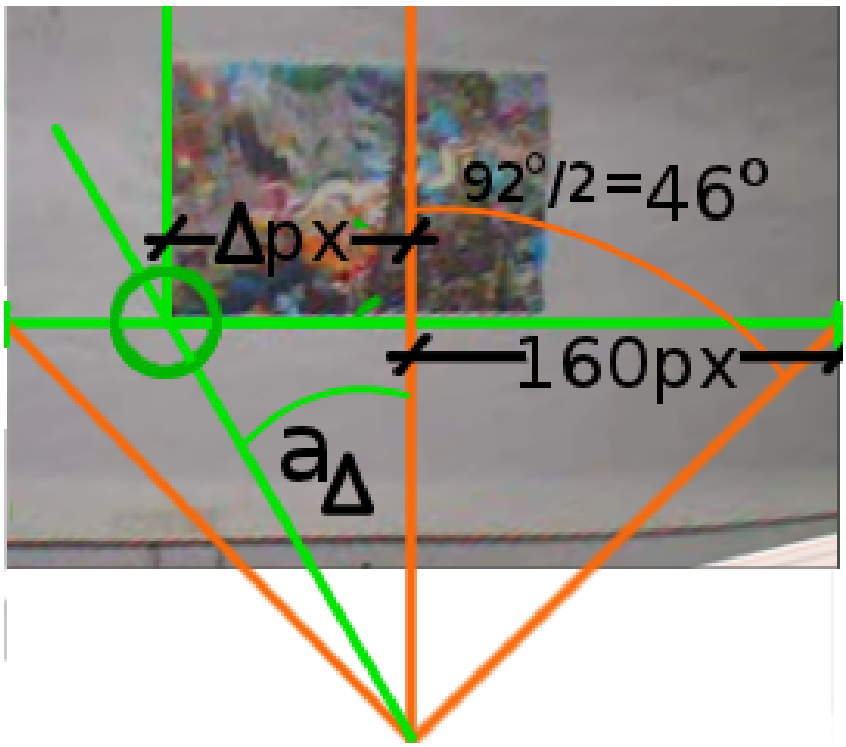
5.3.2 Fra pixel-position til virkelig retning

Da AR.Dronen ikke altid peger præcis i retningen af det der skal afstandsbedømmes (oftest ligger den fulgte feature enten til venstre eller til højre for midten af billedet, se de to billeder i figur 5.11), er det nødvendigt at udregne featurens retning i forhold AR.Dronens front.

Af en features horisontale pixel-position i billedet kan man udregne vinklen i forhold til retningen på AR.Dronens frontkamera, se vinklen a_Δ i fi-

[1] En egnet feature er et karakteristisk punkt i billedet som kan genkendes selvom billedet flytter fokuspunkt. En feature kan være flere ting, bl.a. 2. ordens gradient i billedintensiteten over en hvis magnitude, hjørner på objekter, karakteristisk tekstur, osv.

gur 5.12.

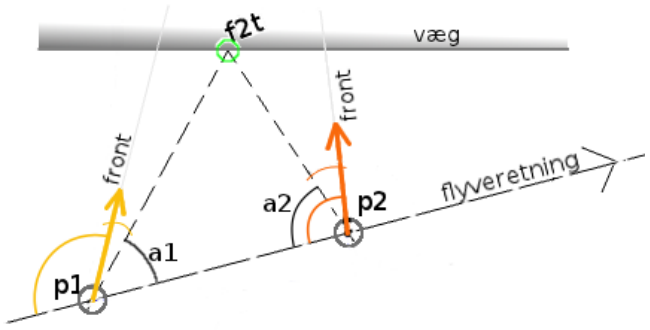


Figur 5.12: Bredden på billederne i videostrømmen fra AR.Dronens frontkamera er 320 pixel og kameraoptikken på AR.Dronens frontkamera har et horisontalt udsyn på 92 °. Hvis den horisontale afstand (Δpx) i pixels mellem 2 punkter (midten og featuren) i et billede er kendt, kan denne viden tilsammen give vinklen a_Δ, som $a_\Delta = \Delta px * \frac{92°}{320px}$.

Vinklen a_Δ er dog kun orienteret i forhold til AR.Dronens front, altså AR.Dronens lokale, interne, koordinatsystem. For at få en retning ud som kan bruges til triangulering, er det nødvendigt at flytte den over i et globalt koordinatsystem, så retninger og positioner har samme referencepunkt og kan bruges i afstandsberegningen.

5.3.3 Yaw som fælles reference

Det er imidlertid et problem at AR.Dronens front nærmest aldrig peger i den samme retning (i figur 5.13 ses at AR.Dronens front peger i forskellige retninger ved de to positioner $p1$ og $p2$). For at triangulere, skal der bruges to pejleretninger med fælles reference. Den piezo-elektriske yaw-sensor er ikke præcis nok til at kunne udgøre et kompas, via dead-reckoning, gennem en hel flyvning. Tanken er dog, at yaw-værdien over korte tidsperioder, alligevel er stabil nok som fælles referencevinkel, f.eks. mellem $p1$ og $p2$.



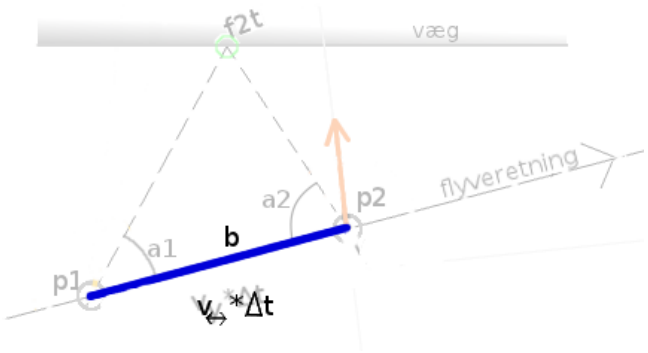
Figur 5.13: Opstillingen er set oppefra. AR.Dronen har flyttet sig fra position $p1$ til $p2$ i planet. Den grønne cirkel (f2t) er et punkt på væggen (feature-to-track) som følges gennem frontkameraet. Vinklerne $a1$ og $a2$ er orienteret i forhold til samme referencepunkt, ved at bruge AR.Dronens yaw-sensor.

5.3.4 Sidestykket b, 3. parameter i trekanten

Der mangler nu en parameter, for at kunne bestemme alle de andre elementer i trekanten $|p1\ f2t\ p2|$, heriblandt afstanden $|f2t\ p2|$. Den sidste ukendte parameter længden $|p1\ p2|$ (se figur 5.14) er lidt besværlig at fremskaffe idet der ikke findes andre afstande at måle udfra. Til gengæld kendes hastigheden som AR.Dronen har fløjet med imellem $p1$ til $p2$. Den sideværts hastighed fåes fra AR.Dronens navdata som v_y, og er hastigheden i y-aksens bevægelsesretning ($v_{\leftrightarrow}$ i figur 5.14).

Testopstillingen er forsøgt simplificeret til kun at foregå i den sideværts

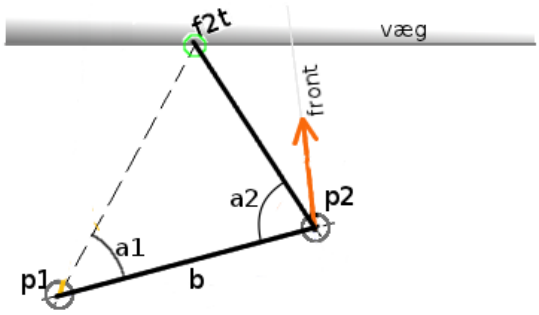
bevægelsesretning. Frem- og tilbage bevægelser, rotationer og ændringer i højden forsøges minimeret under flyvningen imellem $p1$ og $p2$. Det antages herved at de parametre er så små at de kan ignoreres i udledningen af afstanden imellem $p1$ og $p2$.



Figur 5.14: Opstillingen er set oppefra. AR.Dronen har flyttet sig fra position $p1$ til $p2$ i planet på tiden Δt. Her ønskes sidelængden $b = |p1\ p2|$ fundet. Afstanden b findes ved: $b = \Delta t * v_{\leftrightarrow}$.

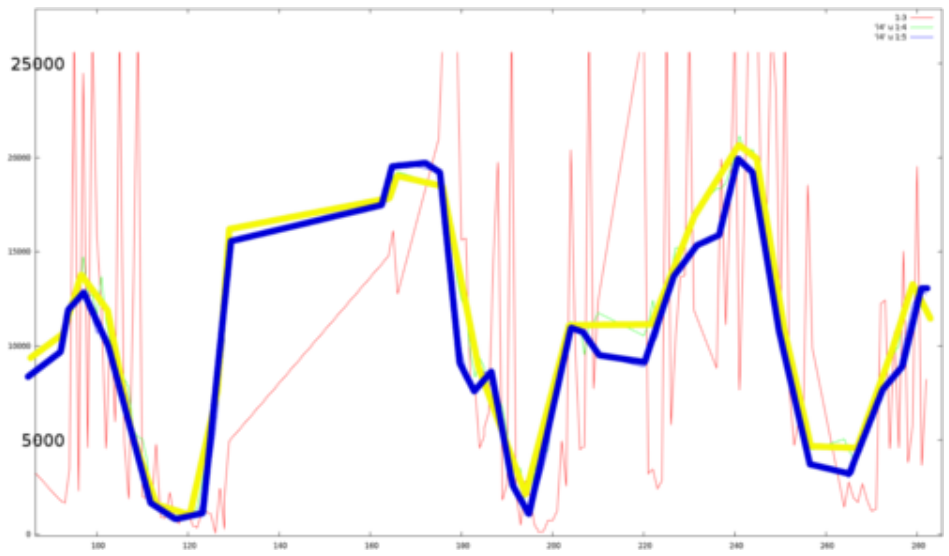
5.3.5 Afstanden ud til objektet findes

Det er sidestykket $|f2t\ p2|$ i trekanten på figur 5.15 der er forsøgt udregnet. Eftersom der nu er fremskaffet tre elementer af trekanten $|p1\ f2t\ p2|$'s vinkler og sider, er det nu muligt at beregne afstanden $|p2\ f2t|$.



