MonoSLAD: Et system for autonom landing av kameradroner

Masteroppgave i kybernetikk og autonome systemer

Sander Folke-Olsen

Oppgave for graden
Master i Kybernetikk og autonome systemer
60 studiepoeng

Institutt for teknologisystemer
Det matematisk-naturvitenskapelige fakultet

UNIVERSITETET I OSLO

Våren 2023

MonoSLAD: Et system for autonom landing av kameradroner

Masteroppgave i kybernetikk og autonome systemer

Sander Folke-Olsen

MonoSLAD: Et system for autonom landing av kameradroner

http://www.duo.uio.no/

Trykk: Reprosentralen, Universitetet i Oslo

Sammendrag

I denne oppgaven har vi foreslått et system for autonom deteksjon av trygge landingsplasser. Systemet er basert på visuell navigasjon og tett dybdeestimering. Systemet bruker tett normalestimering på dybdebildene for å finne de relevante geometriske egenskapene vi trenger for å bestemme om et område kan landes i. Egenskapene fusjoneres så inkrementelt for å bygge og forbedre et kart som vi bruker til å identifisere landingsplasser. Systemet er testet på datasett fra Mid-Air og Euroc, med tilhørende diskusjon av resultatene.

Innhold

Figurer

Tabeller

Forord

Vil takke min veileder Trym Vegard Haavardsholm for å ha delt av sin uendelige kunnskap.

Vil også rette en takk til min samboer Maja for å ha matet meg under arbeidet av denne oppgaven.

Kapittel 1

Innledning

Kameradroner er ubemannede, rotordrevne luftfartøy med et påmontert kamera. Bruk av kameradroner er i dag svært utbredt, til både rekreasjonelle, industrielle og militære formål. De kan for eksempel brukes til personsøk i redningsaksjoner [19], overvåkning av solcelleparker [15] og til å levere pakker [8].

I de aller fleste tilfeller blir de fjernstyrt av en dronepilot, men dette skaper noen begrensninger. Blant annet må piloter lønnes, og piloten må ha en åpen kommunikasjonslinje med dronen for å kunne styre den. Dette begrenser antall droner som kan flys samtidig, rekkevidden til dronen, og gjør dronen sårbar for jamming. Dersom dronene trygt og autonomt kan utføre oppgaven de er satt til å gjøre vil dette kunne åpne for nye og flere bruksområder enn det droner har i dag. En autonom dronesverm vil kunne dekke store områder uten behov for personell, noe som er gunstig for personssøk og overvåkningsoppgaver. Man vil heller ikke trenge en pilot per leveringsdrone, noe som ville påvirket kostnadsskaleringen av et droneleveransesystem. Dette åpner også for utforsking og kartlegging av andre planeter fra lufta, hvor direkte kontroll fra jorden ikke er mulig på grunn av signalforsinkelsen som følger av avstanden [9]. For å kunne oppnå en slik grad av autonomi, kreves det blant annet at dronen kan navigere uten bruk av eksterne signaler, og at den har evne til å lande trygt når den ankommer destinasjonen, eller når forholdene tilsier at videre flyvning er for risikabelt.

Vi ønsker å legge til rette for autonom landing ved å utnytte bildene fra dronens påmonterte kamera. For å kunne bestemme hvor det er trygt å lande kreves det at vi har kjennskap til strukturene i dronens omgivelser, og dronens bevegelse i omgivelsene. Feltet som omhandler uttrekning av denne typen informasjon fra bilder kalles maskinsyn. For å navigere bruker vi en kombinasjon av visuell- og treghetsnavigasjon; ved å benytte bilder, akselerasjonsmålinger og gyroskopmålinger kan vi estimere hvordan dronen har beveget seg. For å finne trygge landingsplasser lager vi en 3D-modell av omgivelsene ved bruk av tett 3D-rekonstruksjon, som vil si at vi forsøker å gjenskape de avbildede 3D-strukturene ved å estimere dybder i store deler av bildet. Dette lar oss bruke romlig informasjon om omgivelsene for å

bestemme om et område kan landes i.

1.1 Mål

Målet med oppgaven er å lage et maskinsynsystem for autonom landing basert på tett 3D rekonstruksjon av omgivelsene, som kan kjøres i sanntid på en kameradrone med et monokulært kameraoppsett. Systemet bør korrekt identifisere tilstrekkelig mange trygge landingsplasser, og det bør ikke identifisere landingsplasser som trygge dersom de ikke er det. For å kunne korrekt klassifisere landingsplasser må systemet bygge en nøyaktig representasjon av omgivelsene i form av en 3D-modell ved å produsere nøyaktige dybdeestimater.

Systemet vil bli evaluert ved å teste systemet på datasett hvor omgivelsene og dybdene per piksel er kjente, slik at vi kan regne ut feilen i dybdeestimatene direkte, og bestemme presisjon og sensitiviteten av landingsplassdetektoren.

1.2 Struktur

Oppgaven er strukturert som følger: i kapittel 2 går vi gjennom de matematiske verktøyene og algoritmene vi bruker for å lage systemet. Først går vi gjennom deteksjon av trygge landingsplasser, hvordan vi definerer dette og hvordan dette kan kvantifiseres. Så kommer vi med en beskrivelse av perspektivkameramodellen, og beskriver hvordan kamerageometrien kan utnyttes til å utlede romlig informasjon fra bilder. Deretter gir vi en kort innføring i visuell navigasjon, som etterfølges av en redegjørelse for tett 3D-rekonstruksjon og hvordan vi har valgt å håndtere dette. I kapittel 3 går vi gjennom vår metode og oppbyggingen av systemet. Dette starter med en oversikt over de forskjellige komponentene i systemet, etterfulgt av en gjennomgang av hvordan vi gjør dybdeestimering og postprosessering av dybdebildet, før vi til slutt beskriver vi hvordan vi klassifiserer landingsplasser. I kapittel 4 gjør vi rede for datasettene vi har brukt, eksperimentene vi har gjort med systemet, og diskuterer resultatene vi har fått. Til slutt konkluderer vi oppgaven i kapittel 5.

Kapittel 2

Bakgrunn

For å kunne bruke bilder fra dronens kamera til å navigere og finne landingsplasser, trenger vi noen matematiske verktøy som lar oss beskrive de fysiske prosessene som inngår i dette. I dette kapittelet vil vi derfor gjøre rede for de matematiske modellene og algoritmene vi benytter.

2.1 Deteksjon av trygg landingsplass (SLAD)

Dersom man ønsker å lande innenfor et større område hvor man har begrensede forhåndskunnskaper om topografien, for eksempel en nødlanding på grunn av værforhold, kreves det at dronen har evne til å finne en egnet landingsplass på egenhånd. Dette problemet kalles trygg landingssone-deteksjon (SLZ detection) [23] eller trygt landingsområdedeteksjon (SLAD) [24]. Generelt går dette ut på å automatisk finne egnede landingsområder ved bruk av dronens sensorer, og bestemme posisjonen til disse områdene. Vi ønsker å demonstrere et system for deteksjon av landingsplasser i ukjent terreng som kan kjøres ombord på en drone med et monokulært (enkeltkamera) oppsett uten eksterne signaler.

Tidligere arbeider på synsbaserte autonome landingssystemer har varierende antagelser og begrensninger. I en rekke av systemene er målet å detektere tidligere kjente landingsplasser, enten ved deteksjon av syntetiske markører på landingsplassen, eller ved å bruke naturlige landemerker som referanse [30]. Når det kommer til deteksjon av landingsplasser i ukjente omgivelser, har det tidligere blitt gjort med et monokulært oppsett, men hvor navigasjonen og bildebehandlingen har blitt gjort på en bakkestasjon [31]. Det er også noen som har laget et tett digitalt høydekart (DEM, Digital Elevation Map) basert på bilder, og bestemt mulige landingssoner ut fra dette [28], men navigert ved hjelp av GPS. Vår landingsplassdeteksjon er inspirert av [25], hvor de detekterte landingsplasser i en tett punktsky som ble bygget ved bruk av Lidar.

Vi definerer et område som landbart dersom det har følgende egenskaper:

- Stort nok
- Tilstrekkelig flatt

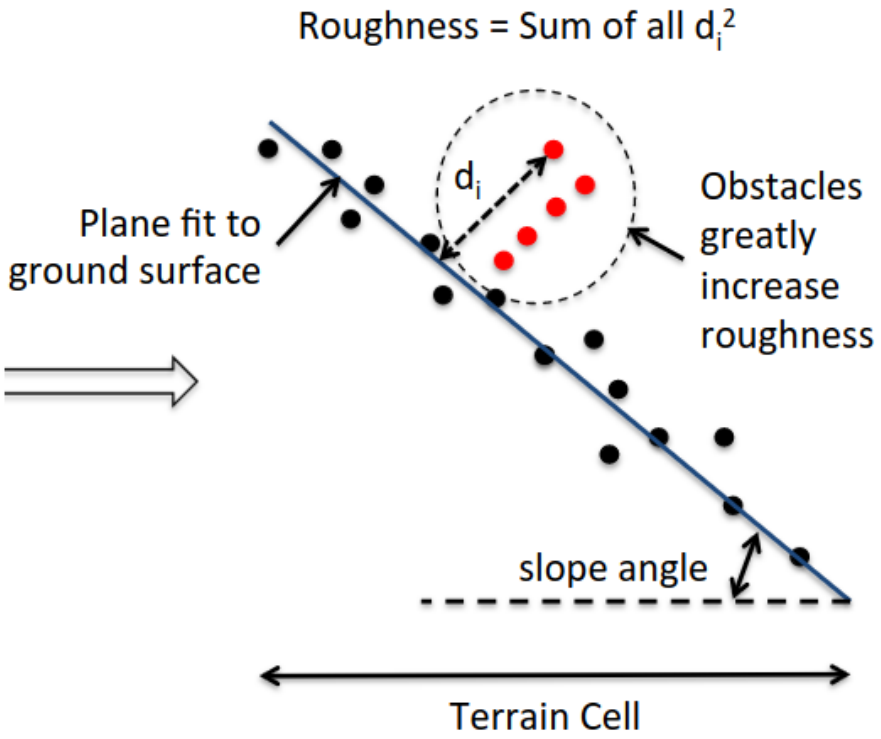
- Lav helling

Dersom vi kan kvantifisere disse egenskapene vil vi kunne vurdere om et område er landbart eller ikke. Vi vil her komme med vårt forslag til kvantifisering av egenskapene ved å betrakte et område på en tenkt overflate.

Vi definerer hellingen i et punkt $\mathbf{p}$ på overflaten som

$$h = \mathbf{n}_p \cdot \mathbf{z_w}, \tag{2.1}$$

der $\mathbf{n}_p$ er normalvektoren til punktet, med positiv retning ut av overflaten, og $\mathbf{z_w}$ er enhetsvektoren som peker mot gravitasjonsretningen. Flathet i $\mathbf{p}$ definerer vi som hvor godt overflaten i et nabolag rundt $\mathbf{p}$ passer til et plan. Størrelseskravet er avhengig av fartøyet og må være kjent på forhånd. Dette kvantifiseres i form av et kvadrat med kjent sidelengde S.



Figur 2.1: Eksempel på egenskaper som kan trekkes ut av punktsky. Fra [25]

Vi kan da finne egnede landingsplasser ved å finne kvadrater med sidelengde S langs bakken som ikke inneholder punkter med mye helling eller lav flathet. Både normalvektoren og flatheten kan estimeres i et punkt ved bruk av integralbilder [11] dersom vi har en ordnet punktsky, altså en punktsky som er organisert som et dybdebilde. Vi har derfor behov for å estimere et tett dybdebilde.

2.1.1 Normal- og krumningsestimering med integralbilder

Et integralbilde I_d som korresponderer med bildet d defineres slik at pikselen $I_d(m,n)$ er summen av pikselverdiene i rektangelet mellom $d(0,0)$ og $d(m,n)$

$$I_d(m,n) = \sum_{i=0}^{m}\sum_{j=0}^{n} d(i,j) \tag{2.2}$$

Med et integralbilde kan man regne ut gjennomsnittet av et kvadrat med sentrum (m,n) og indre radius r i det opprinnelige bildet ved

$$\begin{aligned} S(I_d,m,n,r) = \quad & \tfrac{1}{4r^2}(I_d(m+r,n+r) \\ & -I_d(m-r,n+r) \\ & -I_d(m+r,n-r) \\ & +I_d(m-r,n-r)) \end{aligned} \tag{2.3}$$

For å estimere normalen til en piksel i et dybdebilde kan man gjøre en plantilpasning i et område rundt pikselen ved å regne ut egenverdiene til kovariansmatrisen i området. Kovariansmatrisen kan beregnes hurtig ved å konstruere 9 integralbilder [22]. Tre av dem, I_x, I_y, I_z består av henholdsvis x, y og z-verdiene til punktene, og de resterende 6 er alle de mulige kombinasjonene av punktkoordinater I_{xx}, I_{xy}, I_{xz}, I_{yy}, I_{yz}, I_{zz}, der I_{ab} er den elementvise multiplikasjonen av integralbildene I_a og i_b.

Kovariansmatrisen Σ_p til et punkt $\mathbf{p} = (m,n)$ kan da regnes ut som

$$\Sigma_p = \begin{bmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix}^T \tag{2.4}$$

der

$$c_{ab} = S(I_{ab},m,n,r) \tag{2.5}$$

Fra denne finner vi normalvektoren $\mathbf{n}_p$ som egenvektoren med lavest tilhørende egenverdi.

Flatheten kan defineres som det motsatte av krumningen i punktet. Fra samme kovariansmatrise kan vi finne et mål på krumningen k ved

$$k = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{2.6}$$

der λ_0 er den minste egenverdien. En krumningsverdi på 0 tilsier at alle punktene i skyen ligger på et plan, og en krumningsverdi på $\frac{1}{3}$ tilsier at ethvert plan gjennom tyngdepunktet til skyen vil passe like godt.

2.2 Euklidske transformer

En avbildningsprosess er en matematisk prosess som tar inn en mengde i et domene A og gir ut en mengde i domene B, slik at hver mengde i A

samsvarer med en bestemt mengde i B. En avbildning f av elementet $x \in A$ til $y \in B$ skrives

$$f : x \in A \mapsto y \in B. \tag{2.7}$$

En Euklidsk transform $\mathbf{T}_{ab} \in SE(3)$ er en avbildningsprosess som representerer orienteringen og translasjonen mellom to koordinatsystemer, der $_{ab}$ indikerer at $\mathbf{T}_{ab}$ transformerer punkter gitt i koordinatsystem b til koordinatsystem a. Alternativt kan $\mathbf{T}_{ab}$ tolkes som posituren til b gitt i a, der positur er orientering og translasjon. Vi kan representere avbildningen som en 4x4 matrise på formen

$$\mathbf{T}_{ab} = \begin{bmatrix} \mathbf{R}_{ab} & \mathbf{t}_{ab} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{2.8}$$

der $\mathbf{t}_{ab} \in \mathbb{R}^3$ er translasjonen og $\mathbf{R}_{ab} \in SO(3)$ er orienteringsmatrisen. At $\mathbf{R} \in SO(3)$ vil si at den har følgende egenskaper:

$$\mathbf{R}^T\mathbf{R} = \mathbf{I} \tag{2.9}$$

$$det(\mathbf{R}) = 1. \tag{2.10}$$

$\mathbf{T}_{ab}$ transformerer homogene vektorer mellom koordinatsystemene gjennom matrise-vektor multiplikasjon. En homogen vektor kan konstrueres fra en kartesisk vektor ved å utvide vektoren med elementet 1.

$$\mathbf{x} \in \mathbb{R}^n \mapsto \tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \in \mathbb{P}^n \tag{2.11}$$

En homogen vektor avbildes tilbake til en kartesisk vektor ved

$$\tilde{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ w \end{bmatrix} \in \mathbb{P}^n \mapsto \mathbf{x} = \mathbf{y}/w \in \mathbb{R}^n. \tag{2.12}$$

Dette lar oss finne $\mathbf{x}^a \in \mathbb{R}^3$, altså punktet $\mathbf{x}$ gitt i koordinatsystemet a, fra $\mathbf{x}^b \in \mathbb{R}^3$ som produktet

$$\tilde{\mathbf{x}}^a = \mathbf{T}_{ab}\tilde{\mathbf{x}}^b. \tag{2.13}$$

Se også figur 2.2.

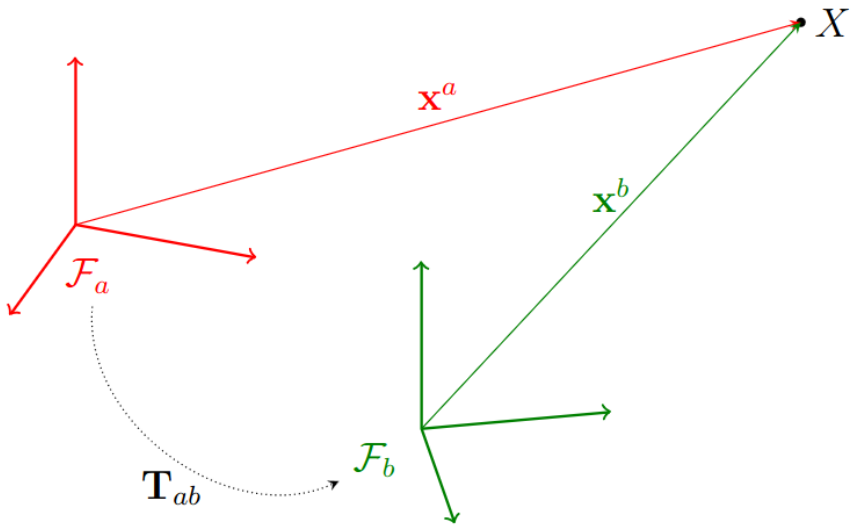
2.3 Kamerageometri

Generelt kan vi beskrive et kameras avbildningsprosess som

$$\mathbf{u} = \pi(\mathbf{x}^c) \tag{2.14}$$

der $\mathbf{u} \in \mathbb{R}^2$ er pikselkoordinatene, π er kameramodellen som avbilder punkter fra 3D-verdenen sett fra kameraets referanseramme til 2D, og $\mathbf{x}^c \in \mathbb{R}^3$ er et punkt gitt i kameraets koordinatsystem.

