Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Informasjonsteknologi, automatisering og signalbehandling	Høst 2014 semesteret, 2014 Åpen
Forfatter: Jan Bjørge Scheyka Løvland	…… (signatur forfatter)
Fagansvarlig: Karl Skretting Veileder(e): Karl Skretting	
Tittel på masteroppgaven: Stereosyn anvendt til robotprogrammering, implementert i MATLAB Engelsk tittel: Stereo vision applied to robot programming, implemented in MATLAB	
Studiepoeng: 30	
Emneord: Bildebehandling, stereosyn, robotprogrammering, ABB, punktsky, Punktreduksjon, MATLAB, Rapid	Sidetall: 113 + vedlegg/annet: 13 Stavanger, 15-12/2014

Stereosyn anvendt til robotprogrammering, implementert i MATLAB.

Jan Bjørge Løvland

Institutt for data- og elektroteknikk
Universitetet i Stavanger

Sammendrag

Bildebehandling er et fagfelt som er på god vei inn i automasjon i industrien. Kombinasjonen av robot og bildebehandling gir store muligheter for adaptive robotsystemer. Spesielt med tanke på å forenkle programmeringen. I denne oppgaven har det blitt sett på hvordan bildebehandling kan anvendes til forenklet robotprogrammering.

Oppgaven er skrevet på eget initiativ og har sterke røtter til et tidligere prosjekt.[1] Samtidig har ABB et ønske om å utforske forenklet robotprogramming.

I oppgaven har en tatt i brukt utstyret som er tilgjengelig på robotlaben hos UiS. Hvor av dette er kamera, ABB robot og kontroller.

Antall punkter som blir plukket ut blir i denne oppgaven blir ofte et par tusen, en må derfor se på hvordan mengden kan reduseres uten på bekostning av nøyaktighet. To ganske kjente metoder ble undersøk sammen med en selvutviklet metode ble sett på. En MATLAB funksjon ble dermed skrev som enkelt implementerer punktreduksjons algoritmene.

En sentral del av oppgaven var å lage en testbane som skal brukes til å teste teori og implementasjon. Testbanen er gitt som spillet ''Dont touch the wire", hvor målet er å føre en ring fra et punkt til et annet uten å komme nær banen.

Testbanen er representert av en punktsky, hvor en senere pukker ut aktuelle punkter fra punktskyen. Nøyaktigheten er målt til omtrent 1cm som gir god nok nøyaktighet til oppgavens formål. Dette gjør at oppgaven betraktes som løst.

[1] ABB robot som plotter, `http://goo.gl/a8TOIk`

Forord

Jeg vil takke veilederen min, Karl Sketting for god veiledning, råd og tilbakemelding gjennom arbeidet mitt på masteroppgaven.

Jeg vil også takke Ståle Freyer og Ivar Austvoll for hjelp når jeg har møtt på problemer under oppgaven.

Til sist vil jeg takke Cathrine Lien for støtte og korrekturlesing, samt familie for støtte gjennom mine år ved UiS.

Jan Bjørge Scheyka Løvland, Stavanger 15/12/2014

Innhold

1 Innledning **1**
1.1 Oppgavebeskrivelse . . . 2

2 Teori **5**
2.1 Lys og farger . . . 5
2.2 Fargebilder . . . 5
2.3 Rigid bevegelse . . . 7
2.3.1 Rotasjon . . . 7
2.3.2 Eksempel . . . 7
2.3.3 Sammensatte rotasjoner . . . 8
2.3.4 **TR** - Matrisen . . . 8
2.3.5 **TR** - eksempel . . . 9
2.3.6 Kvaternion . . . 10
2.4 Nålehulls kameramodell . . . 11
2.4.1 Internmatrisen **K** . . . 13
2.4.2 Metode for kalibrering . . . 13
2.4.3 Linseforvregning . . . 16
2.5 Stereo Kamera . . . 17
2.5.1 Kalibrere stereorigg . . . 18
2.5.2 Bilde likeretting . . . 19
2.5.3 Disparitet . . . 20
2.5.4 Rekonstruksjonsfeil . . . 21
2.6 Minste kvadraters metode, for en linje . . . 24
2.6.1 Minste kvadrat filtrering . . . 25
2.6.2 Flytskjema for punktRed funksjon . . . 26
2.6.3 Eksperimentell sammenligning av KF og smoothing-filter . . . 28
2.7 Punktreduksjon . . . 30
2.7.1 Lineærreduksjon . . . 30
2.7.2 K-means . . . 31
2.7.3 Krumning . . . 32