Figur 5.15: Opstillingen er set oppefra. AR.Dronen har flyttet sig fra position $p1$ til $p2$ i planet på tiden Δt. Den grønne cirkel (f2t) er et punkt på væggen (feature-to-track) som følges gennem frontkameraet. I opstillingen ønskes liniestykket $|f2t\ p2|$ fundet. Ved at kende vinklerne $a1$ og $a2$, samt længden $b = |p1\ p2|$, er det nemt at finde den sidste afstand som: $|f2t\ p2| = \frac{\sin(a2)*b}{\sin(180-(a2+a1))}$

5.3.6 Resultater



Figur 5.16: Den beregnede afstand ud til et fast punkt på væggen i en filmsekvens over 290 billedrammer, hvor AR.dronen flyver langsomt sidelæns ca. 5 meter fra punktet. Den gule og blå streg er hhv. gennemsnittet og gaussfiltreret over et vindue med 11 målinger. Enheden for y-aksen er millimeter, mens x-aksen er billedrammenummeret.

Resultatet af eksperimentet kan ses i figur 5.16. De 16 vildeste outsidere i målingerne er klippet af ved 27 meter. Den sande værdi (afstanden mellem AR.Dronen og væggen) ligger imellem $4,0$ og $6,5$ meter.

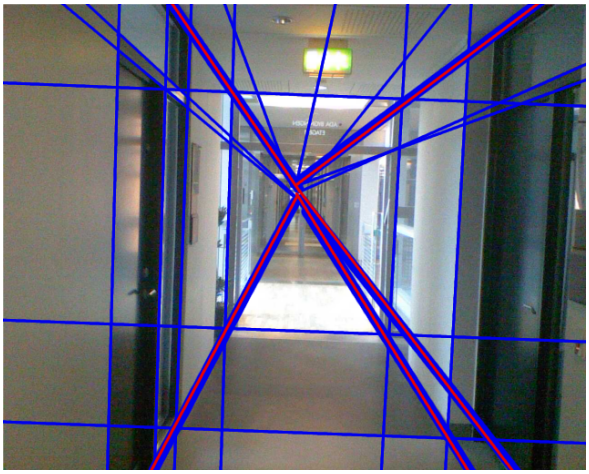
Grunden til de meget svingende resultater med denne opsætning er formodentlig, at vinklen der skal trianguleres udfra bliver for spids. Liniestykket imellem $p1$ og $p2$ er for kort. Det er meget begrænset, hvor langt AR.Dronen kan nå at flyve mellem 2 billedopdateringer. Selv hvis der kun er 10 billedopdateringer per sekund, så vil Δt være lig $0,1$ sekunder. Hvis AR.Dronen bevæger sig med en hastigheden af eksempelvis 100 millimeter per sekund, så er afstanden mellem de to punkter $100\ mm/s * 0,1\ s = 10\ mm$. Hvis de 2 andre sidelængder er ca. 5 m, altså en faktor 500 gange større, er det nemt at se, hvad lidt varians vil have af indflydelse på afstandsbedømmelsen.

En udbygning af afstandsbedømmelsen vil være at lade Δt vokse lidt, således at afstanden udregnes hen over flere billeder ad gangen. Et andet forsøg kunne være, at udbygge implementationen til at projektere featurens horisontale afstand i billedrammen ned på virkelighedens vandret ved at bruge ϕ-vinklen fra AR.Dronens navdata.

5.4 Korridor-detektering

Denne sektion omhandler en metode til at detektere gangarealer og korridorer gennem AR.Dronens frontkamera. Målet for dette eksperiment er at implementere og teste to funktionaliteter: 1) At kunne detektere om AR.Dronen er i en korridor, og 2) At kunne detektere hvor i et billede enden af korridoren (forsvindingspunktet) er.

For en autonom AR.Drone er det hensigtsmæssigt, at have kontekstuel viden, for at kunne træffe funderede beslutninger om navigationen i specifikke omgivelser. Derfor vil det være interessant, at se om det er muligt at implementere en korridor-detektions funktionalitet, som senere kan blive en del af det virtuelle sensorhieraki. Robotten Minerva, [71], fik lov at lave guidede tours på the Smithsonian i to uger. I eksperimentet var robotten blevet givet forskellige opførsler at udføre under forskellige forhold. Eksempelvis skulle den så ofte som muligt bruge analogien Coastal-planner, dvs. at følge væggene rundt så den havde mulighed for at genkende allerede gemte kendemærker. Når Coastal-planner ikke var mulig, skulle den gå over i Open-sea mode, hvor den så havde et andet bevæggrundlag. Et andet eksempel på forskellige operationstilstande er i, [4], hvor en quadrokopter, med frontkamera, med godt resultat kan kategorisere dens omgivelser til at være i en trappeopgang, i et gangareal eller i et lille kontor. I modsætning til Minerva, bruger denne tilgang ikke 2D kort eller omgivelser til pathplanning. Viden om omgivelserne bruge til at indsnævre de optiske strukturer (perspective cues) der skal søges og orienteres efter.

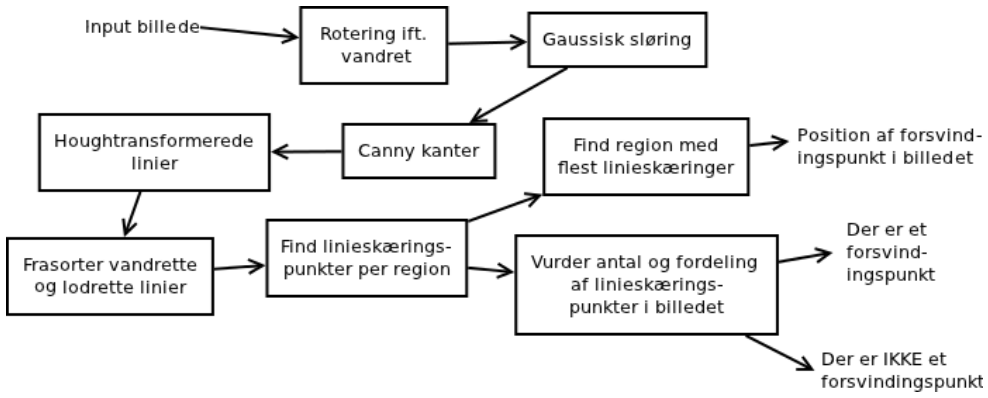


Figur 5.17: Kendetegnende for et billede der er taget ned ad et gangareal er forsvindingspunktet, som består af en del diagonale linier som forsvinder ind i billedet og skærer i cirka det samme punkt i billedet.

Eksistensen af et forsvindingspunkt i billedet fra AR.Dronens frontkamera kan bruges som indikator for om AR.Dronen befinder sig i en korridor. Hvis et billede viser en korridor, vil man i strukturen i billedet kunne se mange linier gå vandret og lodret, mens en del linier vil gå diagonalt i billedet, altså ind i korridoren. Gennem dette eksperiment har det vist sig, at der hvor de diagonale linier skærer hinanden, er der ofte et sammenfald med enden af korridoren i billedet.

5.4.1 Korridordetektering som procesdiagram

Processen med at finde et forsvindingspunkt handler om at finde skæringspunktet, af de diagonale linier som går ind i billedet.



Figur 5.18: Detekteringsprocessen tager et billede fra AR.Dronens frontkamera som input. Billedet filtreres for at finde Houghtransformerede linier. De fundne linier sorteres så kun de diagonale linier er tilbage. Fordelingen af linieskæringspunkter i billedet beregnes. Den celle, eller det område af billedet, med størst koncentration af linieskæringer udpeges som forsvindingspunkt.

Forsvindingspunktets placering i et billede findes gennem en række deltrin (se figur 5.18). Inputbilledet roteres først ift. AR.Dronens roll-vinkel, ϕ (se figur 5.19), herefter beregnes linierne i billedet ved hjælp af et Canny-filter, [8], og Houghtransformation, [34] (gennemgåes i sektion 5.4.2).

Figur 5.19: For senere at lette frasorteringen af de vandrette og lodrette linier fra i den reele korridor, bruges AR.Dronens roll-vinkel ϕ. Herefter kan linier filtrerers via deres hældning alene.

De ikke-diagonale linier frasorteres, hvorefter skæringspunker mellem hvert par af de diagonale linier beregnes. Antallet af linieskæringer tælles op for hver celle i billedet (der er 11×11 celler per billede, se figur 5.21(b)). Afhængig af om der er en korridor eller ikke, er det som sagt oftest ved cellen med størst antal linieskæringer forsvindingspunktet kan findes. Hvorvidt der er tale om et billede af en korridor, kan ofte afgøres ved at kigge på om antallet af linieskæringer i cellen med flest skæringer er over en vis størrelse (en grænse på 140 har vist et godt resultat). Herunder gennemgåes metoder til at finde kanter (edges) via Canny-filter, samt hvorledes disse kanter går fra at være repræsenteret som klumper af billedpunkter til at blive til Houghtransformerede linier (der er en uddybende forklaring af teknikken i appendiks B). Herefter forklares hvordan linierne filtreres, for til sidst at sige noget om et tilstedeværende eller ikke tilstedeværende forsvindingspunkt.