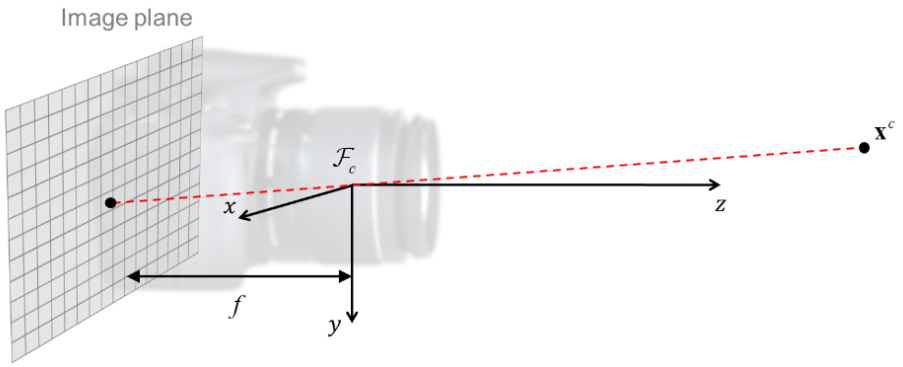
Perspektivkameramodellen (figur 2.3) er en matematisk modell som bruker sentralprojeksjon til å avbilde punkter i 3D-verdenen til punkter i 2D-bildet. Sentralprojeksjon vil si at alle punkter i bildeplanet kan assosieres



Figur 2.2: Et punkt X representert som vektorene $\mathbf{x}^a$ i koordinatsystem $\mathcal{F}_a$ og $\mathbf{x}^b$ i koordinatsystem $\mathcal{F}_b$. $\mathbf{T}_{ab}$ er transformen som lar oss representere enhver $\mathbf{x}^b$ i koordinatsystem $\mathcal{F}_a$ gjennom ligning 2.13. $\mathbf{T}_{ab}$ er også posituren til koordinatsystem $\mathcal{F}_b$ sett fra koordinatsystem $\mathcal{F}_a$. Illustrasjon fra [13].

med en stråle som går fra punktet i bildeplanet, gjennom origo, og gjennom 3D-punktet som avbildes. Sammen med en forvrengingsmodell kan perspektivkameramodellen brukes på de fleste moderne kameraer med god nøyaktighet.

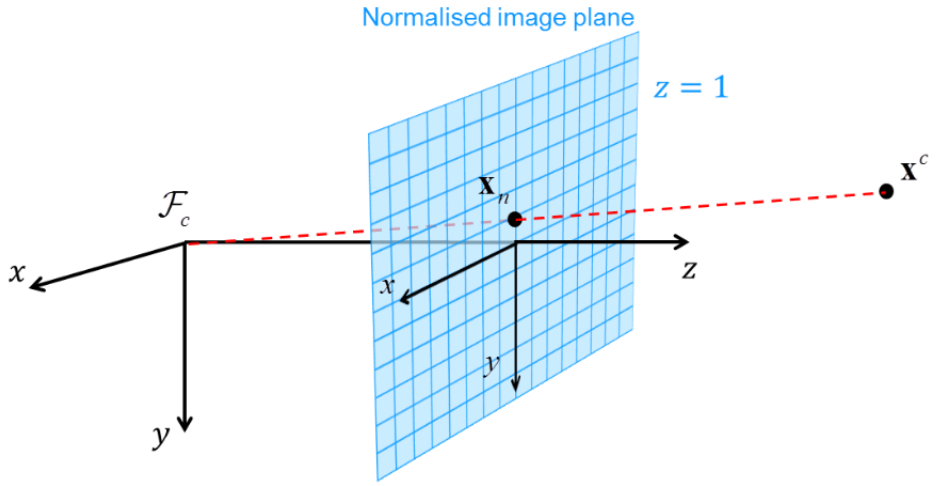
Kameraet representeres ved en 3D referanseramme $\mathcal{F}_c$, med origo i kameraets projektive sentrum. X-aksen peker til høyre, y-aksen peker ned, og z-aksen peker frem langs kameraets optiske akse. Avbildningsprosessen er modellert som en sentralprojeksjon på bildeplanet i $z = -f$ der f er fokallengden [13].



Figur 2.3: Perspektivkameramodellen, bilde tatt fra [13]

Siden denne projeksjonen gir oss bilder som er snudd opp ned, er det mer praktisk å sette bildeplanet foran det projektive sentrum. Det er også

praktisk å definere det normaliserte bildeplanet i $z = 1$, slik at dette representerer et ideelt kamera med fokallengde 1. Punktet hvor z-aksen til $\mathcal{F}_c$ krysser bildeplanet kalles prinsipalpunktet. Dette lar oss beskrive avbildningsprosessen i et fiksert bildeplan, uavhengig av den kamera-avhengige fokallengden.



Figur 2.4: Det normaliserte bildeplanet, tatt fra [13]

Vi kan nå projisere 3D-punktet $\mathbf{x}^c$ på det normaliserte bildeplanet med den homogene perspektivprojeksjonsmatrisen $\mathbf{\Pi}_0$ ved

$$\tilde{\mathbf{x}}_n = \mathbf{\Pi}_0 \tilde{\mathbf{x}}^c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\mathbf{x}}^c = \mathbf{x}^c \tag{2.15}$$

Normalisert gir dette oss

$$\check{\mathbf{x}}_n = \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} = \begin{bmatrix} x^c/z^c \\ y^c/z^c \\ 1 \end{bmatrix} = \frac{1}{z^c}\mathbf{x}^c \tag{2.16}$$

Selve bildet er representert ved referanserammen $\mathcal{F}_i$. $\mathcal{F}_i$ har piksler som enheter, og spenner det normaliserte bildeplanet. Origo defineres som bildets øvre venstre hjørne, u-aksen peker til høyre og v-aksen peker ned. Avbildningen mellom normaliserte bildekoordinater og pikselkoordinater er gitt ved den affine transformasjonen

$$\tilde{\mathbf{u}} = \mathbf{K}\tilde{\mathbf{x}}_n \tag{2.17}$$

der

$$\mathbf{K} = \begin{bmatrix} f_u & s_\theta & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.18}$$

$\mathbf{K}$ samler alle parametere som definerer et spesielt kamera, og kalles ofte kalibreringsmatrisen. Parametrene er fokallengden i x-retning f_u gitt i

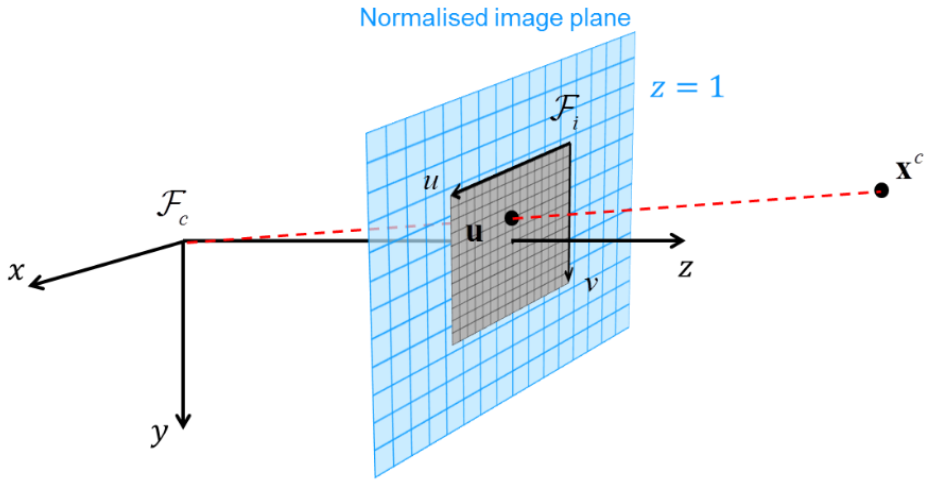
piksler, fokallengden i y-retning f_v, u og v-koordinatet til prinsipalpunktet c_u og c_v. s_θ er en skjevhetsparameter som beskriver hvordan bildesensoren er orientert i forhold til fokalplanet, og tilnærmes ofte som 0 i moderne kameraer.

Siden $\mathbf{K}$ ikke påvirker det tredje elementet i $\tilde{\mathbf{x}}_n$ vil normalisering bevares, og vi har

$$\check{\mathbf{u}} = \mathbf{K}\check{\mathbf{x}}_n = \mathbf{K}\frac{1}{z^c}\mathbf{x}^c. \tag{2.19}$$

Ved å sette dette sammen får vi kameramodellen

$$\mathbf{u} = \pi_p(\mathbf{x}^c) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{K}\frac{1}{z^c}\mathbf{x}^c \tag{2.20}$$



Figur 2.5: Kartleggingen fra 3D-punkt til bildekoordinater, tatt fra [13]

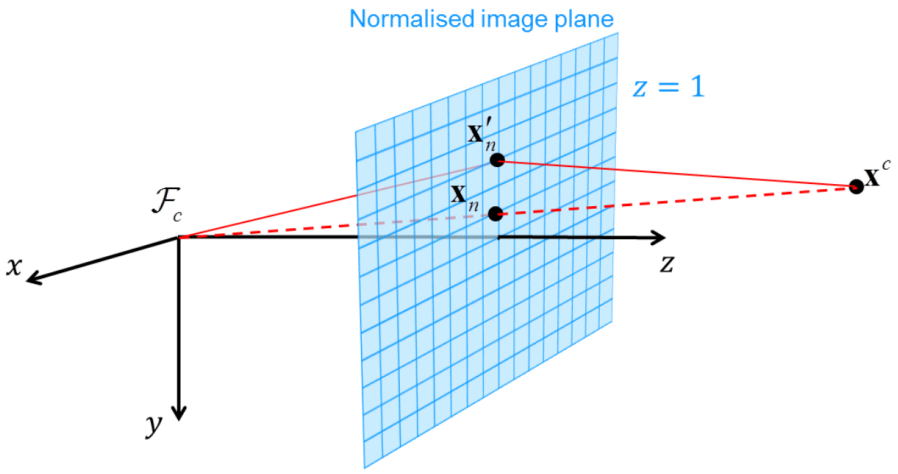
Det er sjeldent ekte kamera passer perfekt med perspektivkameramodellen. Moderne kamera bruker linser for å fokusere lys på bildesensoren, og dette medfører forvrengingseffekter som følge av blant annet linsenes geometri og bildesensorens orientering i forhold til linsen [29]. Derfor er det ofte behov for å kombinere perspektivkameramodellen med en bildeforvrengningsmodell.

En mye brukt bildeforvrengningsmodell er den radiell-tangensielle forvrengningsmodellen. Her antas det at bildeforvrengningen i hovedsak består av en ulineær radiell komponent (figur 2.7) og en ulineær tangensiell komponent (figur 2.8). Forvrengningen i figur 2.6 kan for eksempel bli modellert som

$$\mathbf{x'}_n = \begin{pmatrix} x'_n \\ y'_n \end{pmatrix} = \begin{pmatrix} x_n(1 + k_1 r_n^2 + k_2 r_n^4) + 2p_1 x_n y_n + p_2(r_n^2 + x_n^2) \\ y_n(1 + k_1 r_n^2 + k_2 r_n^4) + p_1(r_n^2 + y_n^2) + 2p_2 x_n y_n \end{pmatrix}, \tag{2.21}$$

hvor

$$r_n^2 = x_n^2 + y_n^2. \tag{2.22}$$



Figur 2.6: Eksempel på bildeforvrenging. Punktet $\mathbf{x}^c$ skal avbildes i $\mathbf{x}_n$ ifølge perspektivkameramodellen, men er i realiteten avbildet i $\mathbf{x}'_n$. Fra [13].

Her er k_1, k_2 koeffisentene til den radielle forvrengningen, og p_1, p_2 koeffisentene til den tangensielle forvrengingen. Det finnes også varianter av denne forvrengningsmodellen som tar med høyere ordens ledd, og varianter som bruker rasjonelle polynomer. Vi kommer likevel ofte langt med 4 koeffisenter.

2.4 Epipolargeometri

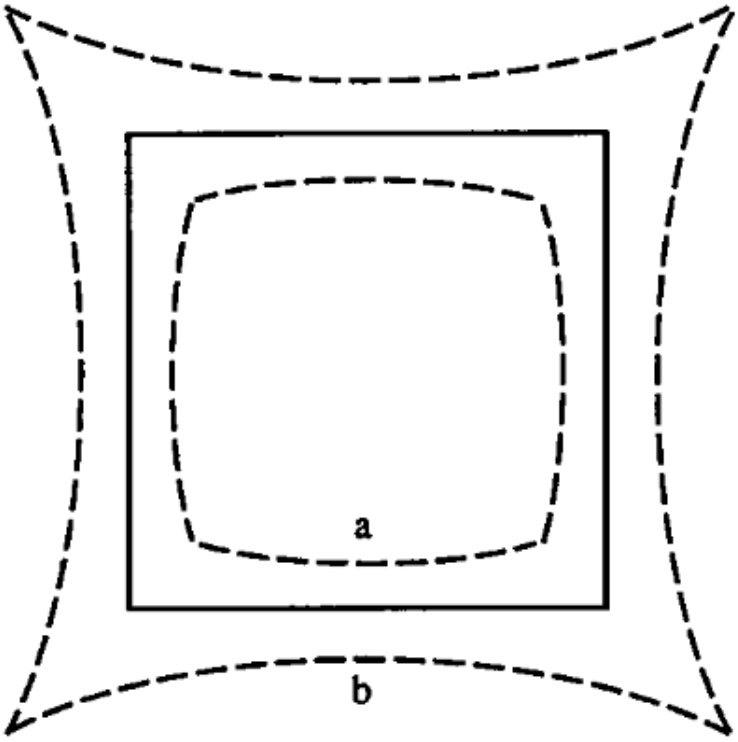
Når vi har flere bilder av samme scene oppstår det en geometrisk begrensning på hvor 3D-punkter kan avbildes i hvert av bildene. Dette lar oss estimere 3D-posisjonen til pikslene og bevegelsen mellom bildene med buntjustering (bundle adjustment), hvor vi behandler det som et begrenset optimeringsproblem. For å få rask konvergens i optimeringen er det ønskelig å starte med et estimat som ligger i nærheten av den optimale løsningen.

I tilfellet med 2 bilder oppstår den geometriske begrensningen som epipolarbegrensningen. Om bildene er tatt med kameraer som kan modelleres med perspektivkameramodellen får epipolarbegrensningen en særlig enkel form som vi skal vise her.

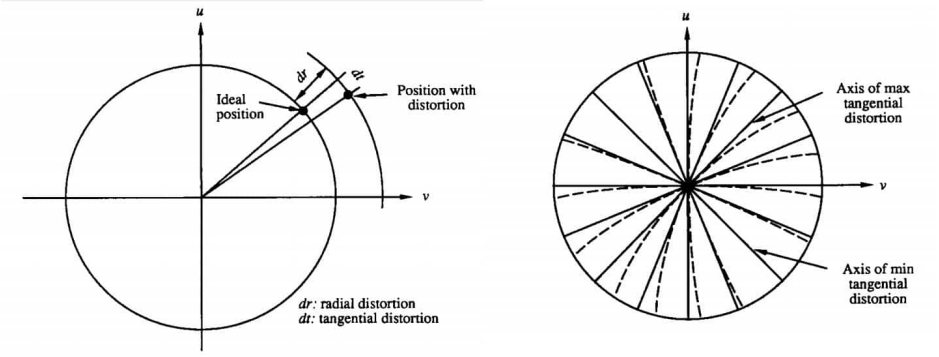
Anta at vi har to perspektivkameraer med referanserammene $\mathcal{F}_a$ og $\mathcal{F}_b$. Referanserammene er relatert gjennom den relative transformasjonen

$$\mathbf{T}_{ab} = \begin{bmatrix} \mathbf{R}_{ab} & \mathbf{t}^a_{ab} \\ 0 & 1 \end{bmatrix} \tag{2.23}$$

De to kamerasentrene $\mathbf{t}_a$ og $\mathbf{t}_b$, og punktet $\mathbf{x}$ som avbildes, spenner et plan som kalles epipolarplanet. Linjen som kan trekkes mellom kamerasentrene kalles baselinjen. Epipolene er der hvor baselinjen krysser de normaliserte bildeplanene, eventuelt avbildningen av kamerasenter $\mathbf{t}_a$ i kamera b, og omvendt. Epipolarlinjene er der epipolarplanet krysser bildeplanene. Konsekvensen av dette er at punktet $\mathbf{x}$ avbildet i a, må være avbildet på den



Figur 2.7: Illustrasjon av radiell forvrengning. Den gjennomtrukkede firkanten er uten forvrengning, a er et eksempel på negativ radiell forvrengning og b er et eksempel på positiv radiell forvrengning. Fra [29].

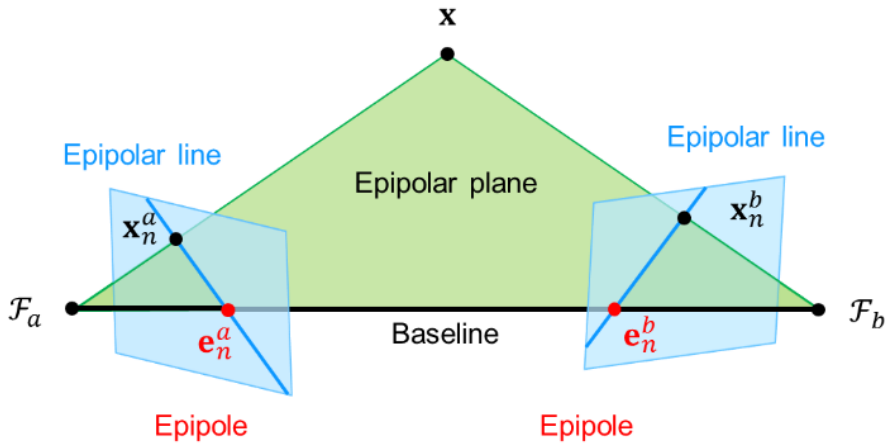


Figur 2.8: Til venstre: Illustrasjon av radiell-tangensiell forvrengning. Til høyre: Illustrasjon av tangensiell forvrengning. Fra [29].

korresponderende epipolarlinjen i b. Det er dette som er epipolarbegrensningen [13].