2.7.4 Punkter og areal . . . 33
2.7.5 Arealtransformen til f(x) . . . 35
2.7.6 Eksperimentell sammenligning av A-transform og normalisert A-transform . . . 38
2.7.7 Volumtransform . . . 39
2.8 Volumtransform punktreduksjon . . . 41
2.9 Eksperimentell sammenligning av Lineær-, Kmeans- og Arealpunktreduksjon . . . 43

3 Implementering 48
3.1 Kamera brukt i oppgaven . . . 49
3.2 Utviling av testverktøy . . . 49
3.2.1 Versjon 1 og 2 . . . 50
3.2.2 Versjon 3 - ”Gaffelen” . . . 51
3.3 Ta bilder . . . 53
3.4 Estimere av Stereo Parametere . . . 53
3.5 Bruk av Stereo Parametere . . . 59
3.6 Last inn bildepar der banen er motivet . . . 59
3.7 Bilde likeretting . . . 60
3.8 Disparitet . . . 62
3.9 Reconstruct Scene . . . 64
3.10 Binært bilde . . . 65
3.11 Finne skjelettet . . . 67
3.12 Sjelettet i punktskyen . . . 70
3.13 Koordianter i felles matrise . . . 72
3.14 Sortere datasett . . . 74
3.15 Kaste dårlig punkter . . . 74
3.16 Punkt glatting . . . 75
3.17 Punktreduksjon . . . 77
3.18 Orientering . . . 78
3.19 Overføring til robot . . . 80
3.20 ABB - Rapid . . . 81
3.21 Utføre bevegelse . . . 83
3.22 Forfatter prøver . . . 83

4 Resultater 84
4.1 Planlagt bane . . . 84
4.2 Feilmargin . . . 84
4.2.1 Målt feilmaring . . . 85
4.3 Kommentarer til implementering . . . 87
4.4 Punktreduksjon . . . 87
4.5 Filmklipp . . . 92

5 Konklusjon og videre arbeid 93

Referanser **95**

A Appendix - MATLAB **98**

B Appendix - Rapid **108**

Figurer

1.1 ”Don’t touch the wire” - Målet i spillet er å lede kroken fra start til slutten uten å komme inntil banen. Bilde er hentet fra `http://tinybearstoyandpartyhire.co.uk/small-party-games/dont-buzz-the-wire/` 3
1.2 Robotlaben på UiS hvor en har ABB robotene Norbert med griperverktøyet til venstre og Rudolf til høyre med sugekoppverktøy montert. 4

2.1 En kombinerer grunnfarger for å produsere sekundærfarger. Sekundærfargen er altså en vektet sum av grunnfargene. Figur er hentet fra ; *wikipedia* [1] 6
2.2 Rotasjon om y_0 og translasjon t_x. 9
2.3 Nålehulls kameraet, viser hvordan et tredimensjonalt objekt blir prosjektert ned til et plan. Bildet er hentet fra : `http://en.wikipedia.org/wiki/Pinhole_camera`. 11
2.4 Nålehulls kameramodell. Legg merke til at en har definert et venstrehånds koordinatsystem, ettersom jeg ønsker at Z skal peke ut fra kamera. 12
2.5 Bilde viser to typer linseforvrengning nålepute og tønneforvrengingen. Hvor høyre bilde er forvrengingen som oppstår i kamera brukt i denne oppgaven. Bilde er hentet fra `http://se.mathworks.com/help/vision/ug/single-camera-calibrator-app.html` 16
2.6 Venstre- og høyre kamera med avstand t. 17
2.7 Venstre- og høyre kamera med oppsett t. 19
2.8 I denne oppgaven er avstanden mellom kameraene nesten bare en ren translasjon å dermed vil de epipolare linjene være nesten horisontale. Men dette er ikke godt nok hvis en skal løse korrespondanse problemet effektivt er en avhengig av perfekt horisontale epipolare linjer. 20
2.9 Studerer en de to bildene ovenfor vil en se at den epipolare linjene er mer horisontale i nederste bildet en øverste. Med unntak av to av linjene som er feilklassifisert. 20