5.4.2 At finde linier i billedet

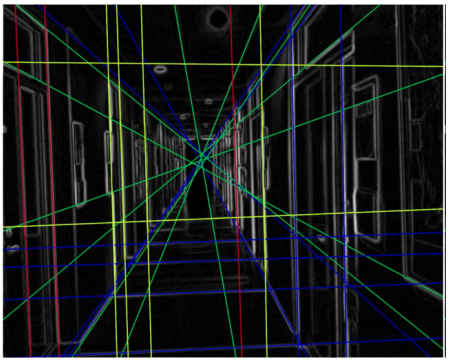
Det at finde linier foregår i to skridt, vha. to forskellige algoritmer. I dette eksperiment er OpenCVs implementation af et Canny-filter, [8], brugt til at finde kanter med. På kanterne er der brugt en OpenCV implementation af Hough-transformation, [34], til at finde linier med. En mere detaljeret

beskrivelse af metoderne til at detektere kanter og linier i appendiks B.

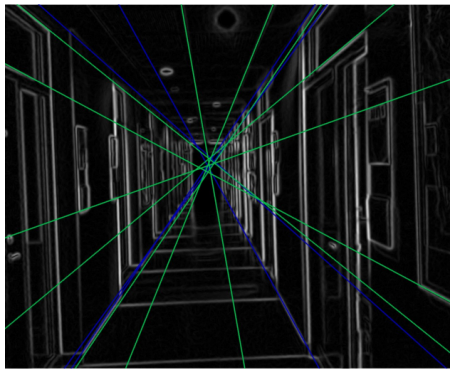
Figur 5.20 illustrerer forskellige trin i filtreringsprocessen fra input billede til en samling af diagonale linier på polær form.

(a) Et billede af en gang påført gaussløring.

(b) Billede a) efter Canny kantdetektering

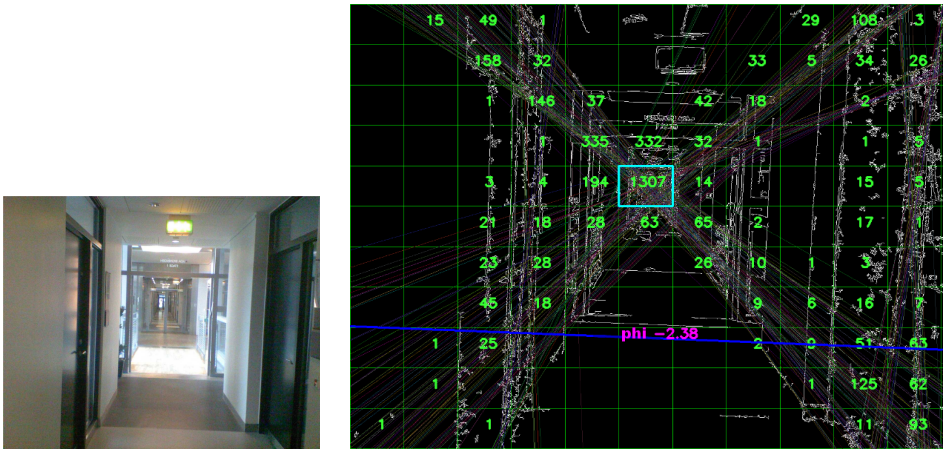


(c) Houghtransformerede linier, udregnes udfra kantpunkterne i billede b)



(d) Vandrette og lodrette hough-linier sorteres fra billede c)

Figur 5.20: Detektering af Hough-linier.



(a) Gangareal Adabygningen, Katrinebjerg

(b) Billedet er opdelt i 11 × 11 celler. I hver celle gemmes antallet af linier som skærer hinanden. I dette tilfælde er cellen med flest skæringer godt overensstemmende med korridorens forsvindingspunkt i billedet.

Figur 5.21: Visualisering af vanishinpoint detektor.

5.4.3 Celle med flest linie-skæringspunkter

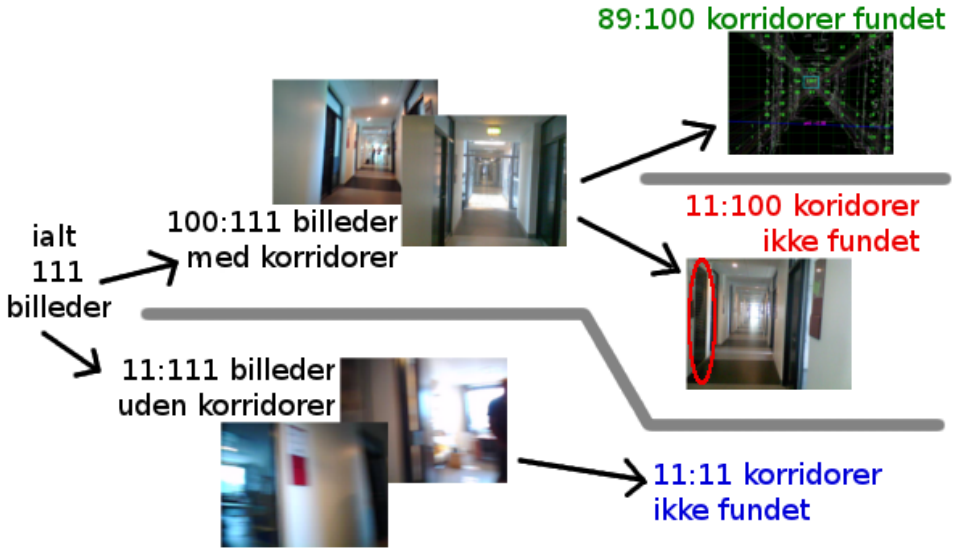
Ved at finde det område af billedet hvor flest linier skærer hinanden, kan man under de rigtige forhold få et godt gæt på hvor billedets forsvindingspunkt kan findes. I figur 5.21(b) ses ved området for celleindgang (5,5) 1307 linie-skæringer: En god indikator på et forsvindingspunkt. Observationer under eksperimenterne har vist, at cellen med flest skæringspunkter skal indeholde et hvis antal for at algoritmen med rimelig sikkerhed kan sige at der faktisk er en korridor med tilhørende forsvindingspunkt.

5.4.4 Resultater

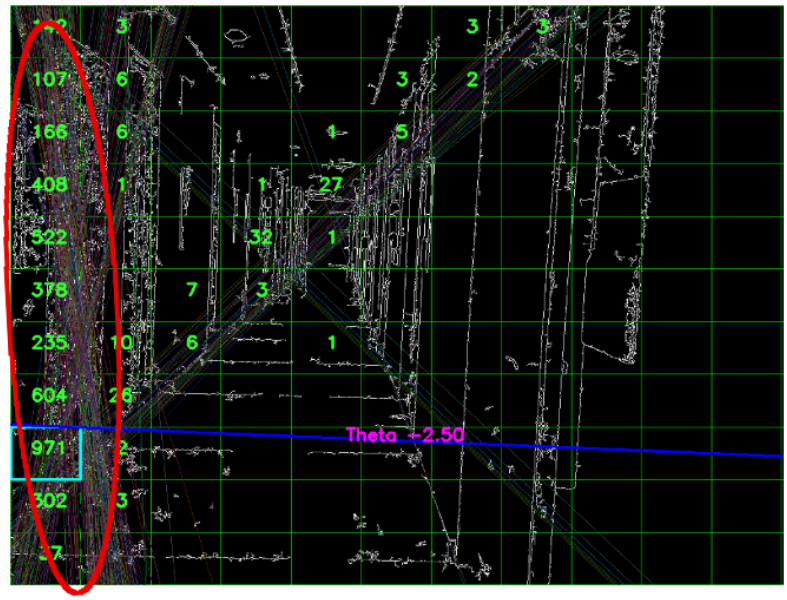
Figur 5.21(a) er et eksempel på et billede med optimale forhold til at finde et forsvindingspunkt.

Ud af de 111 detekteringstest der er lavet, hvor korridorenden ses i midten af et billede, hvor billedet er taget fra midten og ned ad Ada 100-gangen, finder detektoren området for forsvindingspunktet 86 gange. Udover de 86 er

der 11 tilfælde hvor detekteringen ikke virker. Et eksempel er hvis billedet er taget i den ene side af gangen, hvor Canny-filteret har detekteret mange næsten lodrette linier, men ikke lodret nok til at blive frasorteret via ϕ-grænsen. Der vil der blive et stort antal linieskæringer i siden som vil føre til fejl-placering af forsvindingspunktet (se figuren 5.23).