Matematisk kan dette uttrykkes ved essensiellmatrisen $\mathbf{E} \in \mathbb{R}^{3x3}$ i det normaliserte bildeplanet. En punktkorrespondanse må tilfredstille ligningen

$$\tilde{\mathbf{x}}^{aT}\mathbf{E}_{ab}\tilde{\mathbf{x}}^{b} = 0 \tag{2.24}$$



Figur 2.9: Epipolargeometrien, tatt fra [13]

der

$$\mathbf{E}_{ab} = [\mathbf{t}_{ab}^a]_\times \mathbf{R}_{ab}, \tag{2.25}$$

hvor $[\mathbf{t}]_\times$ indikerer matriserepresentasjonen av kryssproduktet

$$[\mathbf{t}]_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}. \tag{2.26}$$

Siden linjen $ax + by + c = 0$ kan skrives på homogen form som

$$\tilde{\mathbf{x}}^T \tilde{\mathbf{l}} = 0, \qquad \tilde{\mathbf{l}} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \in \mathbb{P}^2 \tag{2.27}$$

ser vi fra 2.24 at

$$\tilde{\mathbf{l}}^a = \mathbf{E}_{ab} \tilde{\mathbf{x}}^b \tag{2.28}$$

er epipolarlinjen i det normaliserte bildeplanet til $\mathcal{F}_a$ som korresponderer til punktet $\tilde{\mathbf{x}}^b$ i $\mathcal{F}_b$.

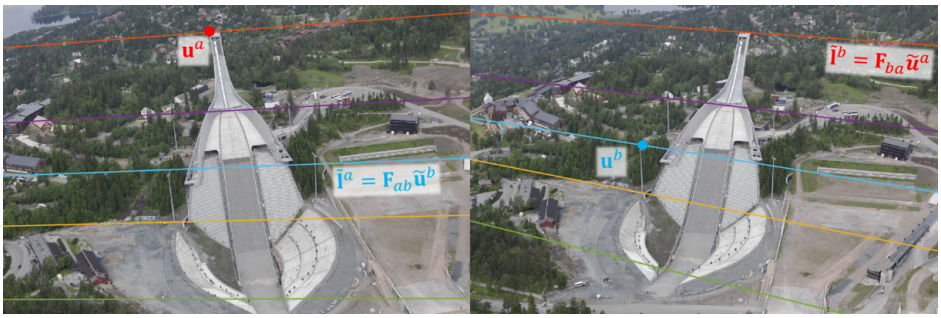
Epipolarbegrensningen utvides også naturlig til pikselkoordinater gjennom kalibreringsmatrisene $\mathbf{K}_a$ og $\mathbf{K}_a$:

$$\tilde{\mathbf{u}}^a \mathbf{F}_{ab} \tilde{\mathbf{u}}^b = 0, \tag{2.29}$$

hvor fundamentalmatrisen $\mathbf{F}_{ab}$ er gitt ved

$$\mathbf{F}_{ab} = \mathbf{K}_a^{-T} \mathbf{E}_{ab} \mathbf{K}_b^{-1}. \tag{2.30}$$

Vi kan altså begrense søket etter punktkorrespondanser til epipolarlinjen dersom vi kjenner den relative posituren mellom bildene (figur 2.10). Dette går også andre veien; om vi kjenner minst 5 punktkorrespondanser kan vi estimere essensiellmatrisen [20], og fra denne kan vi finne den relative posituren mellom bildene, opp til skala. Dette kan vi bruke for å finne et initielt estimat til buntjustering.



Figur 2.10: Eksempel på bildepar med epipolarlinjer. Fra [13].

2.5 Visuell navigasjon

Moderne luftfartøy navigerer typisk med bruk av en kombinasjon av satellittnavigasjon, treghetsnavigasjon og diverse metoder basert på radiokontakt med en bakkestasjon. For mindre luftfartøy som droner er det særlig satellittnavigasjon og treghetsnavigasjon som benyttes. Med satellittnavigasjon estimerer man fartøyets globale positur ved bruk av avstandsmålinger til satellitter med kjente posisjoner. Dette gir konsekvente globale estimater, men kan gi stor lokal feil, særlig i høyde [26]. Med treghetsnavigasjon estimerer man fartøyets positur ved å integrere opp akselerasjon- og gyroskopmålinger. Dette gir lokalt konsekvente estimater, men vil gi avdrift i positur over tid, da man også integrerer støyen i målingene. Det er også mange tilfeller hvor satellittnavigasjon er upålitelig eller ikkeeksisterende, som mellom høye bygninger, i dype daler, på en annen planet, eller at signalet blir jammet. Dette har vært noe av motivasjonen bak utviklingen av visuelle navgiasjonssystemer [10].

I visuell navigasjon vil man estimere et fartøys positur ved bruk av bilder fra et eller flere kamera som er montert på fartøyet. Siden slike systemer ikke er avhengig av eksterne signaler egner det seg godt i tilfellene nevnt ovenfor.

I vårt tilfelle er vi avhengig av nøyaktige positurestimater mellom bildene for å kunne gjøre 3D rekonstruksjon, og dermed identifisere landingsplasser. Ved å estimere dronens positur visuelt håper vi å oppnå dette. Siden vi både navigerer og estimerer visuelt, vil vi forhåpentligvis også kunne få en lokal overensstemmelse mellom hvor dronen befinner seg og hvor landingsplassen er.

2.5.1 Visuell SLAM

For å navigere i tidligere ukjente omgivelser må vi både bygge et kart og samtidig lokalisere oss i kartet ved bruk av påmonterte sensorer. Dette problemet kalles SLAM (Samtidig lokalisering og kartlegging). Om vi bruker kamera til dette kalles det ofte VSLAM (Visuell SLAM).

VSLAM-problemet kan løses med buntjustering, hvor man minimerer reprojeksjonsfeilen av punkter som er observert i flere bilder, med hensyn

på kameraenes positur og punktenes 3D-posisjon. Vi kan identifisere slike nøkkelpunkter ved å finne pikselkorrespondanser mellom bildene. Dette kan vi gjøre ved å finne piksler med unike egenskaper i hvert bilde. Dette er typisk hjørner eller blobber. Til hver slik piksel gjør man en egenskapsuttrekning for å gi dem en deskriptor, som er en kvantifisering av egenskapene til pikselen. Ved å sammenligne deskriptorene til pikslene kan vi finne pikselkorrespondanser. Man vil ofte få noen dårlige korrespondanser, og derfor brukes ofte RANSAC [4] med 5-punktsalgoritmen for å fjerne utliggere.

Vi definerer reprojeksjonsfeilen for et punkt j i kamera i som

$$\varepsilon_{ij} = ||\pi_i(\mathbf{T}_{wc_i}^{-1}\tilde{\mathbf{x}}_j^w) - \mathbf{u}_j^i||^2, \tag{2.31}$$

altså avstanden mellom de kjente pikselkoordinatene $\mathbf{u}_j^i$ og pikselkoordinatene til projeksjonen av et antatt 3D-punkt $\mathbf{x}_j^w$. Den totale reprojeksjonsfeilen over alle punkter og kamera blir da

$$\varepsilon = \sum_i \sum_j ||\pi_i(\mathbf{T}_{wc_i}^{-1}\tilde{\mathbf{x}}_j^w) - \mathbf{u}_j^i||^2. \tag{2.32}$$

Gitt initielle estimater for positurene og punktene (for eksempel fra essensiellmatrisen funnet ved 5-punktsalgoritmen) kan dette minimaliseres med hensyn på både positurene og punktene med en ikke-lineær minste kvadraters løser som for eksempel Levenberg-Marquardt.

Buntjustering løser VSLAM-problemet, men regnetiden vil øke etterhvert som antall positurer og 3D-punkter øker. For å gjøre dette i praksis er det derfor nødvendig å ta i bruk strategier for å begrense utregningstiden. En måte å gjøre dette på er å skille mellom kortsiktig og langsiktig sporing. Kortsiktig sporing kan gjøres ved buntjustering i et lokalt vindu av bilder som observerer felles punkter. Dette vil over tid føre til avdrift i estimatene. Derfor gjør man også langsiktig sporing ved å sammenligne nylig observerte punkter med punkter som er observert lengre tilbake i tid. Om man finner slike korrespondanser kan man gjøre en buntjustering over alle positurer og punkter, og på den måten korrigere for avdriften.

Dersom man kun benytter seg av et kortsiktig sporing kalles det ofte VO (visuell odometri), og kan ses på som begrenset VSLAM. Det er i hovedsak to fremgangsmåter for VO, egenskapsbasert og direkte visuell odometri. Det finnes også hybride tilnærminger som for eksempel i SVO (Semi-direct visual odometry) [6], som vi har brukt i dette prosjektet.

I egenskapsbaserte metoder bruker man reprojeksjonsfeilen som beskrevet tidligere. I direkte metoder bruker man ikke deskriptorer for å finne korrespondanser, men pikselintensitetene direkte. Fordelen med dette er at man kan bruke flere av pikslene i bildene (ikke bare hjørner og blobber), og man slipper egenskapsuttrekningen som kan være regnekrevende. Her benyttes ofte en form for fotometrisk feil, hvor man da ser på forskjellen i intensiteter mellom piksler, som

$$\delta_{ij} = ||I_{c_i}(\pi_i(\mathbf{T}_{wc_i}^{-1}\tilde{\mathbf{x}}_j^w)) - I_{c_i}(\mathbf{u}_j^i)||, \tag{2.33}$$

hvor I_{c_i} er intensitetsbildet tatt av kamera i.

2.5.2 SVO

Til navigasjon har vi brukt SVO. Det ble i utgangspunktet utviklet som et system for visuell odometri, men har senere blitt utvidet med mulighet for å fusjonere inn akselerasjon- og gyroskopmålinger, og med SLAM-kapabilitet [17] [14] [7]. SVO foreslår en hybrid tilnærming til VO der de kun gjør egenskapsuttrekning av aktuelle piksler i nøkkelbilder, men ellers bruker pikselintensitetene for å følge pikslene og estimere bevegelse. Målet med dette er å oppnå hurtig bevegelsesestimering, uten å kompromittere på nøyaktigheten av estimatene.

Bevegelsesestimering

I bevegelsesestimeringen antas det at dybden til de aktuelle pikslene er kjent. Bevegelsen estimeres først ved en glissen bilde-til-modell-justering, altså å minimere den fotometriske feilen til piksler som observerer det samme 3D-punktet.

$$\mathbf{T}^*_{kk-1} = \arg\min_{\mathbf{T}_{kk-1}} \sum_{\mathbf{u}\in\bar{\mathcal{R}}^c_{k-1}} \frac{1}{2}||\mathbf{r}_{\mathbf{I}^c\mathbf{u}}(\mathbf{T}_{kk-1})||^2_{\Sigma_I} \tag{2.34}$$

Der $\bar{\mathcal{R}}^c_{k-1}$ er et sett av piksler fra bildet tatt med kamera c fra positur $k-1$, og

$$\mathbf{r}^c_{\mathbf{I}\mathbf{u}}(\mathbf{T}_{kk-1}) = I^c_k(\pi(\mathbf{T}_{cb}\mathbf{T}_{kk-1}\tilde{\mathbf{x}}_\mathbf{u})) - I^c_{k-1}(\pi(\mathbf{T}_{cb}\tilde{\mathbf{x}}_\mathbf{u})) \tag{2.35}$$

er den fotometriske resten, definert av intensitetsdifferansen mellom piksler i bildene I^c_k og I^c_{k-1} som observerer det samme 3D-punktet $\mathbf{x}_\mathbf{u}$. Σ_I er kovariansmatrisen som beskriver måleusikkerheten, og $||\mathbf{r}||^2_{\Sigma_I}$ er den kvadrerte Mahalanobisavstanden

$$\mathbf{r}^T\Sigma_I^{-1}\mathbf{r}. \tag{2.36}$$

For å øke robustheten foreslår de å samle den fotometriske kostnaden i et område rundt hver piksel, og siden dybden til nabopikslene er ukjent approksimeres det at de har samme dybde som senterpikselen. Dette optimiseringsproblemet kan løses effektivt med standarde iterative ikke-lineære minste kvadratsalgoritmer som Levenberg-Marquardt.

For å minimere drift i estimatet er det ønskelig å registrere et nytt bilde opp mot et så gammelt bilde som mulig. Men de har indikert eksperimentelt at robustheten til denne justeringen ikke kan garanteres når avstanden mellom kameraposisjonene øker. Dette løser de ved å avslappe de geometriske begrensningene gitt av reprojeksjonen av 3D-punktene, og utfører en individuell 2D-justering av korresponderende pikselområder. Dette gjøres med hensyn på bildet hvor pikselområdet først ble identifisert, og dermed sikres det at bildet registreres mot et så gammel bilde som mulig. Denne 2D-justeringen genererer en reprojeksjonsfeil, og derfor utfører de en buntjustering med hensyn på reprojeksjonsfeilen for å optimalisere positurestimatet og pikselens 3D-posisjon.

Kartlegging

I forrige seksjon antas det at dybden til de aktuelle pikslene er kjent. Her antas det at bevegelsen mellom bildene er kjent fra bevegelsesestimeringen.

Dybden til hver enkelt piksel estimeres fra flere observasjoner med et rekursivt Bayesisk dybdefilter. Nye dybdefiltre initialiseres i intensitetshjørner og langs gradientkanter når antallet sporede egenskaper faller under en terskel, og dette defineres som et nytt nøkkelbilde. Hvert dybdefilter blir assosiert med et referansenøkkelbilde r, hvor det initielle dybdeestimatet blir initialisert med en høy usikkerhetsverdi. For et sett med tidligere nøkkelbilder, samt hvert følgende bilde med kjent positur, leter de etter pikselområdet langs epipolarlinjen som gir høyest korrelasjon. Fra pikselen med maksimal korrelasjon, trianguleres dypbdemålingen $\tilde{\rho}_i^k$, som brukes til å oppdatere dybdefilteret. Dersom man får tilstrekkelig mange dybdemålinger slik at usikkerheten i dybde er under en viss terskel, initialiseres det et nytt 3D-punkt med den estimerte dybden, som kan brukes i bevegelsesestimeringen.

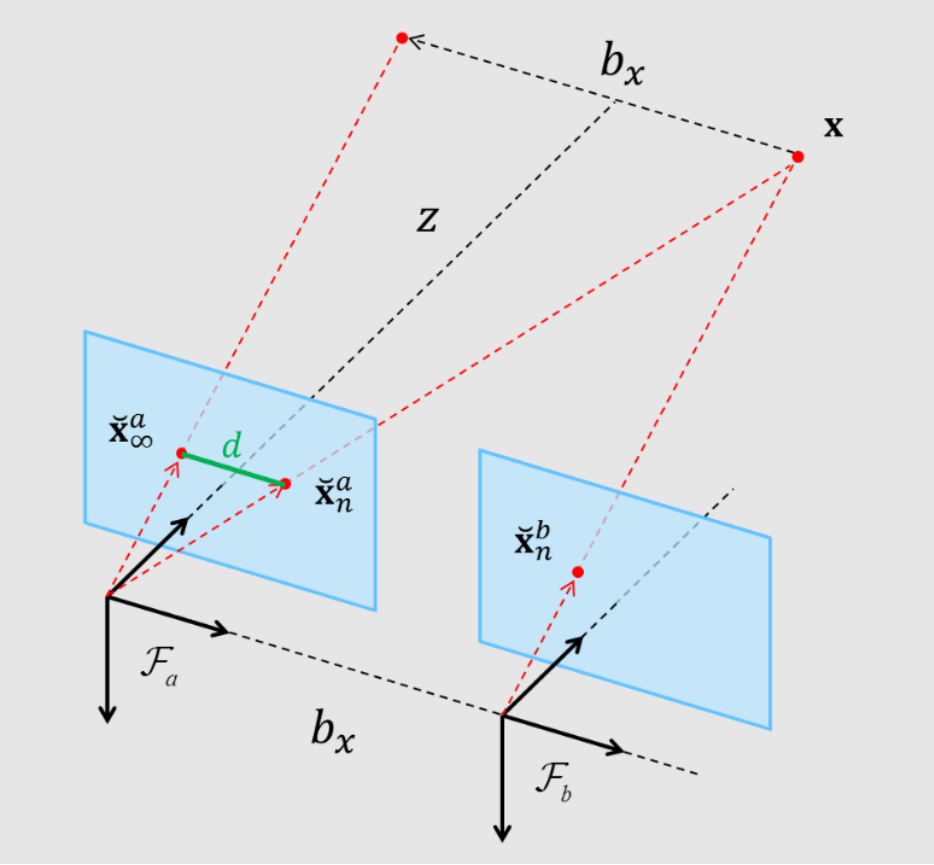
Resultater

Ved å bruke SVO med treghets- og SLAM-utvidelsene får vi ut lokalt og globalt konsekvente positurestimater gitt i et globalt koordinatsystem der tredjeaksen peker mot gravitasjonsretningen. Vi trenger å vite gravitasjonretningen for å regne ut hellingen i et punkt, og vi trenger gode relative positurestimater mellom nærliggende bilder for å estimere dybder. Vi trenger også at positurene er globalt konsekvente for å reprojisere de estimerte hellingene og flathetene på riktig sted i det globale koordinatsystemet.

2.6 Tett 3D rekonstruksjon

Visuelle navigasjonssystemer som SVO lagrer gjerne kartet som en glissen punktsky, og den inneholder kun en begrenset mengde informasjon om 3D-strukturen. Dette egner seg godt til navigasjon, men for å kunne identifisere komplekse 3D-strukturer som landingsplasser er det ønskelig at vi henter ut mer informasjon fra bildene. Vi vil derfor bruke teknikker for tett 3D-rekonstruksjon til å bygge en tett 3D-modell.

Typiske fremgangsmåter for tett 3D-estimering er å bruke et stereokameraoppsett, eller et kamera kombinert med en TOF-sensor (time of flight). I et likerettet og kalibrert stereo-oppsett er det en enkel sammenheng mellom pikselposisjonene og dybden til pikslene. Epipolarlinjen til hver piksel i venstre kamera ligger på samme horisontale linje i høyre kamera. Så ved å søke langs denne linjen kan man effektivt identifisere pikselkorrespondanser. Dette lar oss bestemme dybden til pikselen på grunn av parallakse. Om man tenker seg at kameraet flytter seg til høyre, vil punkter i bildet flytte seg til venstre, og jo nærmere punktet er jo mer vil det flytte seg i bildet



Figur 2.11: Illustrasjon av stereogeometri. Fra [13].