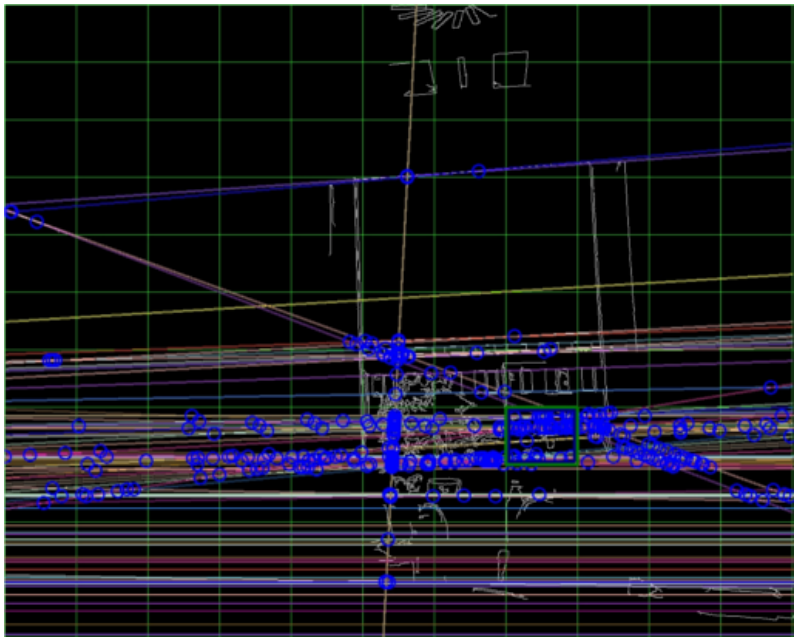
Figur 5.22: Testen har ialt gennemløbet 111 billeder. 11 af de 111 er med det blotte øje vurderet til at være uden korridor (f.eks billeder af dørkarm, eller et vindue i et kontor). Af de resterende 100 billeder indeholdende korridor-ender, har detektoren været i stand til at finde forsvindingspunkter i de 89 af billederne. De resterende 11 billeder kategoriserede detektoren som værende uden forsvindingspunkt, selvom der med det blotte øje kunne ses en korridor ende omkring midten af billedet.



Figur 5.23: Eksempel på synsvinkel hvor detektoren ikke fungerer optimalt. Canny-filteret har fundet mange kantpunkter i venstre side af billedet hvor vægoverfladen er meget uensartet.

Oprindeligt blev der opsat et mål om at detektoren skulle kunne give en sandsynlighed for hvorvidt AR.Dronen befandt sig i en korridor. Det mål er ikke opfyldt. Med de opstillinger der blev lavet test med, er der til gengæld ikke observeret nogle false positives, dvs. detektoren har ikke fejlagtigt vist at der var en korridor i de test hvor der ikke var en korridor. I den opstilling der er lavet test med har detektoren konkret gættet rigtigt 100 gange ud af 111 (se figuren 5.22).

Observationer har indikeret, at detektoren kan gøres mere robust ved yderligere at analysere fordelingen af linieskæringspunkterne i billedet. Skæringspunkterne skal gerne være jævnt fordelt over det meste af billedet, gerne i et diagonalt kryds over forsvindingspunkt som i figur 5.21(b). Eksempelvis bør fordelingen af skæringspunkter ikke kun være koncentreret omkring en enkelt linie, som i figur 5.24, eller kun omkring et enkelt område.



Figur 5.24: Et sted hvor detektoren ikke skal finde en korridor.

En videreudvikling kunne være, at udbygge detektoren til at anvende mere end bare det nyeste billede, evt. ved at parre placeringen af korridorens endepunkt med en ψ-værdi fra AR.Dronens yawvinkel, som hele tiden opdateres også når AR.Dronens front drejer rundt og står på tværs af korridoren, således at endepunktet senere kan genfindes.

Implementationen er en simplificeret løsning, den virker i de lette tilfælde og er knap så resoucekrævende som tilgangen i [4], hvor der blandt andet bruges en hidden Markov model og Viterbi-algoritme, til at huske hvor korridorenden sidst blev set og om det så er rimeligt at tro, at den er der hvor detektoren siger den er nu.

5.5 Odometrisk afstandsmåling

Dette eksperiment undersøger virkningen af en odometrisk afstandsmåler. Den odometriske afstandsmåler er implementeret i den virtuelle sensor DistanceTracker, [25], og holder styr på den afstand AR.Dronen har bevæget

sig i løbet af en flyvning. Konceptet er kendt som en triptæller i biler og som en tacho-counter i motoren på kørende robotter.

Indenfor robotnavigation anvendes ofte en teknik kaldet dead reckoning, [78]. Dead reckoning-teknikken går ud på, at den nuværende position udregnes på baggrund af en tidligere positionsudregning og viden om retning og hastighed. En mulig anvendelse af odemetri-afstandsmåleren kan således være, som en del af en sådan dead reckoning algoritme. En forudsætning for en robust og brugbar dead reckoning er dog, at den udledte afstand er troværdig. Hvilket er det der testes her.

DistanceTrackeren fungerer ved at integrere over de hastighedsestimater (v_x og v_y), der modtages i AR.Dronens navdatastrøm. Disse hastighedsestimater er baseret på en kombination af accelerometerdata og en avanceret hastighedsudledning fra bundkameraet, [5].

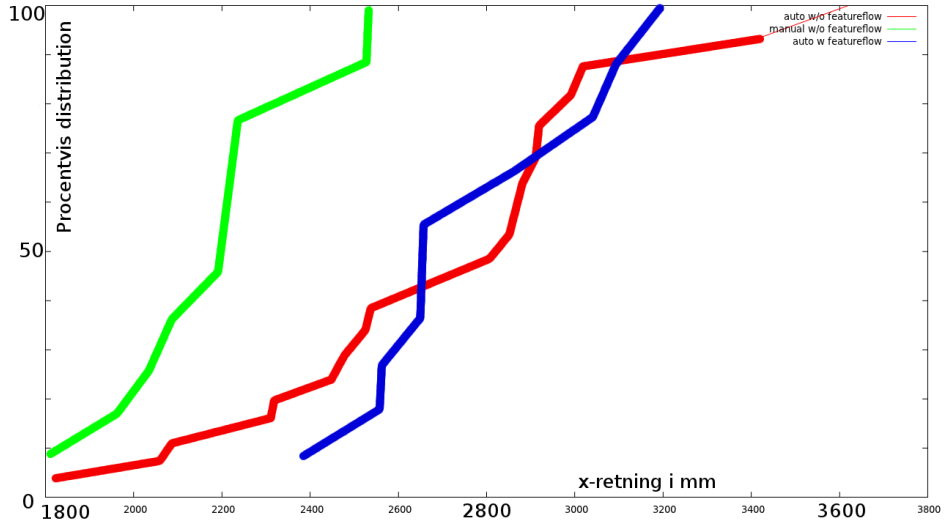
5.5.1 Eksperimentet

Eksperimentet blev udført ved at placere to markører med 3 meters afstand på gulvet. Herefter blev der gennemført 3 forskellige typer flyvninger mellem de to markører: Manuel flyvning med ensartet underlag (uden gode features), automatisk flyvning med ensartet underlag og automatisk flyvning med uensartet underlag (med gode features at tracke). Hver type flyvning blev gentaget 10 gange. Den manuelle styring forgik ved brug af joystick, mens de automatiserede flyvninger blev udført ved brug af en FollowTourTask (sektion 3.3.1). [11] (youtube video) viser en flyvning med en FollowTourTask, hvor den tilbagelagte afstand måles.

5.5.2 Resultater

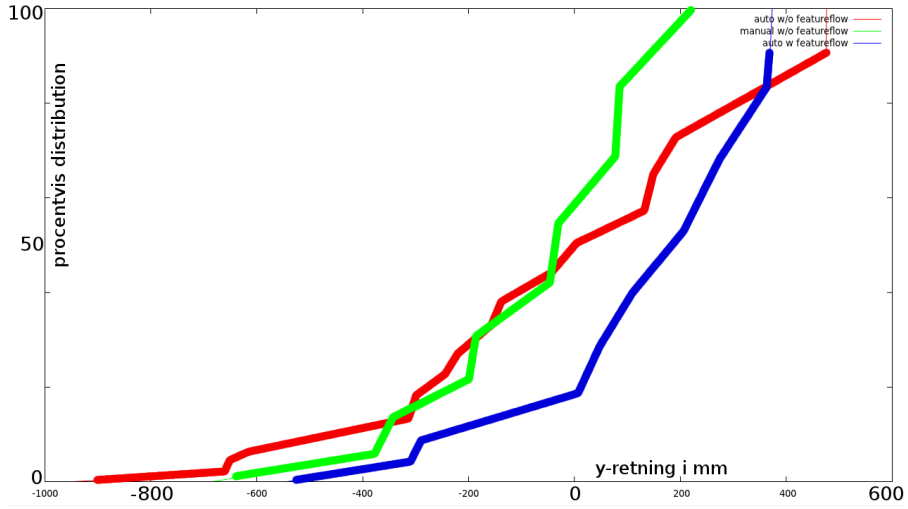
Generelt blev de manuelt styrede flyvninger gennemført med højere hastighed end de automatiserede, mens de automatiserede flyvninger i reglen tog længere tid at færdiggøre. Den længere flyvetid for de automatiserede flyvninger er grundet i, at FollowTourTasken bruger tid på at centrere AR.Dronen over markøren, både før der flyves og før der landes. Desuden flyves der med en FollowTourTask ikke hurtigere, end at markørerne på gulvet kan opfattes.

Resultaterne fra flyvningerne kan ses på figur 5.25 og figur 5.26. Figur 5.25 viser en fordeling over de målte afstanden i fremadrettet flyveretning, mens figur 5.26 viser fordelingen af de målte afstande i y-aksens (sideværts) retning.



Figur 5.25: Kumulativ fordeling af målte afstande. Målingerne blev foretaget under fremadrettet flyvning over 3000 millimeter. Der blev udført manuelle og automatiserede flyvninger over ensartet og uensartet overflade. Grøn er manuel flyvning, mens blå og rød er autonom flyvning.

Af figur 5.25 (og figur 5.26) observeres det, at medianen af de målte afstande under de manuelt styrede flyvninger ligger tydeligt lavere end medianen af afstande målt under de autonome flyvninger.



Figur 5.26: Kumulativ distribution af målte afstande. Målingerne repræsenterer AR.Dronens tilbagelagte afstand på tværs af flyveretningen og burde altid være lav, da AR.Dronen kun blev fløjet fremad og ikke til siderne. Grøn er manuel flyvning, mens blå og rød er autonom flyvning.

En anden observation af eksperimentet og graferne herover er, at fordelingen af de målte afstande ikke er distribueret omkring den sande værdi på 3000 mm, men generelt er fordelt omkring lavere værdier. Der er to mulige forklaringer på observationen: Enten er tidsfaktoren som der integreres over for lille, eller også er hastighedsudledningen fra AR.Dronens navdata generelt for lav. Hvis AR.Dronen sættes til altid kun at flyve med en og samme hastighed, kan problemet løses ved at bruge en større tidsfaktor i integrationen af hastigheden. En mere hensigtsmæssig og mere interessant løsning vil være at undersøge om hastigheden fra AR.Dronen skal kompenseres med både en offsetfaktor og en hældningsfaktor i forhold til den relle værdi.

Kapitel 6

Konklusion

I det følgende konkluderes på specialet tilstand målt i forhold til tese- og mål-formuleringen. Herefter evalueres på valget af specialets metode. Til sidst præsenteres løsningsforslag i forhold til det arbejde der ikke blev færdiggjort samt forslag til fremtidige eksperimenter og metoder.

6.1 Specialets tilstand

1. Konstruktion af klientplatform

Undervejs i hele specialeforløbet har den konstruerede platform understøttet arbejdet med AR.Dronen og har fungeret som en let anvendelig grænseflade til AR.Dronen.

1.(a) Facilitere eksperimenter

På trods af kontinuerlig udvikling under hele projektet, har klientplatformen understøttet eksperimenterne, ved at give let adgang til AR.Dronens sensorstrømme og ved at tilbyde task-abstraktionen ovenpå kontrolinterfacet. Det udviklede tasksystem har været så fleksibelt, at det både har kunnet anvendes til simple, tidsbegrænsede flyvninger i en specifik retning og til sammensatte navigationsmønstre med genkendelse af gulvmarkører.

Tasksystemets træstruktur gør det muligt, at kombinere tasks i henholdsvis sekventielle og parallele grupper. Strukturen giver anledning til et forholdsvis kompliceret tasksystem, som ofte kunne være implementeret i et fladt hieraki med samme resultat. Fordelen ved strukturen i tasksystemet er,

at en forholdsvis kompleks task som FollowTourTask, kan sammensættes af flere mindre og ikke så komplekse subtasks. Dermed åbnes op for genbrug af disse subtasks. En vigtig pointe er at træstrukturen også giver mulighed for grafiske repræsentationer af komplicerede tasks, der giver et bedre overblik over den implenterede funktionalitet.

1.(b) Tilgængeliggøre et interface til AR.Dronen

Eksemplerne med let anvendelse af pythoninterfacet i sektion 3.1 viser, at klientplatformen, og delelementer af denne, er nem at anvende for udviklere. Det implementerede testdevice giver desuden mulighed for udvikling gennem simulering uden en AR.Drone tilstede.

2. Semi-autonom navigation

Klientplatformen har under eksperimenterne med visuel lokalisering vist (sektion 5.2), at den muliggør semiautonom navigation i et miljø med visuelle markører. Semi-autonom navigation defineres her som, at AR.Dronen er i stand til at følge en på forhånd planlagt rute. Denne funktionalitet implementeres af FollowTourTasken, 3.3.1, som kan læse og følge en rute. En mere avanceret opførsel der også lader AR.Dronen undgå mennesker i synsfeltet, kan opnåes ved at afvikle en AvoidTask, 3.3.1, sideløbende med FollowTourTasken. En sikker navigation kan opnåes når markørerne ligger forholdsvis tæt på hinanden (afstand < 3 meter) og positionsdetektoren er kalibreret til lysforholdene. Under disse forhold, er FollowTourTasken blevet afprøvet med gode resultater på en række forskellige ruter(i forbindelse med eksperimenterne).

3. Udledning af yderligere navigationsdata

Udledning af yderligere højere-ordens navigationsdata fra AR.Dronens sensorstrømme, har været et hovedtema i de udførte eksperimenter. Herunder konkluderes på specialets opsatte delmål:

3.(a) Lokalisering via WIFI

Eksperimenter der undersøgte anvendeligheden af WIFI-fingerprinting til lokalisering i sektion 5.1, viste at det var muligt at lokalisere AR.Dronen til at

være i f.eks den ene eller den anden ende af en bygning (lokationer med 6 m afstand). Der er teknikker til at forbedre resultaterne, men teknikken er meget følsom og kræver et større eksperiment-setup end vi har haft mulighed for.

Det forudgående arbejde med at få tilkoblet en USB-WIFI-adapter til AR.Dronen, har dog vist, at det generelt er muligt at udvide AR.Dronen med USB-enheder.

3.(b) Lokalisering via visuelle markører

En stor del af de udførte eksperimenter har omhandlet visuel lokalisering på forskellige måder. Under denne process er der udviklet en visuel markør, figur 5.4, som konsistent kan genkendes med AR.Dronens bundkamera. Markøren og FollowTourTasken, som er udviklet sideløbende, muliggør AR.Dronens navigation efter en planlagt rute.

Desuden er det vist, at AR.Dronens kameraer ikke er anvendelige til afkodning af QR-koder og at anvendelsen af overlagte billeder ikke er anvendelig i forbindelse med lokalisering.

3.(c) Detektering af navigationsmiljøtype

Eksperimenterne med korridordetektion, sektion 5.4, har vist meget lovende muligheder, både for at kunne detektere at AR.Dronen faktisk befinder sig i en korridor og for at kunne navigere via en sådan korridor.

3.(d) Odometrisk afstandsmåling

Eksperimenterne med odometrisk afstandsmåling (sektion 5.5) har vist, at denne teknik kan anvendes til grove estimater for AR.Dronens tilbagelagte afstand. Resultaterne viser, at den målte afstand er meget følsom overfor forskellige miljøfaktorer som belysning, overflade aftegninger i gulvet der flyves over og den hastighed der flyves med. En dead-reckoningalgoritme, kun baseret på denne afstand og en retning, vil ikke være pålidelig, men afstanden kunne f.eks. anvendes til at visualisere AR.Dronens omtrentlige position på et digitalt oversigtskort, evt. kombineret med lokation via WIFI.

3.(e) Optisk triangulering af afstande

På trods af at resultaterne fra eksperimentet med afstandstriangulering (sektion 5.3) ikke var overbevisende, så ser det alligevel ud som om at det er muligt, at forfine teknikken og producere bedre resultater. Teoretisk er der adgang til alle de nødvendige parametre i trianguleringsberegningen og mulige forbedringer til teknikken beskrives sidst i sektionen.

6.2 Metodens anvendelighed

Metodevalget har taget udgangspunkt i AR.Dronen og er samtidig begrænset af AR.Dronen. De eksperimenter der er udført i specialet, er valgt udfra hvad der umiddelbart syntes oplagt at teste, når AR.Dronens specifikationer og egenskaber tages i betragtning. Eksperimenterne er lavet for at teste ideer til udbygning af AR.Dronen som robot udfra AR.Dronens umiddelbare formåen som fjernstyret quadrokopter.

Vurdering af AR.Dronen som robotplatform

I sektion 2.6 er vurderingen af AR.Dronen beskrevet. Generelt vurderes AR.Dronen som et godt alternativ til at bygge en quadrokopter-platform fra bunden. Selvom AR.Dronen var valgt på forhånd i specialet, og eksperimenterne er opstået ud af den, så har AR.Dronen også været en begrænsende faktor i specialet, på godt og ondt. AR.Dronen har været en god hjælp til at afgrænse domænet for specialet, men de indbyggede sensorer har begrænset udviklingsmulighederne: Eksempelvis udelukker den begrænsede videoopløsning afkodning af QR-koder til brug i lokaliseringseksperimentet (sektion 5.2.4), og bestemmelse af AR.Dronens orientering (kompas) besværliggøres på grund af sensordrift. En mulig løsning kunne være at gøre brug af de ny AR.Drone version 2.0, hvorpå der er udskiftet en del sensore og installeret nye (frontkamera med større opløsning og magnetometer-sensor). En anden mulighed er selv at udvide AR.Dronen med andre sensorer som beskrevet i sektion 2.5. I starten af specialet, var der ideer om at montere IR-lys eller laser til afstandsmåling.

Klientplatform

Valget af Python som implementationssprog til klientplatformen har haft både positive og negative konsekvenser. Fra et Performance-synspunkt har Python ikke været et godt valg. Enkelte gange med den trådbaserede taskimplementation, har systemet været presset til det punkt, hvor en flydende afvikling ikke længere var mulig. En sidegevinst har imidlertid været, at det har fordret et fokus på at skrive mere effektiv kode, især i forbindelse med udviklingen af taskssystemet. For at kombinere hastigheden fra sprog som C og C++ med udviklingshastigheden og letlæseligheden fra Python, vil en mulighed være at implementere et eller flere af klientplatformens moduler i C/C++ og så have bygget python-wrappere til disse moduler.

Såfremt det ikke havde været et mål i sig selv at implementere en platform, var det en mulighed at anvende ROS (Robot Operating System), [83]. ROS er et mere generelt framework til robotudvikling og har i forvejen en driver udviklet til AR.Dronen, [73].