(figur 2.11). Dybden finnes ved

$$z^c = f\frac{b}{d} \tag{2.37}$$

der f er fokallengden til kameraet, b er lengden til baselinjen og d er forflytningen til pikselen mellom venstre og høyre bilde.

Dybdeoppløsningen er proposjonal med kvadratet av dybden, og inverst proposjonalt med fokallengden og baselinjen. Stereokameraer har fast baselinje, og begge kameramodulene må være montert på dronen. Et stereokamera med høy baselinje vil derfor ikke være spesielt kompakt. Et typisk stereooppsett (Intel RealSense D435) oppgir derfor baselinje på 5cm, og ideellt dybdespekter som 0.3-3m. I vårt tenkte bruksområde, utendørs droneflyvning, vil avstandene være mye større enn dette, så et stereooppsett vil trolig ikke være egnet.

En TOF-sensor vil ha noe av det samme problemet. Sensoren sender ut lys, og basert på hvor lang tid lyset har brukt på å reise til 3D-punktet og tilbake regnes det ut hvor langt unna punktet er. Sensoren er avhengig av at nok lys reflekteres tilbake, og trenger kraftigere lys for å få gode målinger på punkter som er langt unna. Dette bruker betydelig mye mer energi enn et kamera, og er derfor heller ikke egnet for store avstander.

For å håndtere forholdene man forventer ved utendørs droneflyvning har vi derfor valgt å benytte et monokulært kamera-oppsett og Plane Sweep Stereo-algoritmen [2], en algoritme for tett dybdeestimering ut fra flere bilder av samme scene. Dette lar oss variere baselinjen ved å plukke ut bilder med ønsket avstand mellom hverandre, og på den måten få god dybdeoppløsning på de relevante avstandene. En ulempe med dette er at det krever gode estimater på dronens bevegelse mellom bilder for å få nøyaktige dybder. Derfor vil vi også fusjonere gjentatte observasjoner av nærliggende 3D-punkter for å gi sikrere estimater, og samtidig begrense veksten av dette tette kartet etterhvert som det kommer inn flere punkter.

2.6.1 Plane Sweep Stereo

Når et plan er avbildet med et perspektivkamera kan denne avbildningen beskrives som en homografi

$$H : \tilde{\mathbf{y}} \in \mathbb{P}^2 \mapsto \tilde{\mathbf{x}} \in \mathbb{P}^2, \tag{2.38}$$

hvor $\tilde{\mathbf{y}}$ er et homogent punkt gitt i et koordinatsystem som spenner planet, og $\tilde{\mathbf{x}}$ er avbildningen av punktet i det normaliserte bildeplanet. Dette er en lineær transform og kan beskrives som en invertibel matrise $\mathbf{H} \in \mathbb{R}^{3x3}$. Siden den er lineær og invertibel vil to avbildninger av det samme planet fra forskjellige perspektiver også relateres via en homografi.

$$\mathbf{H}_{ab} = \mathbf{H}_{ap}\mathbf{H}_{bp}^{-1}, \tag{2.39}$$

hvor a og b er to kamerakoordinatsystemer, og p er planets koordinatsystem.

Plane Sweep Stereo [2] er en algoritme for tett dybdeestimering. Den går ut på å finne dybden til mange piksler i et referansebilde ved å projisere N andre bilder av samme scene, men fra forskjellige positurer, inn i referansebildet for flere hypotetiske dybdeplan Π_m via homografiene som blir indusert av planene. De pikslene som ligner og er på samme posisjon har da sannsynligvis dybden til planet som induserte homografien. Siden vi kjenner positurene til bildene kan vi velge ut bilder som har en fornuftig baselinje i forhold til avstandene til det vi avbilder, og dermed oppnå god dybdeoppløsning også utendørs. Algoritmen blir som følger:

Plane Sweep Stereo-algoritmen

Bestem en minimum d_{min} og en maksimum d_{max} dybde du ønsker å sveipe over.

Bestem hvilke dybdeplan du ønsker å undersøke.

For hvert dybdeplan Π_m:

- Finn homografien som avbilder pikslene i dybdeplanet fra bilde k, for $k \in [1, N]$, inn i bilde 1:

$$\mathbf{H}_{\Pi_m} = \mathbf{K}_k(\mathbf{R}_k^T + \frac{\mathbf{R}_k^T \mathbf{t}_k n_m^T}{d_m})\mathbf{K}_{ref}^{-1} \tag{2.40}$$

  Der $\mathbf{K}$ er de respektive kamerakalibreringsmatrisene, $\mathbf{R}_k$ og $\mathbf{t}_k$ er henholdsvis rotasjonen og translasjonen til kamera k relativt til referansekameraet, $\mathbf{n}_m$ er normalvektoren til planet (i det frontoparallelle tilfellet vil $\mathbf{n}_m^T = [0, 0, 1]$)

- For hver piksel i referansebildet, sammenlign med piksler i avbildningen på samme posisjon, for eksempel med nullgjennomsnitt normalisert krysskorrelasjon (ZNCC, zero mean normalized cross correlation) i et vindu rundt pikselen.

Velg det planet som gir best korrelasjon for hver piksel, og da kan vi finne dybden til piksel (x,y) ved

$$Z_m(x, y) = \frac{-d_m}{[x, y, 1]\mathbf{K}_{ref}^{-T}\mathbf{n}_m} \tag{2.41}$$

Her er det også mulighet for å utelukke usikre korrespondanser basert på korrelasjonene.

PlaneSweepLib

PlaneSweepLib [12] er en effektiv implementering av denne algoritmen bygget for C++ med CUDA, som lar oss kjøre dette på en Nvidia GPU. Siden dybdeestimatene fra Plane Sweep Stereo ikke trenger å bli regnet ut sekvensielt, kan man oppnå hurtig dybdebildeestimering ved å estimere mange dybder samtidig. Algoritmen er derfor svært egnet til å kjøres på en

GPU.

PlaneSweepLib bruker ZNCC i et vindu rundt hver piksel for å sammenligne og gi en korrespondanse-score. I tilfellet der det er flere bilder som skal korresponderes til referansebildet regnes det ut en samlet korrespondansekostnad for hvert plan. De bruker to forskjellige strategier til dette. Den første er å ta et gjennomsnitt av kostnaden til 2k bilder, k før og k etter referansebildet. Den andre er å regne ut gjennomsnittet til de k før, og de k etter hver for seg, og velge den siden av referansebildet med lavest kostnad. Hensikten med denne strategien er å ta hensyn til okklusjoner, med antagelse om at kameraet stort sett flytter seg i samme retning.

Deretter velges det planet som gir lavest samlet kostnad for hver piksel, og dybden kan estimeres. Ved å kun velge dybden fra planet direkte vil man derimot få en ikke-glatt flate. Derfor bruker de subpiksel-interpolasjon for å hente dybder som kan ligge mellom de diskrete planene. Ved å se på korrespondanse-kostnaden til naboplanene, interpolerer de seg til et bedre dybdeestimat ved å finne en parabel som passer med kostnadene, og velger dybden som ligger på ekstremalpunktet til parabelen.

Dette gjøres for hver piksel og gir oss et tett dybdebilde.

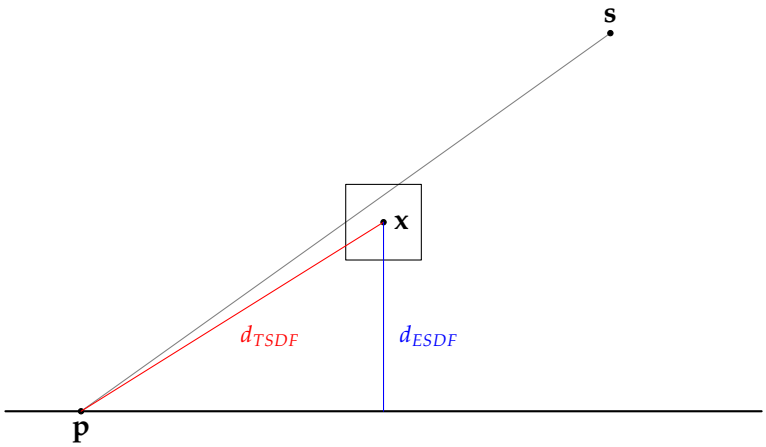
2.6.2 Kartlegging med Voxblox

Med et tett dybdebilde kan vi estimere normalvektoren og krumningen til hvert punkt i bildet, som beskrevet i 2.1. Vi forventer støyete estimater som følge av feil i positur og feilaktige korrespondanser i Plane Sweep, men vi forventer også å observere de samme punktene flere ganger. Dette ønsker vi å utnytte for å forbedre helling- og krumningsestimatene.

Voxblox er et volumetrisk kartleggingsverktøy hovedsaklig basert på trunkerte fortegnede avstandsfelt (Truncated Signed Distance Fields, TSDFs) [21]. Et TSDF er et 3D voksel-kart der hver voksel (volumetrisk piksel) består av den projektive avstanden til overflaten regnet ut innenfor en kort trunkeringsavstand fra overflaten. Projektiv avstand er avstanden fra vokselen til den målte overflaten, langs strålen fra sensoren som målte den:

$$d_{TSDF}(\mathbf{x}, \mathbf{p}, \mathbf{s}) = ||\mathbf{p} - \mathbf{x}||\text{sign}((\mathbf{p} - \mathbf{x}) \cdot (\mathbf{p} - \mathbf{s})) \tag{2.42}$$

Der $\mathbf{x}$ er vokselens sentrum, $\mathbf{p}$ er et punkt på overflaten, og $\mathbf{s}$ er sensorens posisjon. Avstanden regnes da som positiv dersom vokselen er utenfor overflaten, og negativ dersom vokselen er innenfor overflaten. TSDF-er konstrueres ved å integrere opp punktskyer som er målt fra forskjellige positurer. For hver voksel som ligger mellom et punkt i skyen og sensorens posisisjon legges det til en vekt og en vektet projektiv avstand, der vektfunksjonen representerer sikkerheten i målingen og bør avhenge av typen sensor som brukes til dybdemålingen. Fra TSDF-en kan man trekke ut den underliggende overflaten som en mesh med Marching Cubes-algoritmen [18], og et euklidisk fortegnet avstandsfelt (ESDF). I et ESDF inneholder hver voksel i stedet den euklidiske avstanden til nærmeste



Figur 2.12: Avstandene som benyttes i et TSDF og et ESDF. I et TSDF benyttes den projektive avstanden $d_{TSDF}(\mathbf{x}, \mathbf{p}, \mathbf{s})$ (fra 2.42) markert med rødt. I et ESDF er avstanden $d_{ESDF}(\mathbf{x})$ definert som avstanden til nærmeste overflate.

overflate, og er bedre egnet enn et TSDF til baneplanlegging da en bane krever tilstrekkelig euklidisk avstand fra hindringer.

Vekting og fusjonering

Voxblox oppdaterer vokselene etter følgende regler:

$$w_{const}(\mathbf{x}, \mathbf{p}) = 1 \tag{2.43}$$

$$D_{i+1}(\mathbf{x}, \mathbf{p}, \mathbf{s}) = \frac{W_i(\mathbf{x}) D_i(\mathbf{x}) + w(\mathbf{x}, \mathbf{p}) d(\mathbf{x}, \mathbf{p}, \mathbf{s})}{W_i(\mathbf{x}) + w(\mathbf{x}, \mathbf{p})} \tag{2.44}$$

$$W_{i+1}(\mathbf{x}, \mathbf{p}) = \min(W_i(\mathbf{x}) + w(\mathbf{x}, \mathbf{p}), W_{\max}) \tag{2.45}$$

Her er $w_{const}(\mathbf{x}, \mathbf{p})$ en konstant vektfunksjon, men Voxblox foreslår også vektfunksjonen:

$$w_{quad}(\mathbf{x}, \mathbf{p}) = \begin{cases} \frac{1}{z^2} & -\epsilon < d \\ \frac{1}{z^2} \frac{1}{\delta - \epsilon}(d + \delta) & -\delta < d < -\epsilon \\ 0 & d < -\delta \end{cases} \tag{2.46}$$

hvor de bruker trunkeringsavstanden $\delta = 4v$ og $\epsilon = v$, der v er vokselstørrelsen. Motivasjonen bak denne vektfunksjonen er at usikkerheten til en måling langs en stråle varierer tilnærmet proposjonalt med z^2, der z er dybden i kamerareferanserammen.

For å kunne fusjonere punktskyer i sanntid benytter Voxblox en strategi der de kun utfører raycaster en gang per sluttvoksel, og dermed utnytter at vokselstørrelsen er relativt stor i forhold de inkommende punktskyene. Raycasting er å følge en stråle fra kameraets optiske sentrum til sentrum av hvert punkt i skyen, og oppdatere alle vokselene fra kamerasentrumet til en trunkeringsavstand δ bak det observerte punktet. Voxblox foreslår å

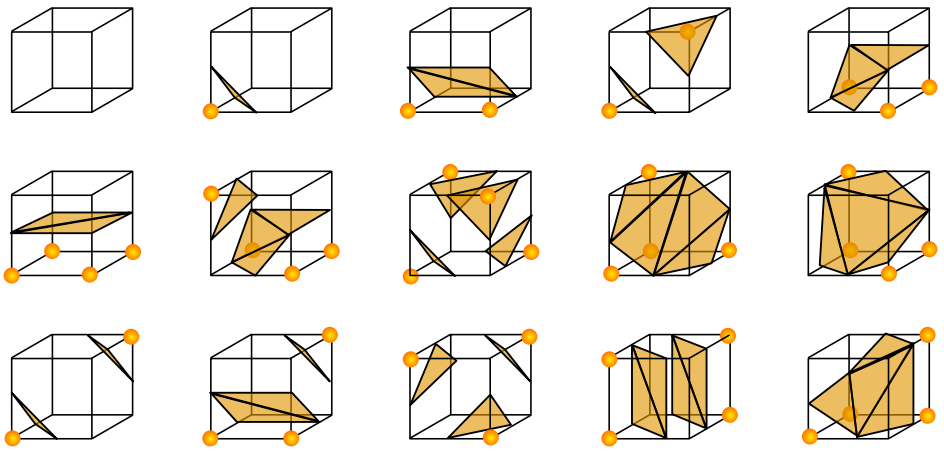
bruke fremgangsmåten de kaller gruppert raycasting for å gjøre dette hurtig uten noe stort tap av nøyaktighet. For hvert punkt i skyen, projiserer de punktets posisjonen til vokselrutenettet, og grupperer det sammen med alle de andre punktene som projekteres til samme voksel. Deretter tar de gjennomsnittet av avstanden og eventuelt fargen i gruppen og utfører raycastingen kun en gang.

Marching Cubes

Marching Cubes er en algoritme for rekonstruksjon av overflater fra diskrete skalarfelt, i vårt tilfelle et vokselkart med avstandsverdier. Den går ut på å sveipe ("marsjere") over vokselkartet, hvor man betrakter 8 nabovoksler samtidig. Vokslene behandles som hjørner i en imaginær kube, og fra vokselverdiene bestemmes det hva slags polygoner som trengs for å representere overflaten som går gjennom kuben.

Dette gjøres ved bruk av en tabell av de $2^8 = 256$ mulige polygonkonfigurasjonene som kan oppstå, avhengig av om hvert hjørne har en positiv eller negativ verdi. Indeksen til tabellen er da et 8-bits heltall, og hver bit til indeksen settes til 1 dersom bitens tilhørende hjørne har en positiv verdi (utenfor overflaten), og 0 dersom det har en negativ verdi (innenfor overflaten). Når vi nå har konfigurasjonen til kuben, finner vi polygonenes hjørner langs kantene ved lineær interpolasjon av avstandsverdiene til kantenes hjørner.

På grunn av rotasjonall og reflektiv symmetri ble det i utgangspunktet foreslått at det bare var 15 unike konfigurasjoner i tabellen. Dette ble senere utvidet til 33, da de opprinnelige 15 ikke håndterte tvetydigheter som kunne oppstå i den trilineære interpolanten langs kubeflatene og på innsiden av kuben [27].



Figur 2.13: De 15 originale unike polygonkonfigurasjonene. Illustrasjon av Jean-Marie «jmtrivial» Favreau [3].

Konstruksjon av et ESDF fra et TSDF

Voxblox har basert sin metode for ESDF-konstruksjon på [16], hvor de konstruerer et ESDF fra et beleggskart, i form av et vokselrutenett med en binær opptatt eller ledig-verdi som algoritmen ikke kan endre på. Ut fra dette regner de ut den euklidiske avstanden til nærmeste opptatte voksel. I Voxblox har de erstattet dette konseptet med et fiksert bånd rundt overflaten, hvor ESDF-voksler tar sin verdi fra sine korresponderende TSDF-voksler, og denne verdien kan ikke endres. Størrelsen på dette båndet er definert av TSDF-voksler der avstandene oppfyller $|d(\mathbf{x})| < \gamma$, hvor γ er båndets radius.

Den generelle algoritmen er basert på ideen om bølgefronter. Man kan tenke seg en bølge som propagerer fra en startvoksel til sine naboer, hvor avstandene blir oppdaterte og sender bølgen videre til sine naboer igjen. De bruker to bølgefronter: en for å øke avstandene og en for å senke dem. En voksel blir sendt til økningskøen dersom den nye avstanden fra TSDFet er høyere enn den tidligere avstanden som er lagret i ESDF-vokselen. Dette vil si at vokselen, og alle vokselene som har blitt oppdatert som følge av den, må ugyldiggjøres. Bølgefronten propagerer frem til det ikke er noen voksler igjen med en overordnet voksel som har blitt ugyldiggjort.