OpenCV

Anvendelsen af OpenCVs billedbehandlingsalgoritmer har været en nødvendighed. Det havde ikke været muligt at behandle nær samme antal billed, hvis algoritmerne var forsøgt implementeret i Python. Anvendelsen af OpenCVs Python interface har været relativt nemt, på trods af få tilfælde hvor det ikke er helt klart, hvilken version af en algoritme der bruger hvilke parametertyper (billede-argumenterne repræsenteres som IPlImage-type i en version og som et array i en anden version. Samtidig er ikke alle dele af biblioteket lige opdateret).

6.3 Fremtidigt arbejde

Det er oplagt at undersøge alternativer til anvendelsen af Psyco i forbindelse med billeddekodning. En mulighed kunne være at implementere hele dekodningsalgoritmen i C og så lave python wrappere hertil, en anden mulighed kunne være at anvende eller tilpasse en allerede fungerende JPEG afkoder.

Det har været planen at udvide sensorhierakiet med flere virtuelle sensorer. Efter de lovende resultater med korridordetektoren, kunne denne algoritme med fordel implementeres i klientplatformen som en virtuel sensor.

Udledning af AR.Dronens retning ved hjælp af den udviklede markør er en forbedring, som kan opnåes med forholdsvis simple midler.

Som vist er der muligt at udvide AR.Dronen med USB-moduler, det vil være nærliggende at undersøge denne tilgang yderligere, ved at montere andre sensorer til at supplere med udledningen af mere højere-ordens navigationsdata. Specielt hvis AR.Dronen skal flyve indendørs blandt mennesker vil det øge sikkerheden at montere afstandssensor rundt på AR.Dronen.

At afstandsmålingen gennem triangulering fra AR.Dronen ikke lykkedes er ærgeligt, og er en oplagt mulighed at genoptage. Eksperimentet viste at beregningen var forkert, men det svarer ikke på om trianguleringen kan eller ikke kan lade sig gøre. Der er flere faktorer der skal være opfyldt, før man kan sige det kan lade sig gøre. Både den udledte odometriske afstand, vinklen udledt fra frontkameraets optik og orienteringen fra piezo-sensoren skal opfylde nogle tolerancekrav for at afstandsmålingen er brugbar. Det vil være et meget interesant eksperiment at færdiggøre.

AR.Dronens store udbredelse og det aktive udviklermiljø vil fungere som både hjælp og inspiration under udviklingen af nye projekter med AR.Dronen.

Selv om andre kritisere AR.Dronen som legetøj og ikke anvendelig som videnskabelig platform, [58] og andre allerede er begyndt at eksperimentere med mindre quadrokoptere med mindre inerti-kapacitet, [72], så virker AR.Dronen til prisen som værende god-nok.

Erfaringen med AR.Dronen gennem projektet er, at den måske ikke virker helt så godt til indendørs brug som man skulle tro (når der nu følger et indendørs skjold med). I små rum har AR.Dronen vanskeligheder ved at stå stille i luften, pga. vind-turbulens, ekko fra de omgivende overflader mv. I stedet for at AR.Dronen bruges på et museum indendørs, er et andet og bedre brugscenarie måske at den skal bruges udendørs. AR.Dronen kan flyve steder hvor der ikke er umiddelbar adgang for mennesker (se eksemplet med inspektioner af broer og højspændingsledninger). Denne egenskab kan udnyttes, så AR.Dronen kan bruges som trækplaster til events eller på turist-atraktioner.

Figur 6.1: AR.Drone flyvning ved Stevns klint. Der filmes udenfor klinten og ind på kirken.

Højerup gamle kirken ved Stevns klint vil være et oplagt sted. For hundrede kroner kan gæster styre AR.Dronen ud for at se Kirken fra havsiden. Gæster kan se og dele andres flyvninger på optagelser. For et lille ekstra beløb kan gæsten købe filmoptagelsen på en CD eller SD-kort.

Litteratur

[1] C. Anderson. Ar.drone hacking? `http://www.diydrones.com/forum/topics/ardrone-hacking`, juli 2012.

[2] P. Bahl and V. Padmanabhan. Radar: an in-building rf-based user location and tracking system. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 775 –784 vol.2, 2000.

[3] D. Bernoulli. *Danielis Bernoulli Joh. Fil. Hydrodynamica, sive, De viribus et motibus fluidorum commentarii [microform] / opus academicum ab auctore, dum Petropoli ageret, congestum.* Sumptibus Johannis Reinholdi Dulseckeri, typis Joh. Henr. Deckeri, Typographi Basiliensis, Argentorati :, 1738.

[4] C. Bills, J. Chen, and A. Saxena. Autonomous mav flight in indoor environments using single image perspective cues. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5776 –5783, may 2011.

[5] P. Bristeau, F. Callou, and V. D. The navigation and control technology inside the ar.drone micro uav. In *IFAC World Congress*, volume 18, pages 1477–1484, 2011.

[6] J. Brown. Zbar bar code reader. `http://zbar.sourceforge.net/index.html`, juli 2012.

[7] S. Budman. The coolest video game? it's a hovercraft. `http://www.nbcbayarea.com/news/tech/The-Coolest-Video-Game--Its-A-Hovercraft-87412122.html`, juli 2012.

[8] F. J. Canny. A Computational Approach to Edge Detection. *j-IEEE-PAMI*, 8(6):679–698, 1986.

[9] T. L. Community. Lejos, java for lego mindstorms. http://lejos.sourceforge.net/, juli 2012.

[10] T. L. Community. Package lejos.robotics.subsumption. http://lejos.sourceforge.net/nxt/nxj/api/index.html, juli 2012.

[11] M. Daugaard and T. Thyregod. Ar.drone distancetracker run. http://www.youtube.com/watch?v=zxgVehK9vCY, juni 2012.

[12] M. Daugaard and T. Thyregod. Ar.drone followtourtask. http://www.youtube.com/watch?v=DAPj7RbFz3I, juni 2012.

[13] M. Daugaard and T. Thyregod. Compiling code for the ar.drone. http://taghof.github.com/Navigation-for-Robots-with-WIFI-and-CV/blog/2012/01/13/Compiling-Code-For-The-ARDrone/, juli 2012.

[14] M. Daugaard and T. Thyregod. controllers.py. https://github.com/taghof/Navigation-for-Robots-with-WIFI-and-CV/blob/master/controllers.py, juli 2012.

[15] M. Daugaard and T. Thyregod. Enabling the drone usb port. http://taghof.github.com/Navigation-for-Robots-with-WIFI-and-CV/blog/2012/01/12/Enabling-The-Drone-USB-Port/, juli 2012.

[16] M. Daugaard and T. Thyregod. load.sh. https://github.com/taghof/Navigation-for-Robots-with-WIFI-and-CV/blob/master/load.sh, juli 2012.

[17] M. Daugaard and T. Thyregod. load.sh. https://github.com/taghof/Navigation-for-Robots-with-WIFI-and-CV/blob/master/load.sh, juli 2012.

[18] M. Daugaard and T. Thyregod. mapcreator.py. https://github.com/taghof/Navigation-for-Robots-with-WIFI-and-CV/blob/master/mapcreator.py, juli 2012.

[19] M. Daugaard and T. Thyregod. map.py. `https://github.com/taghof/Navigation-for-Robots-with-WIFI-and-CV/blob/master/map.py`, juli 2012.

[20] M. Daugaard and T. Thyregod. Navigation for robots with wi-fi and cv. `http://taghof.github.com/Navigation-for-Robots-with-WIFI-and-CV/`, juli 2012.

[21] M. Daugaard and T. Thyregod. Non-threaded tasks. `https://github.com/taghof/Navigation-for-Robots-with-WIFI-and-CV/blob/master/newesttasks.py`, juli 2012.

[22] M. Daugaard and T. Thyregod. sensordisplay.py. `https://github.com/taghof/Navigation-for-Robots-with-WIFI-and-CV/blob/master/sensordisplay.py`, juli 2012.

[23] M. Daugaard and T. Thyregod. taskcreator.py. `https://github.com/taghof/Navigation-for-Robots-with-WIFI-and-CV/blob/master/taskcreator.py`, juli 2012.

[24] M. Daugaard and T. Thyregod. Udping atcommands from inside the drone. `http://taghof.github.com/Navigation-for-Robots-with-WIFI-and-CV/blog/2012/01/23/UDPing-ATcommands-from-inside-the-drone/`, juli 2012.

[25] M. Daugaard and T. Thyregod. virtualsensors.py. `https://github.com/taghof/Navigation-for-Robots-with-WIFI-and-CV/blob/master/virtualsensors.py`, juli 2012.

[26] T. Dean. Ultrasonic acoustic sensing. `http://www.cs.brown.edu/people/tld/courses/cs148/02/sonar.html`, juli 2012. (cached i /webcache/ paa den vedlagt CD).

[27] G. Depoyant and M. Ludmann. L.u.d.e.p leading units and drone enabled probing. `http://www.ludep.com`, december 2011.

[28] G. Depoyant and M. Ludmann. Tracking algorithm: considering the inclination of the drone. `http://www.ludep.com/tracking-algorithm-considering-the-inclination-of-the-drone/`, december 2011. (cached i /webcache/ paa den vedlagte CD).

[29] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16:97–166, December 2001.

[30] Dlink. Dwl-g122 54mbps traadloes usb adapter. `http://www.dlink.dk/cs/Satellite?c=Product_C&childpagename=DLinkEurope-DK%2FDLProductCarouselMultiple&cid=1197319529299&p=1197357728135&packedargs=ParentPageID%3D1197337625277%26ProductParentID%3D1197318706946%26TopLevelPageProduct%3DBusiness%26category%3DQuickProductFinder%26locale%3D1195806935729%26term%3DDWL-G122&pagename=DLinkEurope-DK%2FDLWrapper`, juli 2012. (cached i /webcache/ paa den vedlagte CD).

[31] R. Droms. Dynamic Host Configuration Protocol. RFC 2131 (Draft Standard), mar 1997. Updated by RFCs 3396, 4361, 5494.

[32] Droneparts.de. Brand new motor-set for the ar.drone 2.0. `http://droneparts.de/AR-Drone-1-0-AR-Drone-1-0-Original-Ersatzteile-Parrot-Motor-Set-for-AR-Drone-2-0--Neu-und-OVP!/a41728502_u5094_z68bb60df-2637-4789-a29b-3dace8eff92e/`, juli 2012.

[33] Droneparts.de. Droneparts.de. `http://www.droneparts.de`, juli 2012.

[34] R. Duda and P. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[35] A. Fiori. qrcode for python. `http://pyqrcode.sourceforge.net/`, juli 2012.

[36] U. I. Forum. Usb on-the-go. `http://www.usb.org/developers/onthego/`, juli 2012.

[37] S. Gaeremynck. navdata_common.h. `https://projects.ardrone.org/embedded/ardrone-api/d9/d3d/navdata__common_8h-source.html`, juli 2012.

[38] google.com. Google maps, ada-bygningen, aarhus universitet. `https://maps.google.com/maps?q=Aarhus+universitet&hl=en&ll=56.172202,10.187921&spn=0.000884,0.002411&sll=37.0625,-95.677068&sspn=39.`

371738,79.013672&t=h&hq=Aarhus+universitet&radius=15000&z=19, juli 2012.

[39] google.com. Google maps, zuse-bygningen, aarhus universitet. https://maps.google.com/maps?q=Aarhus+universitet&hl=en&ll=56.173142,10.189312&spn=0.000847,0.002411&sll=37.0625,-95.677068&sspn=39.371738,79.013672&t=h&hq=Aarhus+universitet&radius=15000&z=19, maj 2012.

[40] S. Lenser, J. Bruce, and M. Veloso. A modular hierarchical behavior-based architecture. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Computer Science*, pages 79–99. Springer Berlin / Heidelberg, 2002. 10.1007/3-540-45603-1_54.

[41] N. Metni and T. Hamel. A uav for bridge inspection: Visual servoing control law with orientation limits. *Automation in Construction*, 17(1):3 – 10, 2007.

[42] R. Murphy. *Introduction to AI robotics*. The MIT Press, 2000.

[43] nokiawp.net. Control ar-drone with nokia n900. http://www.nokiawp.net/1344/control-ar-drone-with-nokia-n900/, juli 2012.

[44] OpenCV. Motion analysis and object tracking. http://docs.opencv.org/modules/video/doc/motion_analysis_and_object%_tracking.html?highlight=calcopticalflowpyrlk, juli 2012.

[45] OpenCV. Welcome to opencv documentation! http://opencv.willowgarage.com, juli 2012.

[46] OpenCV. Welcome to the full opencv wiki development. http://opencv.willowgarage.com/wiki/FullOpenCVWiki, juli 2012.

[47] Parrot. Ar.drone parrot – technologies. http://ardrone.parrot.com/parrot-ar-drone/en/technologies, januar 2012.

[48] Parrot. Technical specifications. http://ardrone2.parrot.com/ardrone-2/specifications/, juli 2012.

[49] Parrot.com. Ar.drone parrot - en. http://ardrone.parrot.com/, april 2012.

[50] Parrot.com. Parrot. http://www.parrot.com/, maj 2012.

[51] S. Piskorski. Code execution on the drone. https://projects.ardrone.org/boards/1/topics/show/560, juli 2012.

[52] S. Piskorski. Sdk question/documentation. https://svn.ardrone.org/boards/1/topics/show/585#message-665, juli 2012.

[53] S. Piskorski and N. Brulez. Ar.drone developer guide. Technical report, Parrot, februar 2011.

[54] J. Postel. User Datagram Protocol. RFC 768 (Standard), Aug. 1980.

[55] J. Postel. Internet Protocol. RFC 791 (Standard), sep 1981. Updated by RFC 1349.

[56] J. Postel and J. Reynolds. Telnet Protocol Specification. RFC 854 (Standard), may 1983. Updated by RFC 5198.

[57] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (Standard), oct 1985. Updated by RFCs 2228, 2640, 2773, 3659, 5797.

[58] P. Pounds, R. Mahony, and P. Corke. Modelling and control of a quadrotor robot. In *Australasian Conference on Robotics and Automation 2006*, Auckland, New Zealand, 2006. Australian Robotics and Automation Association Inc.

[59] M. Quan, E. Navarro, and B. Peuker. Wi-fi localization using rssi fingerprinting. http://digitalcommons.calpoly.edu/cpesp/17, 2010.

[60] J. Ransom-Wiley. Us army using xbox 360 controller in future combat systems tests. http://www.joystiq.com/2007/02/05/us-army-using-xbox-360-controller-in-future-combat-systems-tests/, jun 2012.

[61] A. Rigo. Psyco. http://psyco.sourceforge.net/, juli 2012.

[62] Shellware. looking for a roundel target? http://www.shellware.com/BlogEngine.Web/post/2011/04/27/Looking-for-a-Roundel-Target.aspx, juli 2012.

[63] S. Shen, N. Michael, and V. Kumar. Autonomous multi-floor indoor navigation with a computationally constrained mav. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 20–25. IEEE, 2011.

[64] P. Shinners. Pygame documentation. `http://www.pygame.org/docs/ref/joystick.html`, juni 2012.

[65] J. Sloan. Deconstructing the ar.drone: Part 6. `http://coverclock.blogspot.dk/2011/04/deconstructing-ardrone-part-6.html`, juli 2012.

[66] Szuu. Pygame documentation. `http://www.ardrone-flyers.com/forum/viewtopic.php?t=2704&p=20218l`, april 2012.

[67] T. T. team. pcap - packet capture library. `http://www.tcpdump.org/pcap3_man.html`, juli 2012.

[68] A. R. Technologies. Australian rc technologies. `http:\aust-rc-tech.com.au`, juli 2012.

[69] techpowerup.com. Parrot ar.drone, the package. `http://www.techpowerup.com/reviews/Parrot/AR_Drone/2.html`, juli 2012.

[70] the GNU development team. Gimp – the gnu image manipulation program. `http://www.gimp.org/`, juli 2012.

[71] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Minerva: A second-generation museum tour-guide robot. In *In Proceedings of IEEE International Conference on Robotics and Automation (ICRA'99)*, 1999.

[72] M. Turpin, N. Michael, and V. Kumar. Trajectory design and control for aggressive formation flight with quadrotors. *Autonomous Robots*, 33:143–156, 2012.

[73] B. University. Ros driver for the parrot ar.drone. `http://code.google.com/p/brown-ros-pkg/wiki/ardrone_brown`, juli 2012.

[74] B. Venthur. Converting the drones video frames to jpeg files. `https://projects.ardrone.org/boards/1/topics/show/2912`, juli 2012.

[75] B. Venthur. Python-ardrone. https://github.com/venthur/python-ardrone/blob/master/libardrone.py, jun 2012.

[76] D. Vlasenko. Busybox. http://www.busybox.net/, maj 2012.

[77] Wikipedia. http://en.wikipedia.org/wiki/Euler_angles#Tait.E2.80.93Bryan_angles, juli 2012.

[78] Wikipedia. Dead reckoning. http://en.wikipedia.org/wiki/Dead_reckoning, juli 2012.

[79] Wikipedia. Helicopter. http://en.wikipedia.org/wiki/Helicopter, juli 2012.

[80] Wikipedia. Opencv. http://en.wikipedia.org/wiki/OpenCV, juli 2012.

[81] Wikipedia. Pid controller. http://en.wikipedia.org/wiki/PID_controller, juli 2012.

[82] Wikipedia. Qr code. http://da.wikipedia.org/wiki/QR_Code, juli 2012.

[83] Wikipedia. Qr code. http://da.wikipedia.org/wiki/ROS_(Robot_Operating_System), juli 2012.

[84] Wikipedia. Ralink. http://en.wikipedia.org/wiki/Ralink, juli 2012.

[85] Wikipedia. Willow garage. http://en.wikipedia.org/wiki/Willow_Garage, juli 2012.

[86] Wikipedia.org. Gaussian blur – wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Gaussian_blur, juni 2012.

[87] Wikipedia.org. Parrot ar.drone – wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/AR.Drone, april 2012.

[88] Wikipedia.org. Parrot company – wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Parrot_(company), maj 2012.

[89] Wikipedia.org. Quadrotor – wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Quadrotor, maj 2012.

[90] Wikipedia.org. Sinusrelation – wikipedia, den frie encyklopedi. http://da.wikipedia.org/wiki/Sinusrelation, juli 2012.

[91] Wikipedia.org. Step detection – wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Step_detection#Global`, juli 2012.

[92] Wikipedia.org. Ubifs – wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/UBIFS`, februar 2012.

[93] Xbox. Kinect – xbox.com. `http://www.xbox.com/en-US/Kinect/`, juli 2012.