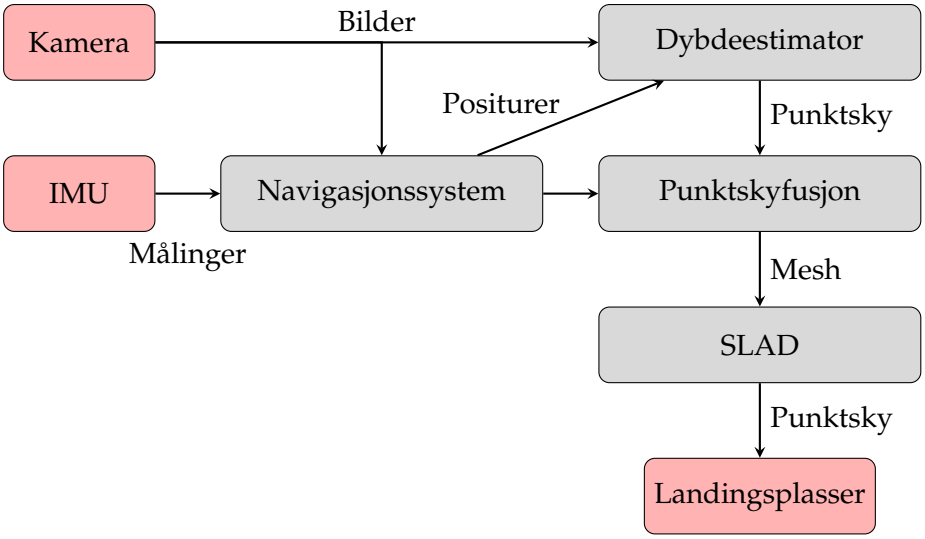
Senkningsbølgefronten begynner ved at en ny fiksert voksel blir lagt til i kartet, eller at en tidligere observert voksel senker verdien sin. Avstandene i nabovokslene blir oppdatert basert på sine nabovoksler og avstandene deres til den nåværende vokselen. Bølgefronten tar slutt når det ikke lengre er noen voksler som kan senkes av naboene sine.

Kapittel 3

Metode

3.1 Oversikt

Systemet består totalt av 4 moduler. En modul for navigasjon, en for dybdeestimering, en for punktskyfusjonering og en for landingsplassdeteksjon (figur 3.1). Vi har ikke bidratt med noe system for baneplanlegging, men systemet er lagt til rette for å kunne utvide med dette ved å konstruere et ESDF.



Figur 3.1: Overblikk av systemet.

Navigasjonsmodulen vi har brukt er SVO, men systemet er ikke avhengig av hva slags navigasjonssystem man bruker. Dybdemodulen tar inn bilder og positurestimater, estimerer normaler, og returnerer punktskyer hvor helling og krumning er egenskaper som tilhører punktene. Punktskyene og deres tilhørende positurer sendes til punktskyfusjonsmodulen, som her er Voxblox, for å bygge et TSDF og eventuelt et ESDF for baneplanlegging. Meshen trekkes ut fra TSDFet, og sendes så videre til landingsplassdetektoren som identifiserer potensielle landingsplasser i meshen, og returnerer dette som en punktsky.

Parameter	Beskrivelse
d_{min}	Minimum avstand mellom bilder.
s	Skaleringsfaktor til nedskalering av bildene.
N_{sweep}	Antall bilder brukt per Plane Sweep.
v_{vindu}	Vindusstørrelse til utregning av ZNCC.
Okklusjonsmodus	Okklusjonshåndteringsstrategi. "Ingen", "Beste k" eller "Referansesplitt".
k_{best}	Antall bilder som tas vare på ved bruk av okklusjonsmodusen "Beste k".
v_{Gauss}	Størrelse på den Gaussiske filterkjernen som brukes på Laplacebildet.
σ	Standardavviket til det Gaussiske filteret.
L_{max}	Maksimal Laplace-respons før vi ekskluderer dybdepiksler.
v_{normal}	Vindusstørrelse til estimering av normalvektorene.
Δz_{max}	Maksimal dybdedifferanse mellom nabopiksler som anses å tilhøre samme overflate.

Tabell 3.1: Parameterne til dybdeestimatoren.

3.2 Dybdeestimering

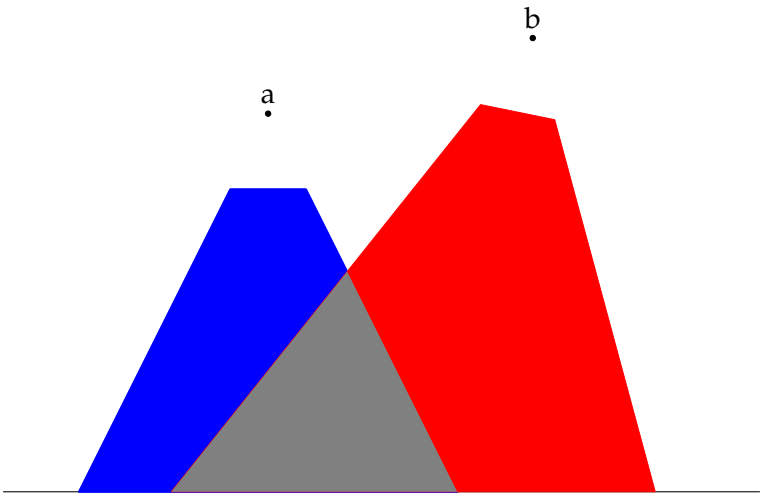
Dybdeestimatoren tar inn bilder fra et kamera, samt bildenes tilhørende positurestimater fra navigasjonssystemet. Den har også flere justerbare parametere, listet i tabell 3.1. Ved behov blir bildene skalert ned og forvrengingen fjernet basert på en radiell-tangensiell forvrengningsmodell. Når det kommer inn et nytt bilde med avstand større enn d_{min} fra alle tidligere bilder legges det i en kø sammen med posituren. Når vi har N_{sweep} bilder kjøres Plane Sweep Stereo-algoritmen med PlaneSweepLib. Referansebildet velges ved at vi regner ut hvilket bilde som ligger lengst frem langs den optiske aksen til det første bildet som kom inn. Dette blir da det største prikkproduktet mellom siste kolonne i $\mathbf{R}_{c_0}^w$ og enhver $\mathbf{t}_{c_i}^w$,

$$i_{ref} = \arg\max_i (\mathbf{R}_{c_0}^w \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \cdot \mathbf{t}_{c_i}^w). \tag{3.1}$$

På denne måten velger vi det bildet som har minst dekning av strukturene, og unngår estimater av områder som ikke er dekket av alle bildene, noe som gir områder i bildet med usikre korrespondanser.

3.2.1 Postprosessering av dybdebildet

Plane Sweep Stereo gir et støyete dybdebilde, så det er nødvendig med et postprosesseringssteg for å fjerne dårlige korrespondanser. I vårt tilfelle er det viktigere at vi får korrekte dybder enn at vi får flere dybdeestimater i bildet, siden feilaktige, frittstående punkter vil tolkes som hindringer i TSDFet og ESDFet. I TSDFet kan dette håndteres ved å sette et minste vektkrav til hver voksel som skal inngå i meshuttrekningen, men i ESDFet vil hvert slikt punkt generere en ikke-traverserbar kule i rommet med



Figur 3.2: Illustrasjon av frustum-avskjæring. Det blå området er $\mathbf{P}_a$, det røde området er $\mathbf{P}_b$, trunkert til det fjerneste dybdeplanet. Vi beholder kun punkter som befinner seg i det grå området $\mathbf{P}_a \cap \mathbf{P}_b$.

radius tilsvarende klaringskravet til fartøyet. Derfor setter vi strenge krav til at en dybdepiksel skal anses som sikker.

De mest sikre dybdene finner vi i områdene som er avbildet i alle bildene. Siden den maksimale og den minimale dybden vi er interessert i må være kjent, kan vi geometrisk utelukke punkter som ikke kan ha blitt observert i alle bildene, og dermed fjerne mange usikre korrespondanser. Dette kalles frustum-avskjæring (frustum culling). Anta to bilder a og b, med felles bildestørrelse WxH, relativ positur $\mathbf{T}_{ab}$ og felles kjent kalibreringsmatrise $\mathbf{K}$. Vi bruker a som referansebilde, og kjenner minimumsdybden z_{min}, som er dybden til ett plan som er fronto-parallell med a. Avbildningsregionen $\mathbf{P}_a$ til a, altså den regionen i 3D-rommet som a kan avbilde, er da avgrenset av et frustum, som er et trapesformet prisme. Frustumet er et konvekst polyeder, som vil si at det avgrenser en undermengde $\mathbf{P} \subseteq \mathbb{R}^3$ hvor

$$t\mathbf{x} + (1-t)\mathbf{y} \in \mathbf{P} \mid \mathbf{x}, \mathbf{y} \in \mathbf{P}, \ \ 0 \leq t \leq 1. \tag{3.2}$$

Altså vil alle linjestykker som blir trukket mellom punkter i mengden også være i mengden. Vi kan representere dette som et sett med halvrom separert av sideflatene. Vi representerer sideflatene som plan på formen

$$\mathbf{n}^T \tilde{\mathbf{p}} = 0 \tag{3.3}$$

hvor

$$\mathbf{n}^T \tilde{\mathbf{p}} = \begin{pmatrix} n_x, & n_y, & n_z, & -d \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = n_x x + n_y y + n_z z - d \tag{3.4}$$

De utvidede normalvektorene til prismet er

$$\begin{aligned}
\mathbf{n}^a_{\text{a nær}} &= \begin{pmatrix} 0 \\ 0 \\ 1 \\ -zmin \end{pmatrix} \\
\mathbf{n}^a_{\text{a venstre}} &= \begin{pmatrix} f_u \\ 0 \\ c_u \\ 0 \end{pmatrix} \\
\mathbf{n}^a_{\text{a høyre}} &= \begin{pmatrix} -f_u \\ 0 \\ c_u \\ 0 \end{pmatrix} \\
\mathbf{n}^a_{\text{a opp}} &= \begin{pmatrix} 0 \\ f_v \\ c_v \\ 0 \end{pmatrix} \\
\mathbf{n}^a_{\text{a ned}} &= \begin{pmatrix} 0 \\ -f_v \\ c_v \\ 0 \end{pmatrix}
\end{aligned} \tag{3.5}$$

Settet med punkter som er inneholdt i prismet er da

$$\begin{aligned}
\mathbf{P}^a_a = \{\mathbf{p} \mid & \mathbf{p} \cdot \bar{\mathbf{n}}^a_{\text{a nær}} \geq 0, \\
& \mathbf{p}^a \cdot \bar{\mathbf{n}}^a_{\text{a venstre}} \geq 0, \\
& \mathbf{p}^a \cdot \bar{\mathbf{n}}^a_{\text{a høyre}} \geq 0, \\
& \mathbf{p}^a \cdot \bar{\mathbf{n}}^a_{\text{a opp}} \geq 0, \\
& \mathbf{p}^a \cdot \bar{\mathbf{n}}^a_{\text{a ned}} \geq 0\}
\end{aligned} \tag{3.6}$$

hvor $\bar{\mathbf{n}} = \frac{\mathbf{n}}{|\mathbf{n}|}$ er enhetsnormalvektorene.

Avbildningsregionen til b er også en mengde $\mathbf{P}^b_b$ som er begrenset av et tilsvarende prisme. Siden vi har antatt at $\mathbf{K}$ er felles, har vi at

$$\mathbf{n}^b_b = \mathbf{n}^a_a \tag{3.7}$$

Så må vi finne planene utspent av b i a. Vi har

$$\mathbf{n}^{bT}_b \tilde{\mathbf{p}}^b = 0 \tag{3.8}$$

Planet som inneholder de samme punktene transformert til a må oppfylle

$$\mathbf{n}^{aT}_b \mathbf{T}_{ab} \tilde{\mathbf{p}}^b = 0 \tag{3.9}$$

Vi ser at vi får

$$\mathbf{n}^{aT}_b \mathbf{T}_{ab} = \mathbf{n}^{bT}_b \tag{3.10}$$

$$\mathbf{n}^{aT}_b = \mathbf{n}^{bT}_b \mathbf{T}^{-1}_{ab} \tag{3.11}$$

$$\mathbf{n}_b^a = \mathbf{T}_{ab}^{-T}\mathbf{n}_b^b \tag{3.12}$$

Dette lar oss definere

$$\begin{aligned}\mathbf{P}_b^a = \{\mathbf{p} \mid & \mathbf{p}^a \cdot \bar{\mathbf{n}}_{\text{b nær}}^a \geq 0, \\ & \mathbf{p}^a \cdot \bar{\mathbf{n}}_{\text{b venstre}}^a \geq 0, \\ & \mathbf{p}^a \cdot \bar{\mathbf{n}}_{\text{b høyre}}^a \geq 0, \\ & \mathbf{p}^a \cdot \bar{\mathbf{n}}_{\text{b opp}}^a \geq 0, \\ & \mathbf{p}^a \cdot \bar{\mathbf{n}}_{\text{b ned}}^a \geq 0\}\end{aligned} \tag{3.13}$$

Den felles avbildningsregionen, altså regionen som kan avbildes av både a og b, er da omsluttet av

$$\mathbf{P}_{ab}^a = \mathbf{P}_a^a \cap \mathbf{P}_b^a = \{\mathbf{p} \mid \mathbf{p} \in \mathbf{P}_a^a, \mathbf{p} \in \mathbf{P}_b^a\} \tag{3.14}$$

Vi kan da bestemme om et punkt $\mathbf{p}$ er inneholdt i $\mathbf{P}_{ab}^a$ ved å sjekke at det er i $\mathbf{P}_a^a$ og $\mathbf{P}_b^a$. Dette lar seg naturlig utvide til n forskjellige bilder for å kun beholde punktene som har blitt observert i alle bildene.

$$\mathbf{P}_{ab\ldots n}^a = \mathbf{P}_a^a \cap \mathbf{P}_b^a \cap \ldots \cap \mathbf{P}_n^a \tag{3.15}$$

Dette fjerner en del områder uten dybdeinformasjon, men det vil fortsatt være en del støy i resten av bildet. Denne støyen har noen særtrekk. Siden vi avbilder forholdsvis glatte overflater, vil vi svært ofte få høy lokal frekvens i piksler som har fått feil dybde. Vi bruker derfor et Laplace-filter til å identifisere usikre dybder i bildet. Laplace-operatoren er i det kontinuerlige 2D-tilfellet definert som

$$\nabla^2 f(x,y) = \frac{d^2 f}{dx^2} + \frac{d^2 f}{dy^2}. \tag{3.16}$$

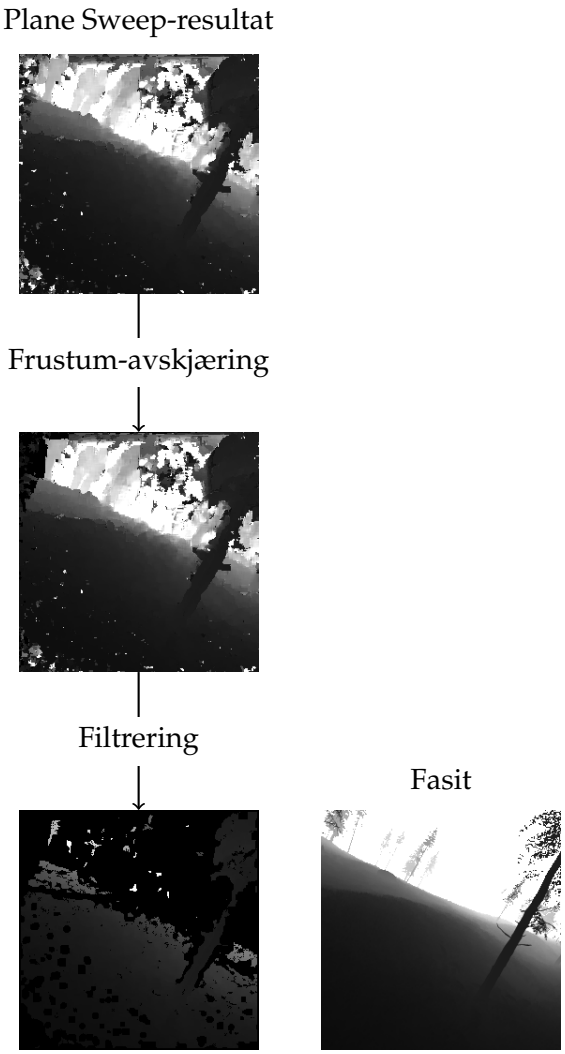
Dette kan tilnærmes i det diskrete tilfellet som konvolusjonen

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * I, \tag{3.17}$$

hvor I er et intensitetsbilde. Deretter bruker vi et Gaussisk filter med filterkjerne G på magnitudene i Laplace-bildet for å spre høye magnituder til nabopikslene, og da ekskludere mindre områder rundt usikre piksler. For å sikre at den Gaussiske glattingen ikke demper høye magnituder tillater vi kun glattingen å øke magnitudene:

$$L^+ = \max(G * L, L). \tag{3.18}$$

Piksler hvor $L^+ > L_{max}$ blir da ekskludert. En konsekvens av dette er at vi mister noen reelle kanter, men de fleste kan bevares da kantene typisk har lavere respons på Laplace-filteret enn støyen.



Figur 3.3: Dybdeestimeringsprosessen.

Vi regner så ut normalene i dybdebildet ved bruk av integralbilder, med en adaptiv, dybdeavhengig kvadratstørrelse.

Målet med å variere kvadratstørrelsen er å minimere antall normalestimater vi må gjøre, da Voxblox uansett vil bruke gjennomsnittet av punktegenskapene innenfor endevokslene til raycastingen. Ved å gjøre kvadratstørrelsen avhengig av dybden kan vi få kvadratstørrelsen til å tilsvare en voksel på en bestemt avstand. Vi gjør et grovt estimat av den lokale maksimaldybden ved å dele dybdebildet i et 5x5 rutenett. For hvert slikt underbilde finner vi den største dybden, og kvadratet som brukes i underbildet settes slik at det er på størrelse med en voksel på denne dybden. Dersom dybden er for stor vil kvadratstørrelsen gå mot 0, så den minste kvadratstørrelsen som tillates er 3x3. Hellingen $h = \mathbf{n}_p \cdot \mathbf{z}_w$ og krumningen k settes som egenskaper tilhørende punktet, og tyngdepunktet innenfor kvadratet settes som posisjonen. Den resulterende punktskyen sendes så videre til Voxblox for å bygge kartet inkrementelt.