[94] youtube user Zonked420. Xbox 360 controller modded for military use. `http://www.youtube.com/watch?v=FqaI2sXf9iI`, juni 2012.

[95] ZDnet. Photos: The army's vision for soldier tech. `http://www.zdnet.com/photos/photos-the-armys-vision-for-soldier-tech/67546?seq=9&tag=siu-container;photo-frame#photo-frame`, jun 2012.

Bilag A

AT kommandoer

Dette afsnit beskriver de enkelte AT-kommandoer, en mere detaljeret beskrivelse findes i [53], side 29. En AT-kommando må ikke deles over flere UDP-pakker, men en UDP-pakke kan indeholde flere AT-kommandoer, kommandoerne skal blot være adskilt med en "linefeed"karakter og den samlede længde må ikke overskride 1024 karakterer. Såfremt en kommando overskrider længdebegrænsningen eller på anden måde ikke overholder syntaksen, vil AR.Dronen ignorere kommandoen.

Tabel A.1 viser hvilke kommandoer der jf. [53], kan bruges til at kontrollere AR.Dronen.

Navn	Argumenter	beskrivelse
REF	input	Kommando til takeoff, land og nødstop
PCMD	flag, roll, pitch, gas, yaw	Flytter AR.Dronen
FTRIM		Sætter den horisontale referenceværdi
CONFIG	key, value	Ændrer en konfigurationsparameter
LED	animation, frequency, duration	Viser en LED animation
ANIM	animation, duration	Afspiller en flyvesekvens
COMWDG		Resetter komunikationstimeren

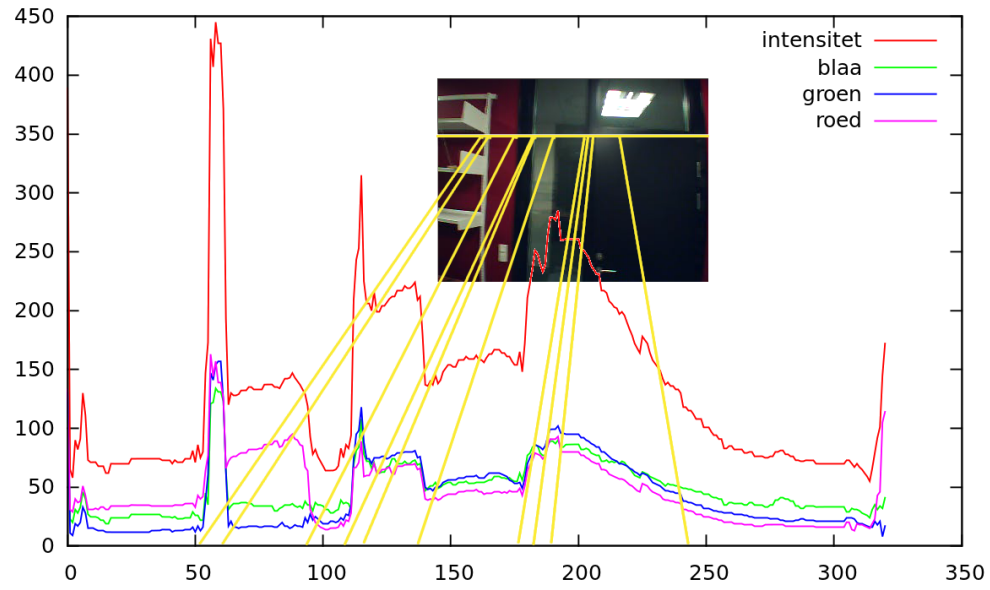
Tabel A.1: Gyldige AT-kommandoer

Bilag B

Canny kanter og Hough linier

B.1 At finde kanter i et billede

Herunder gives en definition på hvad kanter (edges) er, samt deres egenskaber. Forklaring er simplificeret og tager udgangspunkt i et eksempel med kantdetektering i kun 1 dimension, hvor den rigtige teknik bruger gradienten i 2 dimensioner.



Figur B.1: Graf over rød-, grøn-, blå-farveværdier for en linie i billedet med pixel indeks $j = 67$, $i = \{0, 1, \ldots, 320 - 1\}$. Samt pixel intensitet, her givet ved summen af de tre farver for i og j. Der hvor grafen, f.eks. for intensiteten, har stejlest hældning er der stor sandsynlighed for en kant i billedet.

Kanter i et billede defineres som abrupt skift i farveværdi eller lysintensitet lokalt[1] mellem naboer af pixels. Kanter er ofte en indikator for eksistensen af bl.a. ændringer i dybden af et billede og for overfladeændringer m.fl. (se figur B.1).

For at forbedre resultatet af kant-filteret, kan indput-billedet forinden påføres en udglatning over, som udføres ved at folde billedet med en gaussisk klokke-operation, [86], (fra billedbehandlingsprogrammer, f.eks. GIMP, [70], er funktionen kendt som Gaussian-blur).

For herefter at finde kanter bruges Cannyfilteret, [8]. Canny blev valgt i stedet for Sobel, fordi Canny blev kendt først i specialet, men også fordi det ser ud til at kantmarkeringerne af et Sobelfilter er bredere end med Canny. Hvilket senere vil have indvirkning på Houghtransformens udførsel.

[1]Man kan også lede efter kanter globalt, se f.eks., [91], men det er en anden tilgang. I dette afsnit beskrives en metode der søger lokalt.

Cannyfilteret tager et billede i sort/hvide nuancer (1 kanal), så inden da, kopieres billedet fra tre kanaler til en kanal. Hvor billedet med den ene kanal er intensiteten (summen af tre farve kanaler rød, grøn og blå). Cannyfilteret returnere et binært billede (sort/hvid) som det i figur 5.20(b). Hvide billedpunkter indikere at der er en kant i det oprindellige billede, mens sort betyder at der ikke er ændringer.

Tager man alle pixelpositioner hvor den første afledte af intensiteten er over threshold[2] kan man ende op med mange tykke kanter. For at præcisere placeringen kan man udtynde dem, ved at søge i retningen af den lokale gradient og istedet finde optimum.

B.2 Liniedetektering via Houghtransformation

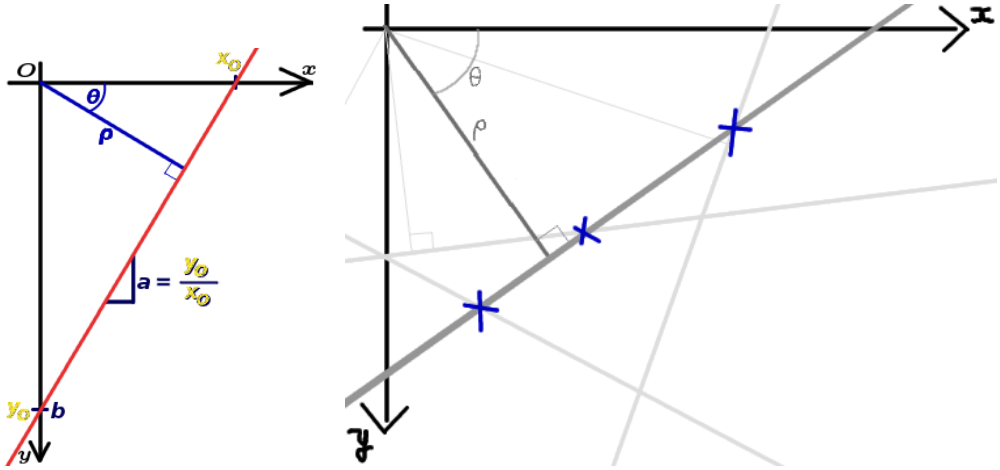
Generaliseret Houghtransformation, [34], er en metode til at finde perfekte linier hvor der ikke er andet end billedpunkter. Dvs. metoden finder steder i et billeder hvor der ligger mange billedpunkter på en lige række, og kalder rækken for en linie.

For hvert billedpunkt (x_0, y_0) i billedet kan man skrive de linier der går gennem punktet som

$$\rho_\theta = x_0 * \cos\theta + y_0 * \sin\theta \tag{B.1}$$

Hvor $0 \leq \theta < 2\pi$ og $0 \leq \rho_\theta$. De linier der går gennem punktet (x_0, y_0) repræsenteres ved de (θ, ρ_θ)-par der er løsninger til linien.

[2]Forklaret via det simple eksempel i figur B.1, ellers vil det være gradienten (x og y)



(a) En linie beskrevet ved det velkendte almindelige cartesisk koordinatsystem som: $y = ax + b$. Samme linie beskrevet på polarform ved en vinkel θ og en afstand ρ. Forholdet imellem de 2 repræsentationer er $y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{\rho}{\sin\theta}\right)$

(b) Houghtransformeret linie der gennemløber 3 billedpunkter (blå kryds).

Man kan opsætte et kriterie for hvad der gør en linie god, f.eks. at linien skal gennemløbe mindst X billedpunkter, for komme i betragtning til at være detekteret som en linie (minimum for X er 3 punkter). De linie-par som er løsning til flest konkrete B.1-ligninger vil rangere højest i resultatlisten. Resultatet af at køre det Cannyfiltrerede billed igennem en Houghtransform er en liste med linier, repræsenteret ved (θ_i, ρ_i) par, og sorteret efter godhed.

OpenCV har to implementationer af Houghtransformationer: En standard som ovenfor beskrevet, og en probabilistisk. Den probalististiske returnerer en liste af linier, repræsenteret ved liniernes endepunkter $(x_{i1}, y_{i1}), (x_{i2}, x_{i2})$ istedet for et (θ_i,ρ_i) par som ovenfor.