3.3 Fusjonering

Som nevnt tidligere oppdaterer Voxblox avstanden og vekten i vokslene etter reglene gitt i ligningene 2.44 og 2.45. Fargene til vokslene blir også oppdatert etter regelen:

$$\mathbf{f}_{i+1}(\mathbf{x}, \mathbf{p}) = \frac{W_i(\mathbf{x})\mathbf{f}_i(\mathbf{x}) + w(\mathbf{x}, \mathbf{p})\mathbf{f}_{\text{ny}}(\mathbf{x}, \mathbf{p})}{W_i(\mathbf{x}) + w(\mathbf{x}, \mathbf{p})}. \tag{3.19}$$

Ved å putte hellingen og krumningen til hvert punkt i hver sin fargekanal, så fusjoneres også disse verdiene på samme måte som avstandene. Dette lar oss bruke flere observasjoner av de samme områdene for å forbedre helling- og krumningsestimatet per voksel.

Voxblox har kun støtte for 8-bits RGBA-farger. For å sikre god oppløsning gjør vi derfor en avbildning av hellingen og krumningen slik at de mest signifikante verdiintervallene havner i intervallet $[0, 255]$, etterfulgt av en trunkering til nærmeste heltall:

$$h_{farge} = \lfloor 255 \cdot \text{lerp}(h, 0.85, 1) \rfloor, \tag{3.20}$$

$$k_{farge} = \lfloor 255 \cdot \text{lerp}(k, 0, 0.05) \rfloor \tag{3.21}$$

hvor

$$\text{lerp}(x, x_{min}, x_{max}) = \begin{cases} 0 \text{ hvis } x \leq x_{min} \\ (x - x_{min})/(x_{max} - x_{min}) \text{ hvis } x_{min} < x < x_{max} \\ 1 \text{ hvis } x \geq x_{max} \end{cases} \tag{3.22}$$

er lineær interpolasjon. Vi setter k_{farge} på rød-kanalen siden lavere k er bedre, og h_{farge} på grønn-kanalen da høyere h er bedre.

Ved å velge det signifikante hellingsintervallet som $[0.85, 1]$ dekker vi

hellingsverdier mellom $\arccos(1) = 0$ og $\arccos(0.85) \approx 32$ grader, som er forholdsvis bratt, men kan være av interesse. Krumningsintervallet er vanskeligere å tolke direkte, men visuelt ser det ut til å gi gode overganger mellom flatt og ujevnt terreng.

3.4 Landingsplassdeteksjon

3.4.1 Landbarhet

Vi har tidligere definert et område som landbart dersom det har følgende egenskaper:

- Stort
- Flatt
- Lav helling

Hellingen og flatheten ligger allerede i TSDFet i Voxblox. Når vi ønsker å initiere en landing trekker vi ut meshen fra Voxblox for å plassere den estimerte hellingen og flatheten på en overflate (figur 3.4). Vi putter et hjørne fra hver trekant i meshen inn i en punktsky $\mathbf{C}_L$, med tilhørende helling h og krumning k som egenskaper.

Kravet til størrelsen på landingsområdet er avhengig av størrelsen på fartøyet, så størrelseskravet er derfor en parameter S som settes når man starter systemet. Vi tar hensyn til størrelsen ved å dele punktskyen opp i et kvadratisk 2D rutenett med akser som ligger normalt på gravitasjonsretningen, der hver rute har sidelengde S. Til hver rute tillegger vi et punkt, med rutens tyngdepunkt som posisjon $\mathbf{x}_L$. Egenskapene til punktet regnes ut ved en å finne en avstandsvektet maksimal av hellingen og krumningen i punktskyen, slik at verdier langt fra tyngdepunktet får mindre innflytelse. Vekten regnes ut ved å evaluere en Gaussisk funksjon av den normaliserte avstanden i xy-planet.

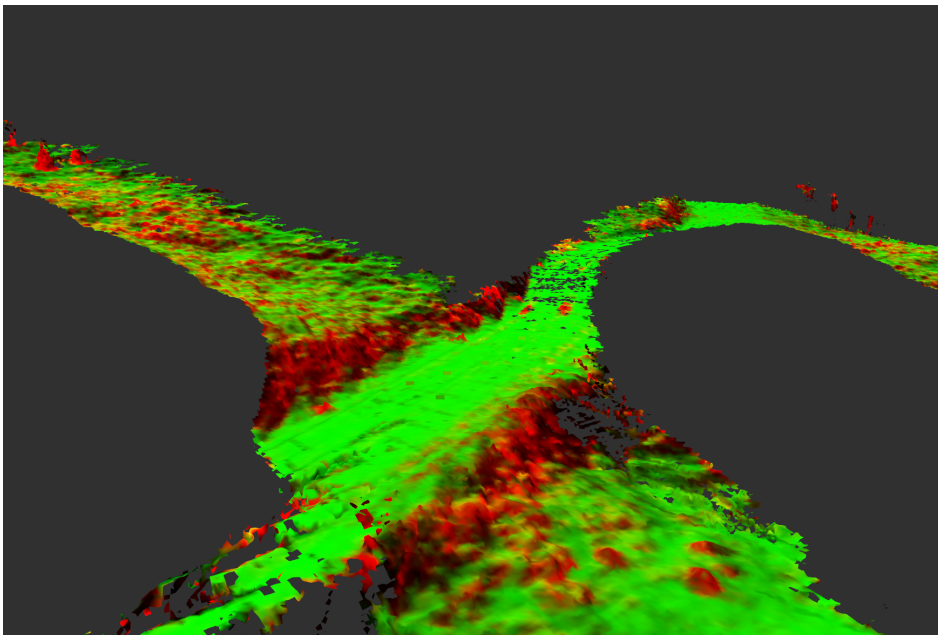
$$v(\mathbf{x}) = \exp(-2\frac{|\mathbf{x} - \mathbf{x}_L|^2}{S^2}) \tag{3.23}$$

$$h_L = \max_{\mathbf{x} \in \mathbf{C}_L} v(\mathbf{x})h(\mathbf{x}) \tag{3.24}$$

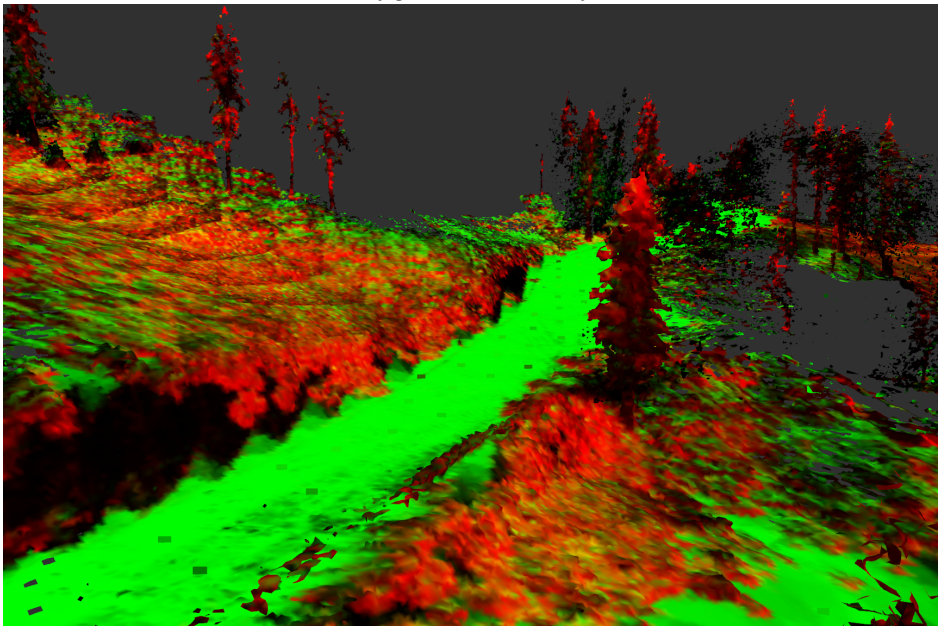
$$k_L = \max_{\mathbf{x} \in \mathbf{C}_L} v(\mathbf{x})k(\mathbf{x}) \tag{3.25}$$

Vektfunksjonen er konstruert fra en normalfordeling med standardavvik $\frac{1}{2}$ uten normaliseringsfaktoren $\frac{1}{\sigma\sqrt{2\pi}}$, slik at $v(\mathbf{x}_L) = 1$, og avstander på en halv sidelengde gir $v = \exp(-0.5) \approx 0.6$ da tyngdepunktet vil ligge nært midten av arealet. Slik sikrer vi at det potensielle landingsområdet er tilstrekkelig stort, og at det er robust mot støy, særlig mot kantane av S hvor egenskapene har mindre å si. Dette sikrer oss også at området har tilstrekkelig avstand fra hindringer, da hindringer gir stort utslag på krumning eller helling, avhengig av størrelsen på hinderet.

Mesh bygd med dybdeestimater



Mesh bygd med sanne dybder



Figur 3.4: Eksempel på uttrekt mesh. Farget med helling i grønt og krumning i rødt.

Parameter	Beskrivelse
S	Sidelengde på det klarerte kvadratet.
N_{min}	Minstekrav til antall punkter i kvadratet.
θ_{max}	Maksimal godtatt helling i kvadratet, i grader, $h_{min} = \cos\theta_{max}$.
k_{max}	Maksimal godtatt krumning i kvadratet.

Tabell 3.2: Parametere til landingsplassdetektoren.

Når vi nå har funnet de relevante målene må vi bestemme den beste landingsplassen å bruke. Siden målingene på helling og flathet er noe usikre, vil ikke en sammenligning av to områder med lav helling og lite krumning nødvendigvis gi oss den optimale landingsplassen. Derfor har vi valgt å innføre en brukerstyrt maksgrense på helling og krumning, da forskjellige fartøy vil ha forskjellige krav. Eksperimentelt har vi fått gode resultater med maksimal helling på mellom 20 og 30 grader, og maksimal krumning mellom 0.01 og 0.05. Systemet velger deretter det nærmeste området som er innenfor terskelverdiene. En oppsummering av parameterene til detektoren er gitt i 3.2.

Kapittel 4

Resultater og diskusjon

I dette kapittelet vil vi gå gjennom eksperimentene vi har gjennomført og resultatene vi har fått. Alle eksperimentene er kjørt i sanntid på en Nvidia Jetson TX2.

4.1 Datasett

Til å teste systemet har vi brukt datasett fra Mid-Air [5], og Euroc [1].

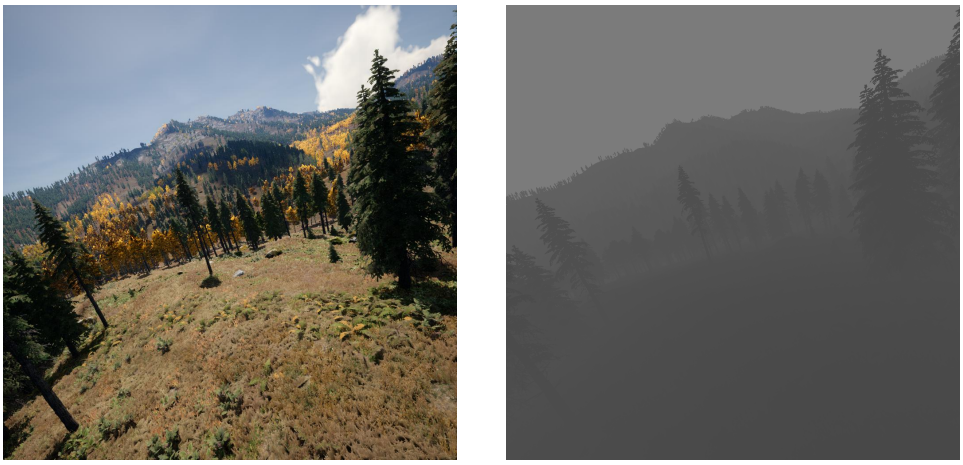
Mid-Air er et syntetisk datasett laget med en flysimulator, bestående av utendørs droneflyvninger i lav høyde. Dette har gitt dem mulighet til å lage et datasett med nøyaktige fasiter under forhold hvor det ellers ville vært svært utfordrende. Datasettet inkluderer blant annet:

- RGB-bilder fra et binokulært (stereo) fremovervendt kameraoppsett
- RGB-bilder fra et monokulært nedovervendt kamera
- Målinger fra et simulert akselerometer, gyroskop og GPS
- Sanne avstandsbilder fra det venstre fremovervendte kameraet
- Sanne normalbilder fra det venstre fremovervendte kameraet
- Sanne semantisk segmenterte bilder fra det venstre fremovervendte kameraet
- Sann bevegelsesdata, inkludert positur, hastighet, akselerasjon og vinkelhastighet

Siden dybdene er gitt som Euklidske avstander fra kameraets fokalpunkt per piksel (x_i, y_i, z_r), og vi estimerer dybdene som avstand langs den optiske aksen (x_i, y_i, z_c), har vi konvertert avstandsbildene til dybdebilder ved

$$r_{xy} = \frac{z_r}{\sqrt{(x_i - w/2)^2 + (y_i - h/2)^2 + f^2}}, \tag{4.1}$$

$$z_c = r_{xy} f \tag{4.2}$$



Figur 4.1: Eksempel på RGB-bilde og avstandsbilde fra Mid-Air.

hvor f er kameraets fokallengde, h er høyden til bildet og w er bredden til bildet. For Mid-Air er dette

$$f = f_u = f_v = h/2 = w/2 = 512. \tag{4.3}$$

På grunn av problemer med å få initialisert SVO på dette datasettet har vi benyttet de sanne positurene i estimeringen, og testet dybdeestimatoren og landingsplassdeteksjonen isolert fra navigasjonen.

Euroc er et datasett bestående av innendørs droneflyvninger og inkluderer bilder fra et stereokamera, og målinger fra et akselerometer og et gyroskop. Euroc har opptak fra to miljøer, Machine Hall og Vicon Room, med litt forskjellige fasiter. Til Machine Hall (figur 4.2) har de tatt opp dronens nøyaktige posisjon med en lasersporer, og til Vicon Room har de tatt opp nøyaktig positur med et Vicon bevegelsessporingssystem. Til Vicon Room har de også en nøyaktig 3D-scan av miljøet (figur 4.3).

4.2 Dybdeestimering

Som mål på dybdeestimering har vi brukt gjennomsnittet av absolutt feil, gjennomsnittet av relativ absolutt feil, og dekningsgrad. Gjennomsnittlig absolutt feil er

$$\epsilon_{MAE} = \frac{1}{N}\sum_{i=0}^{N} |\hat{z}_i - z_i|, \tag{4.4}$$

hvor $\hat{z}$ er estimert dybde, z er den sanne dybden og N er antall estimater. Gjennomsnittet av relativ absolutt feil er

$$\epsilon_{MRAE} = \frac{1}{N}\sum_{i=0}^{N} \frac{|\hat{z}_i - z_i|}{z_i}. \tag{4.5}$$

Grunnen til at vi også bruker gjennomsnittet av relativ absolutt feil er at det gir et tydeligere skille mellom store og mindre feil (figur 4.8). Dekningsgrad

Figur 4.2: Bilde fra Machine Hall

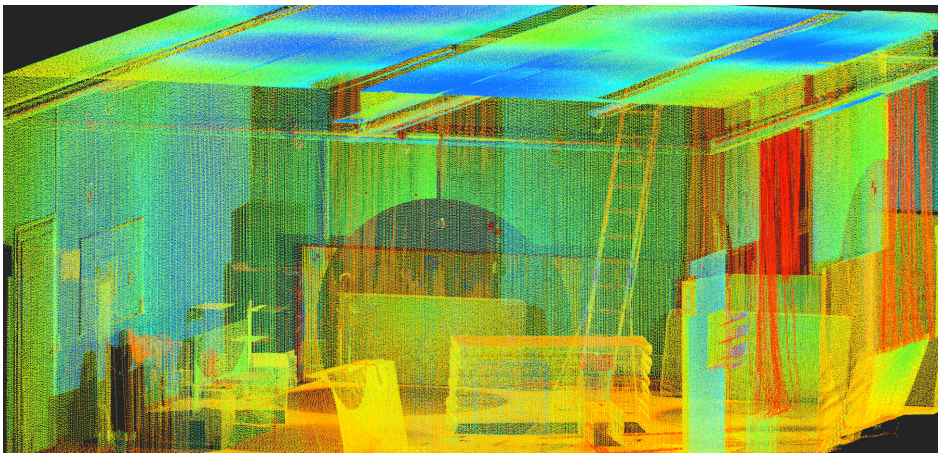
har vi definert som andelen estimater der hvor den sanne dybden var innenfor det relevante dybdevinduet, i forhold til antall sanne dybder innenfor dybdevinduet.

$$D = \frac{\text{Antall } \hat{z}_i \mid z_{min} < z_i < z_{max}}{\text{Antall } z_i \mid z_{min} < z_i < z_{max}} \tag{4.6}$$

Dette gir et mål på hvor stor andel av de mulige estimatene vi har inkludert, men sier ingenting om hvor gode estimatene er. Derfor har vi også regnet ut feilene av de estimerte pikslene som ble ekskludert av frustum-avskjæringen og filtreringen, for å se på effekten av estimatekskluderingen.

Her så vi kun på dybdene mellom 5m og 30m. Kjøreparameterene som ble brukt er gitt i tabell 4.1. Det ble estimert totalt 246 dybdebilder med størrelse 256x256 i løpet av 88 sekunder, som gir en dybdebildefrekvens på ca 2.8 bilder per sekund. Resultatene er oppsummert i tabell 4.2.

For å se hvordan feilen blir påvirket av avstand regnet vi også ut feilen som funksjon av sann dybde (figur 4.4 og 4.5). Det vi ser er at feilen øker en del med dybden, og at vi får et sterk bias mot underestimering. For å finne en mulig forklaring på dette fenomenet så vi på et eksempel fra bildeserien (figur 4.6). I dette tilfellet vil den fine strukturen til trærne bli smurt utover av Plane Sweepen, noe som gir store feil i dybdeestimeringen. Dette påvirker ikke landingsplassdeteksjonen i særlig grad, da dette er punkter i rommet som uansett ligger nært hindringer. I eksempelbildet har vi også fått med noen feilestimerte piksler fra himmelen, noe gir store feil da den sanne dybden til himmelen er satt til $2^{16} - 1 = 65535$. Hvis vi ser på sannsynlighetsfordelingen av feilen (figur 4.7 og 4.8) ser vi at det er et mindretall punkter med stor feil.



Figur 4.3: Punktskyen av Vicon Room

Parameter	Verdi
d_{min}	0.1
s	0.25
N_{sweep}	6
v_{vindu}	7
Okklusjonsmodus	Ingen
k_{best}	Ikke brukt
v_{Gauss}	5
σ	5
L_{max}	3
v_{normal}	3
Δz_{max}	200

Tabell 4.1: Parameterne brukt til test av dybdeestimatoren.

4.3 SLAD

For å teste landingsplassdeteksjonen har vi kjørt det på Mid-Air-datasettet og Euroc-datasettet. Machine Hall har ingen fasit på strukturen, så her har vi bare inspisert resultatet og gjort en kvalitativ vurdering.

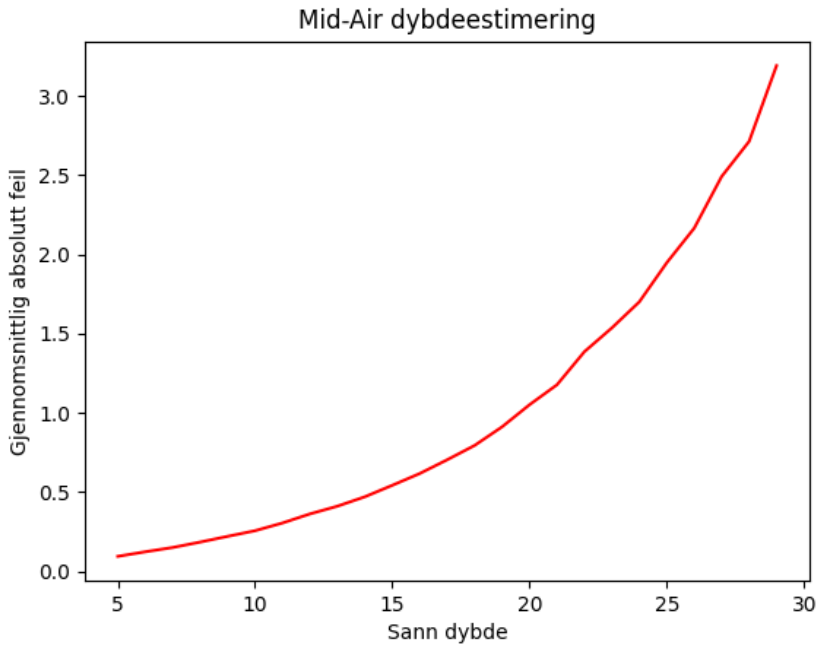
4.3.1 Mid-Air

Som nevnt tidligere testet vi på Mid-Air uten navigasjon, og brukte sanne positurer. Som sammenligningsgrunnlag har vi generert tilnærmet sanne trygge landingsplasser med å kjøre systemet på de sanne dybdene. Deretter behandler vi det som et binært klassifiseringsproblem, hvor klassene er ”landbart” og ”ikke landbart”. Målene vi bruker er presisjon og sensitivitet, definert som

$$\text{presisjon} = \frac{\text{antall sanne positive}}{\text{totalt antall estimater}} \tag{4.7}$$

Datasett	Trajectory 4000 (fall)
ϵ_{MAE} inkluderte	814.52
ϵ_{MAE} ekskluderte	14362.9
ϵ_{MRAE} inkluderte	0.068
ϵ_{MRAE} ekskluderte	0.542
D	0.7215

Tabell 4.2: Resultater fra dybdeestimeringen. Vi kan se at estimatekskluderingen er effektiv.



Figur 4.4: Gjennomsnittlig absolutt feil som funksjon av sann dybde. Gjennomsnittet er regnet ut innenfor intervaller på 1 meter.

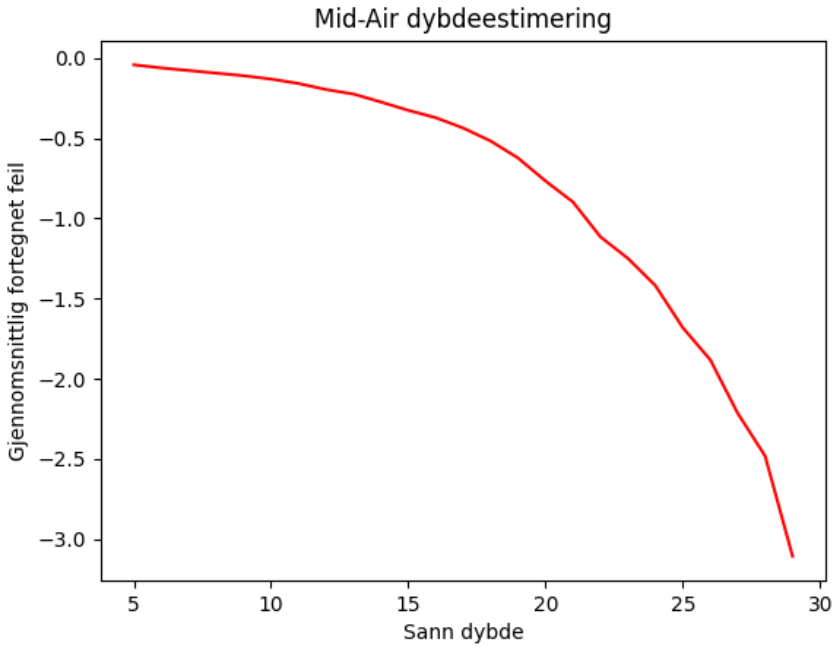
$$\text{sensitivitet} = \frac{\text{antall sanne positive}}{\text{antall mulige positive}} \tag{4.8}$$

Når vi klassifiserer landingsplasser kan vi tolerere at vi ikke korrekt klassifiser alle de mulige landingsplassene, men falske positiver er å betrakte som kritiske feil. Presisjon er derfor det kritiske målet, og sensitivitet er sekundært.

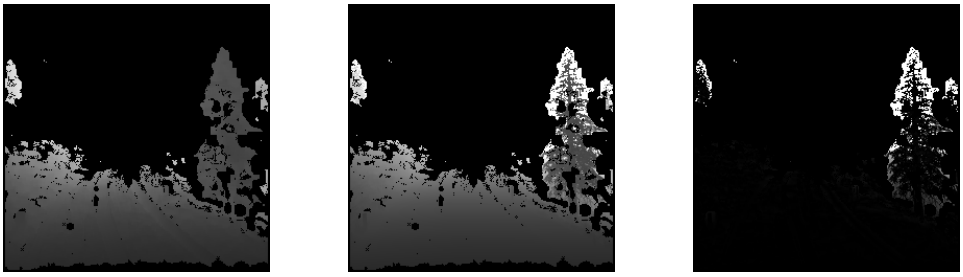
Kravet vi har satt til landingsplassene her er maksimal helling på 20 grader, maksimal krumning på 0.02, og klarert areal på 3x3m.

Alle estimatene er korrekt klassifisert som trygge landingsplasser med unntak av en fra den nedovervendte kjøringen. Testen med nedovervendt kamera klarer likevel å identifisere flere, og har høyere sensitivitet.

Den økte sensitiviteten ved nedovervendt kamera skyldes trolig at det gir mindre støyete dybdebilder, da man ikke ser trær eller himmel.



Figur 4.5: Gjennomsnittlig fortegnet feil $\frac{1}{N}\sum_{i=0}^{N}\hat{z}_i - z_i$ som funksjon av sann dybde. Gjennomsnittet er regnet ut innenfor intervaller på 1 meter. Det er et klart bias mot underestimering.



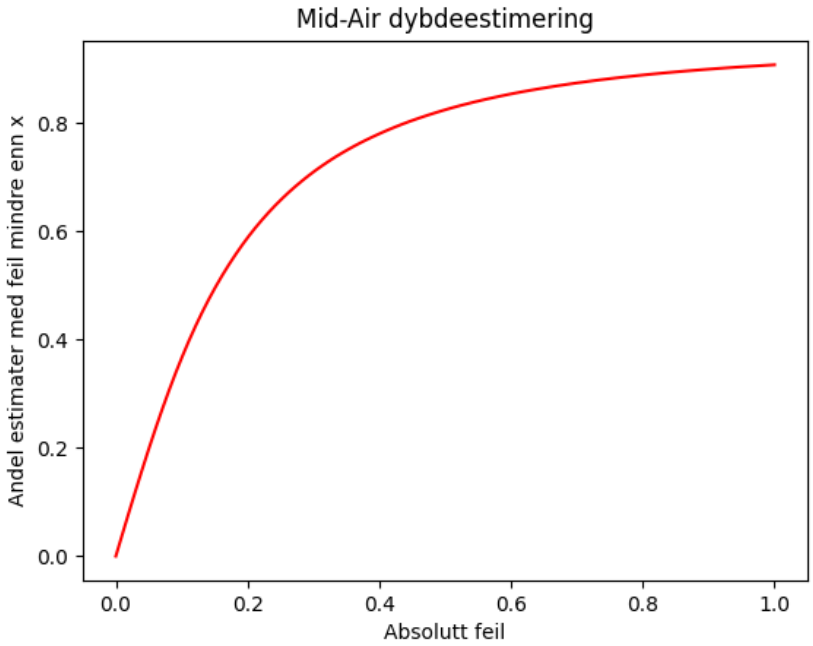
Figur 4.6: Fra venstre: estimert dybde $\hat{z}$, sann dybde z, absolutt feil $|\hat{z} - z|$.

Antall mulige: 273	Fremovervendt	Nedovervendt
Antall estimater	16	41
Sanne positiver	16	40
Presisjon	1	0.98
Sensitivitet	0.06	0.15

Tabell 4.3: Resultater av landingsplassdeteksjon. Vi oppnår høy presisjon, som ønsket. Sensitiviteten er mer variabel, men også brukbar.

4.3.2 Euroc

Her har vi testet systemet med SVO. I både Machine Hall og Vicon Room har vi satt krav om klaring på 0.5x0.5m, helling på mindre enn 30 grader og krumning på mindre enn 0.05.



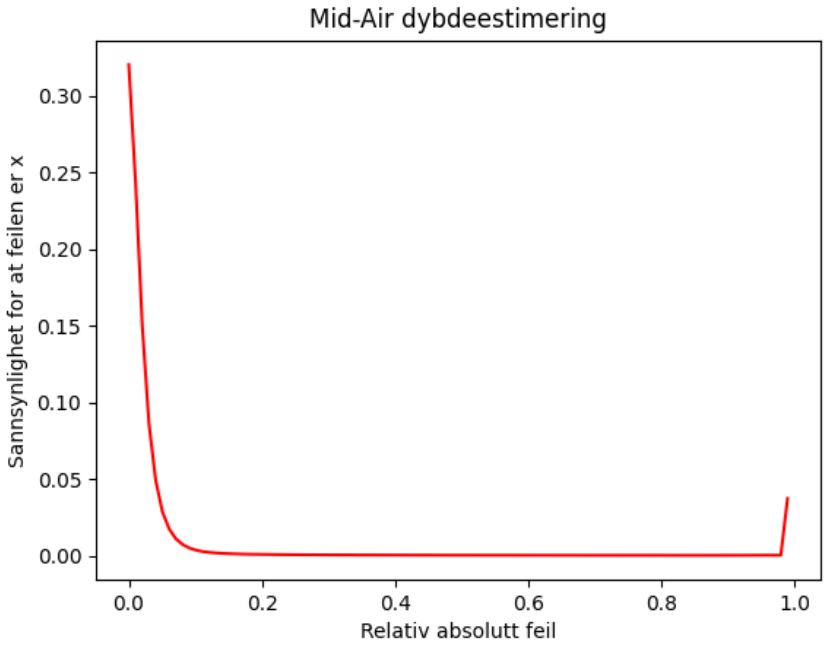
Figur 4.7: Sannsynlighet for at feilen er mindre enn x. Oppløsning på $\Delta x = 0.01$. Fra figuren kan man se at 90% av estimatene har feil på mindre enn 1m.

I Machine Hall-tilfellet har vi ingen fasit på strukturen, så vi kan ikke med sikkerhet si at resultatene er gode, men de ser stort sett fornuftige ut. Det er dog en landingsplass som har blitt detektert i løse luften, noe som skyldes at det er detektert overflater med lav helling og krumning både over og under punktet. Vi kunne tatt hensyn til slike tilfeller ved å regne ut en varians i gravitasjonsretningen innenfor kvadratet, og på den måten diskvalifisere landingsplassen.

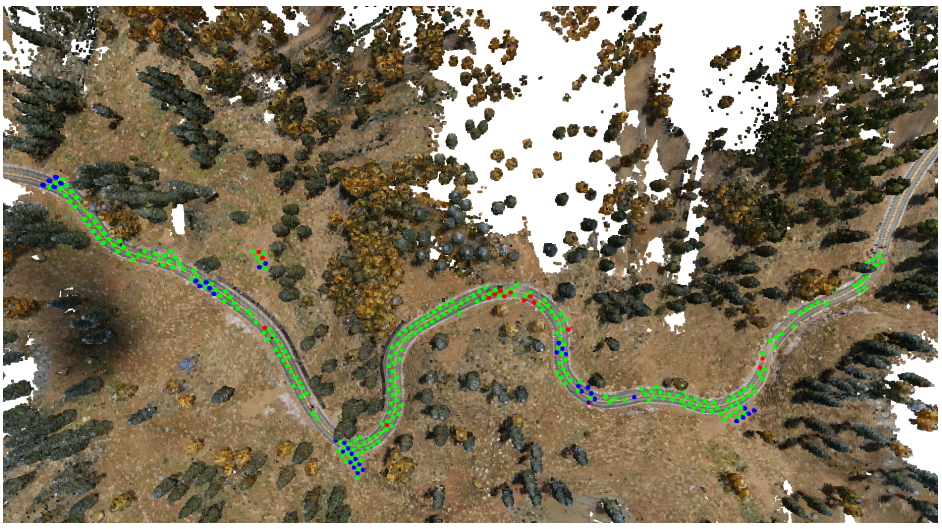
I Vicon-tilfellet kjenner vi ikke transformasjonen mellom koordinatsystemet brukt av SVO og koordinatsystemet brukt i den sanne punktskyen, så vi har forsøkt å manuelt finne frem til transformen. Dette kan ha medført mindre feil i plasseringen av landingsplassene. Utover det ser resultatene fornuftige ut også her. Her har vi kun funnet frem til en presisjonsverdi ved inspeksjon, oppsummert i tabell 4.4.

	Machine Hall	Vicon Room
Antall estimater	12	5
Sanne positiver	11	5
Presisjon	0.917	1

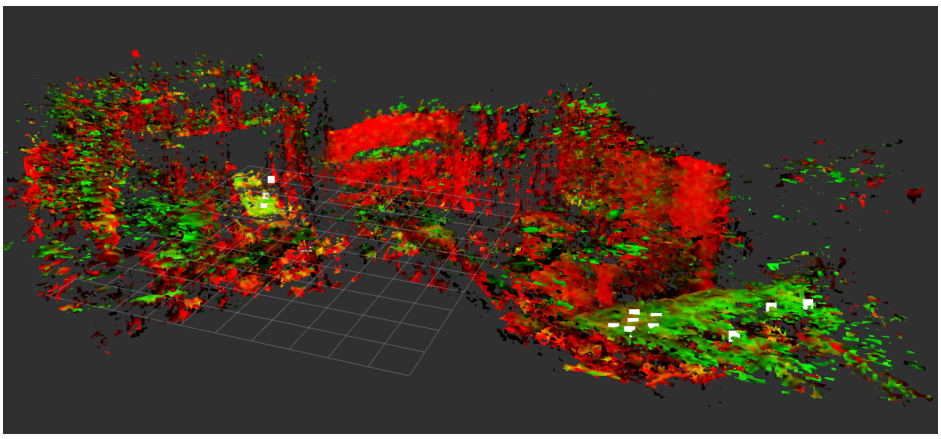
Tabell 4.4: Resultater fra kjøring på Euroc-datasettene.



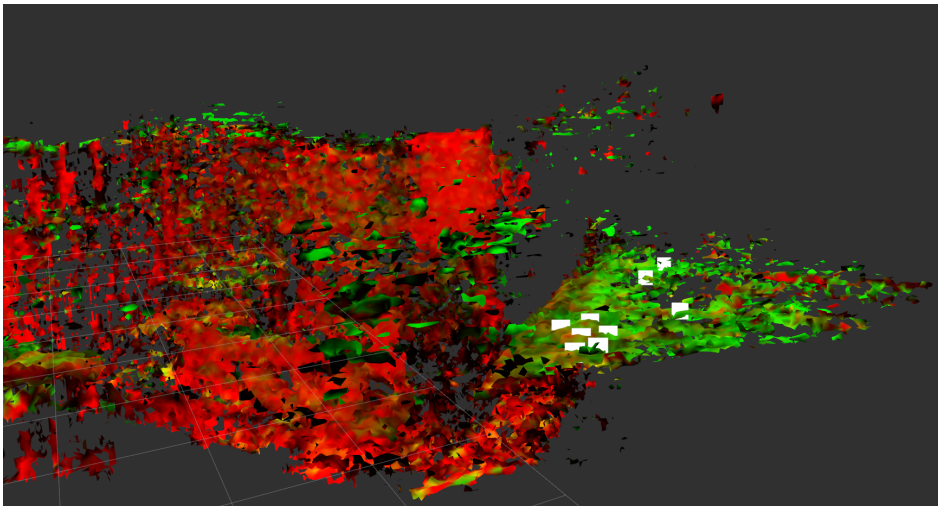
Figur 4.8: Sannsynlighet for at feilen er x. Oppløsning på $\Delta x = 0.01$.



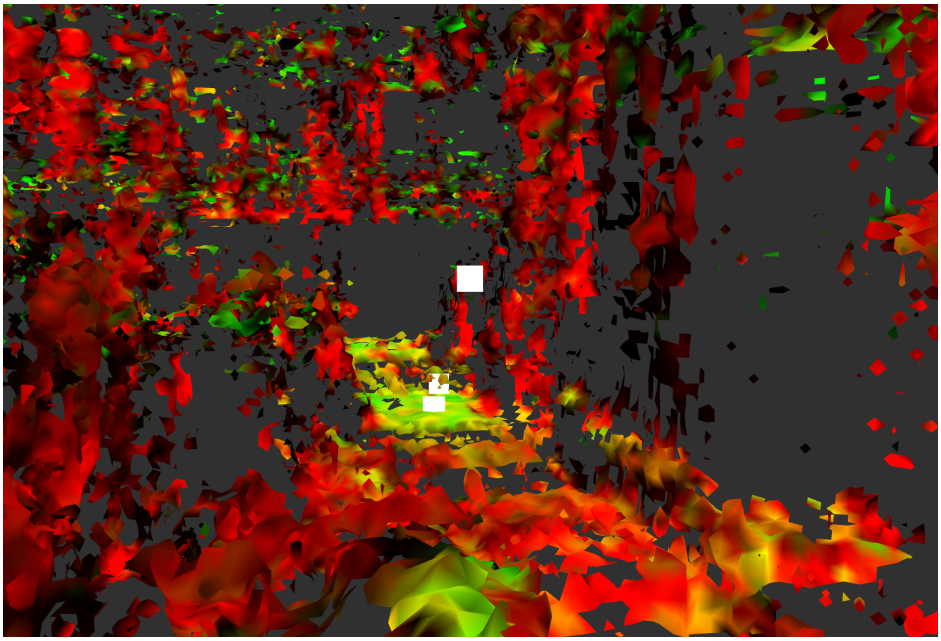
Figur 4.9: To kjøringer av landingsplassdeteksjon på Mid-Air, prosjektert inn i en sann punktsky. Grønne punkter er sanne landingsplasser, røde punkter er fra en kjøring med fremovervendt kamera, blå punkter er fra en kjøring med nedovervendt kamera.



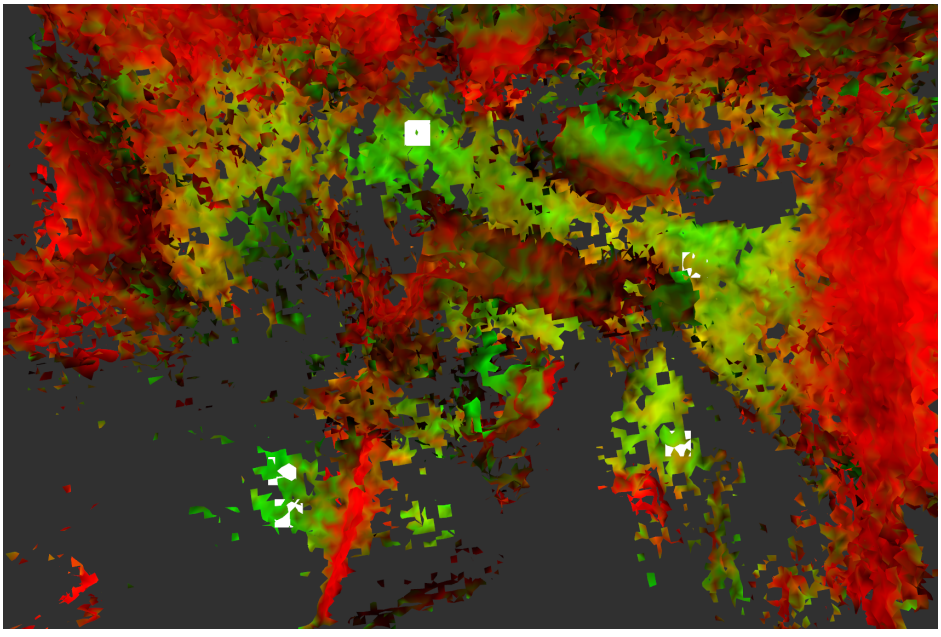
Figur 4.10: Resultat fra Machine Hall. Overblikk av den resulterende meshen, farget med helling og krumning. De hvite firkantene er detekterte landingsplasser.



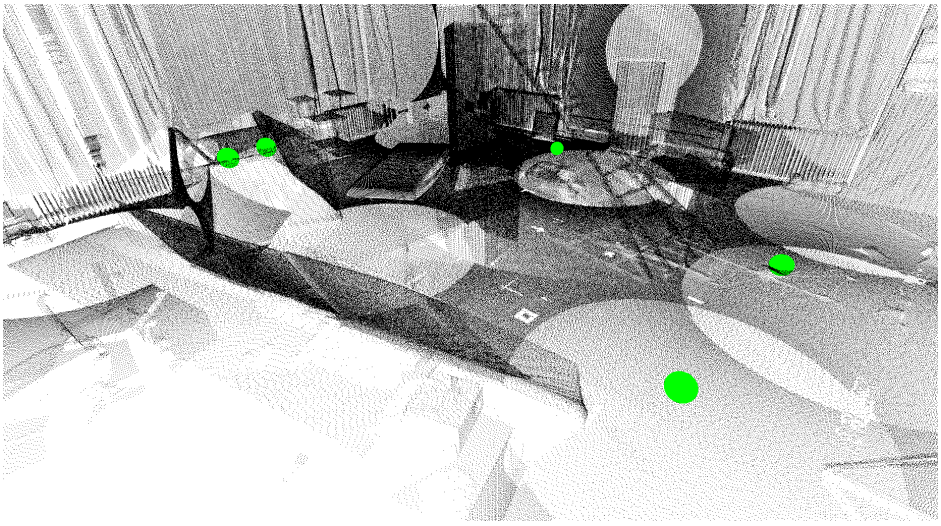
Figur 4.11: Machine Hall. En flate med mange fornuftige deteksjoner.



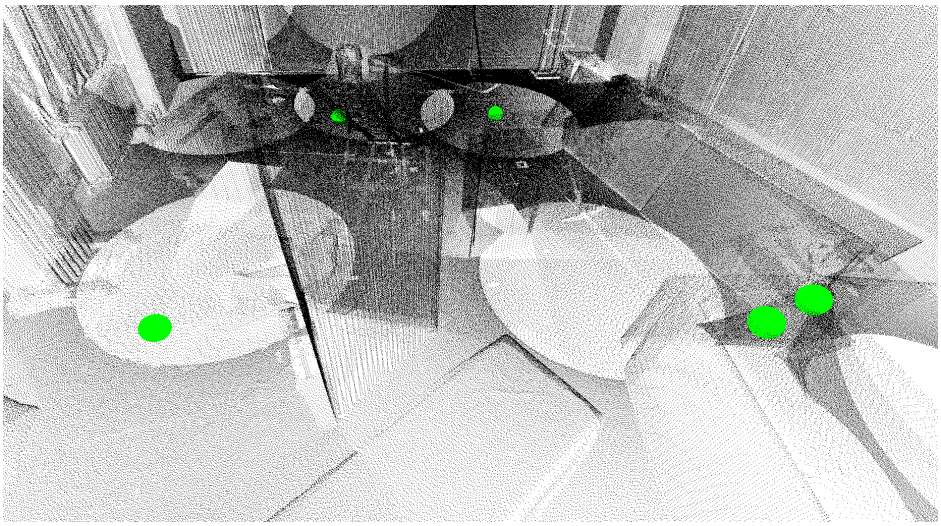
Figur 4.12: Machine Hall. Fra gangen til venstre i overblikksbildet. Her er det et treff som svever i luften. Dette skyldes at hellingen og krumningen kun regnes ut innenfor vokslene, og det er estimert overflater med lav helling og krumning både over og under deteksjonen. Systemet er i utgangspunktet designet for utendørs bruk, så vi har ikke tatt hensyn til dette.



Figur 4.13: Resultat fra Vicon Room. Overblikk av den resulterende meshen, farget med helling og krumning. De hvite firkantene er detekterte landingsplasser.



Figur 4.14: Vicon Room. De detekterte landingsplassene registrert i den sanne punktskyen. Alle deteksjonene ser fornuftige ut.



Figur 4.15: Vicon Room. De detekterte landingsplassene registrert i den sanne punktskyen, sett fra en annen vinkel. Systemet klarte å identifisere landingsplassen dronen lander på og tar av fra i datasettet (de to deteksjonene til høyre).

Kapittel 5

Konklusjon

5.1 Oppsummering

For å kunne bruke autonome droner er det i mange tilfeller nødvendig at de har evne til å selv finne egnede landingsplasser. I denne oppgaven har vi derfor beskrevet et system for autonom landingsplassdeteksjon til droner med et monokulært kameraoppsett. Vi gikk gjennom de matematiske verktøyene og algoritmene vi har brukt til å modellere kamerageometri, og hvordan dette kan utnyttes til å navigere og hente ut de geometriske egenskapene til omgivelsene. Dette tillater oss å vurdere om et område kan landes i, gjennom tett dybde- og normalestimering. Vi gikk også gjennom hvordan vi kan sette sammen flere estimater for å inkrementelt bygge et kart vi kan finne landingsplasser i. Vi beskrev så hvordan vi har satt sammen systemet. Her gikk vi gjennom hvordan vi har forsøkt å ekskludere usikre dybdeestimater slik at vi får best mulig grunnlag for å estimere normaler, og hvordan vi har gjort hurtig normalestimering for å minimere utregningstid. Deretter gikk vi gjennom hvordan vi fusjonerer sammen flere observasjoner av samme område for å forbedre estimatene, og hvordan vi til slutt finner egnede landingsplasser basert på de samlede observasjonene.

Resultatene tilsier at vår metode potensielt er egnet til å finne trygge landingsplasser, da den har vist høy presisjon og brukbar sensitivitet i eksperimentene vi har kjørt. Dette åpner muligheten for å kunne bruke autonome droner til flere bruksområder enn det de har i dag. Vi har dog kun testet metoden på forhåndsinnspilte data, så testing i ekte miljøer vil være nødvendig for å verifisere metodens ytelse i praksis.

5.2 Videre arbeid

5.2.1 Navigasjon og baneplanlegging

Vi fikk dessverre ikke testet systemet på våre tenkte bruksområder med et navigasjonssystem, som ville bidratt med feil. Vi har likevel grunn til å tro at det vil gi brukbare resultater, da Voxblox vil håndtere feilestimater ved å bruke flere observasjoner, som vi viste med vår begrensede demonstrasjon

på Euroc-datasettet. Noe som også kunne vært en mulighet er å gjøre en lokal buntjustering på bildene som inngår i Plane Sweepen, for å sikre lokal konsekventhet i de relative positurene og på den måten forbedre dybdeestimatene.

Det vil også være mulig å inkludere baneplanlegging til landingsplass i systemet. En av hovedutfordringene er at Voxblox klarerer en radius rundt fartøyet hver gang man sender en punktsky, og vår dybdebildefrekvens er for lav til å konsekvent klarere banen vi har tatt. Dette vil føre til hull i ESDF-et som forhindrer planlegging. Dette kunne vært utforsket videre.

5.2.2 Semantikk

Fra resultatene ser vi at vi klarte å finne gode landingsplasser med vår metode, men det er klart tilfeller hvor systemet ikke er tilstrekkelig. Dersom veien hadde vært trafikkert hadde det ikke vært trygt å lande der, og systemet skiller ikke mellom vei og slette. Vann og dynamiske objekter som biler vil også kunne medføre problemer både i dybdeestimeringen og landingsplassklassifiseringen. Ved å introdusere semantisk segmentering av bildene ville vi kunne fjernet problematiske områder i bildene som himmel og vann for å forbedre dybdeestimatene. Man kunne også utvidet egenskapene til landingsplasskandidatene med semantiske egenskaper for å unngå landinger på uønskede flater som vei, hustak eller lignende.

Bibliografi

[1] Michael Burri mfl. «The EuRoC micro aerial vehicle datasets». I: *The International Journal of Robotics Research* (2016). DOI: 10.1177/0278364915620033. eprint: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.full.pdf+html. URL: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract.

[2] R.T. Collins. «A space-sweep approach to true multi-image matching». I: *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 1996, s. 358–363. DOI: 10.1109/CVPR.1996.517097.

[3] Jean-Marie «jmtrivial» Favreau. *Marching cube cases*. 2006. URL: https://commons.wikimedia.org/wiki/File:MarchingCubes.svg.

[4] Martin A. Fischler og Robert C. Bolles. «Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography». I: *Commun. ACM* 24.6 (jun. 1981), s. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: https://doi.org/10.1145/358669.358692.

[5] Michael Fonder og Marc Van Droogenbroeck. «Mid-Air: A multi-modal dataset for extremely low altitude drone flights». I: *Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*. Jun. 2019.

[6] Christian Forster, Matia Pizzoli og Davide Scaramuzza. «SVO: Fast Semi-Direct Monocular Visual Odometry». I: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2014, s. 15–22. DOI: 10.1109/ICRA.2014.6906584.

[7] Dorian Gálvez-López og J. D. Tardós. «Bags of Binary Words for Fast Place Recognition in Image Sequences». I: *IEEE Trans. Robot.* 28.5 (okt. 2012), s. 1188–1197. DOI: 10.1109/TRO.2012.2197158.

[8] Anne Goodchild og Jordan Toy. «Delivery by drone: An evaluation of unmanned aerial vehicle technology in reducing CO2 emissions in the delivery service industry». I: *Transportation Research Part D: Transport and Environment* 61 (2018). Innovative Approaches to Improve the Environmental Performance of Supply Chains and Freight Transportation Systems, s. 58–67. ISSN: 1361-9209. DOI: https://doi.org/10.1016/j.trd.2017.02.017. URL: https://www.sciencedirect.com/science/article/pii/S136192091630133X.

[9] Håvard Grip mfl. «Guidance and Control for a Mars Helicopter». I: jan. 2018. DOI: 10.2514/6.2018-1849.

[10] Nasser Gyagenda mfl. «A review of GNSS-independent UAV navigation techniques». I: *Robotics and Autonomous Systems* 152 (feb. 2022), s. 104069. DOI: 10.1016/j.robot.2022.104069.

[11] S. Holzer mfl. «Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images». I: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, s. 2684–2689. DOI: 10.1109/IROS.2012.6385999.

[12] Christian Häne mfl. «Real-Time Direct Dense Matching on Fisheye Images Using Plane-Sweeping Stereo». I: *2014 2nd International Conference on 3D Vision*. Bd. 1. 2014, s. 57–64. DOI: 10.1109/3DV.2014.77.

[13] Trym Vegard Haavardsholm. *A handbook in Visual SLAM*. Sep. 2021.

[14] M. Kaess mfl. «iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree». I: *"Int. J. Robot. Research"* 31.2 (feb. 2012), s. 217–236.

[15] Nallapaneni Manoj Kumar mfl. «On the technologies empowering drones for intelligent monitoring of solar photovoltaic power plants». I: *Procedia Computer Science* 133 (2018). International Conference on Robotics and Smart Manufacturing (RoSMa2018), s. 585–593. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.07.087. URL: https://www.sciencedirect.com/science/article/pii/S1877050918310366.

[16] Boris Lau, Christoph Sprunk og Wolfram Burgard. «Improved updating of Euclidean distance maps and Voronoi diagrams». I: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, s. 281–286. DOI: 10.1109/IROS.2010.5650794.

[17] S. Leutenegger mfl. «Keyframe-Based Visual-Inertial SLAM using Nonlinear Optimization». I: *Int. J. Robot. Research* (2015).

[18] William E. Lorensen og Harvey E. Cline. «Marching Cubes: A High Resolution 3D Surface Construction Algorithm». I: *SIGGRAPH Comput. Graph.* 21.4 (aug. 1987), s. 163–169. ISSN: 0097-8930. DOI: 10.1145/37402.37422. URL: https://doi.org/10.1145/37402.37422.

[19] Sven Mayer, Lars Lischke og Paweł Woźniak. «Drones for Search and Rescue». I: mai 2019.

[20] David Nistér. «An efficient solution to the five-point relative pose problem». I: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26 (2004), s. 756–770.

[21] Helen Oleynikova mfl. «Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning». I: sep. 2017, s. 1366–1373. DOI: 10.1109/IROS.2017.8202315.

[22] Fatih Porikli og Oncel Tuzel. «Fast Construction of Covariance Matrices for Arbitrary Size Image Windows». I: okt. 2006, s. 1581–1584. DOI: 10.1109/ICIP.2006.312610.

[23] Md Shah Alam og Jared Oluoch. «A survey of safe landing zone detection techniques for autonomous unmanned aerial vehicles (UAVs)». I: *Expert Systems with Applications* 179 (2021), s. 115091. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2021.115091. URL: https://www.sciencedirect.com/science/article/pii/S0957417421005327.

[24] Sanjiv Singh og Sebastian Scherer. «Pushing the Envelope - Fast and safe landing by autonomous rotorcraft at unprepared sites». I: *Annual Forum Proceedings - AHS International* 4 (jan. 2013), s. 2877–2881.

[25] Sanjiv Singh og Sebastian Scherer. «Pushing the Envelope - Fast and safe landing by autonomous rotorcraft at unprepared sites». I: *Annual Forum Proceedings - AHS International* 4 (jan. 2013), s. 2877–2881.

[26] KN Tahar og SS Kamarudin. «UAV ONBOARD GPS IN POSITIONING DETERMINATION.» I: *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* 41 (2016).

[27] Evgueni Tcherniaev. «Marching Cubes 33: Construction of Topologically Correct Isosurfaces». I: (jan. 1996).

[28] Todd Templeton mfl. «Autonomous Vision-based Landing and Terrain Mapping Using an MPC-controlled Unmanned Rotorcraft». I: mai 2007, s. 1349–1356. DOI: 10.1109/ROBOT.2007.363172.

[29] J. Weng, P. Cohen og M. Herniou. «Camera calibration with distortion models and accuracy evaluation». I: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.10 (1992), s. 965–980. DOI: 10.1109/34.159901.

[30] Long Xin mfl. «Vision-Based Autonomous Landing for the UAV: A Review». I: *Aerospace* 9 (okt. 2022), s. 634. DOI: 10.3390/aerospace9110634.

[31] Tao Yang mfl. «Monocular Vision SLAM-Based UAV Autonomous Landing in Emergencies and Unknown Environments». I: *Electronics* 7.5 (mai 2018), s. 73. ISSN: 2079-9292. DOI: 10.3390/electronics7050073. URL: http://dx.doi.org/10.3390/electronics7050073.