2.10 Bildeserien viser hvordan en kan finne disparitet til et objekt ved forskyvning av det ene bildet i forhold til det andre. Legg merke til at banen er nesten helt svart, som vil si lik null. Ved en forskyvning på 45 piksler finnes banen. Linken i fotnote 2 på side 20 anbefales for en bedre visuell fremstilling. 21
2.11 Pikselen som er assosiert med P_2 gir større usikkerhet en pikselen som er assosiert med P_1 22
2.12 Dette viser at ved relativt liten feilmargin i x_l og x_r er det ca. 4 ganger større feilmargin i dybdeestimeringen en i XY-planet. 23
2.13 Punktene P_1 til P_N og tilhørende avvik. 24
2.14 Første plot viser originalt- og behandlet signal, hvor en ser at områder hvor det er liten ending eller linjer beholder egenskapene sine. Mens områder med mye ending blir prøvd tilnærmet linje. Andre plot viser at filteret konvergerer etter ca. 45 iterasjoner 26
2.15 Flytskjema til punktRed-funksjonen 27
2.16 Plottene viser de forskjellige testsignalene 28
2.17 Signalene som er behandlet smootingfilter og KF. Studerer man grafen nøye ser en tendenser til at smooth vandrer rundt det originale signalet. . 29
2.18 Plottene viser tier-logaritmen til original- minus behandletsignal i andre potens. KF filtrere ser ut til å være mer stabilt og en hel del glattere . . . 29
2.19 Figuren viser opprinnelig signal og redusert signal fra 1800 punkter til 25 punker. 30
2.20 Figuren viser opprinnelig signal og redusert signalet fra 1800 punkter til 25 punkter med K-mean clustering. 31
2.21 Den øverste grafen viser hvordan arealet mellom 3 punkter utvikles. Grafen under viser arealet som er fremhevet i farger. Fargen har ikke noe med verdien å gjøre, bare estetisk. 33
2.22 A er et simpelt polygon, B er ikke. 34
2.23 $A(x_0)$ som en funksjon av $f(x_0 \pm \delta)$ 35
2.24 Arealene $A_{-1 \to 1}$ er gitt av 2.46. Studerer en figuren litt nærmere legges det merke til at A_{-1} er en del større en A_0. Dette har sitt opphav i at funksjonen blir mer lineær om x_0. 36
2.25 D som funksjon av δ og $f(x_0 \pm \delta)$ 37
2.26 $f(x) = x^3$ 38
2.27 $f(x) = sin(2x)$ 38
2.28 Vektorene **a,b,c** definerer et parallellepiped. Bilde er hentet fra [`http://en.wikipedia.org/wiki/Parallelepiped`] 39
2.29 Volumpunktredusjon anvendt på testsignal. Hvor fargene gjenspeiler originalt og behandlet datasett. Hvor en bør legge merke til at punktene nå har blitt lagt litt mer strategisk med tanke på at toppene har fått flere punkter som beskriver denne delen av kurven. 42
2.30 Resultatet fra MATLAB-koden gitt på forrige side, det blir vist Areal vs Lineær punktreduksjon. 45

2.31 Resultatet fra MATLAB-koden gitt på forrige side, det blir vist K-mean vs Lineær punktreduksjon. 46
2.32 I tabellen over er resultatene fra MATLAB-koden. 46
2.33 Resultat ved at punktreduksjon $SNR = 28.8$ 47

3.1 μEye2 utviklet av IDS her sammen med M-sjokolade for å illustrere den lille størrelsen på kameraet. Bilde er hentet fra `http://en.ids-imaging.com/` 49
3.2 Testoppsettet fra forskjellige vinkler 50
3.3 Skissen ble laget med utgangpunkt i 10mm ytre diameter på røret. Det er også lagt til en liten sikkerhetsfaktorer i usikkerhetsberegningene. Fargene representerer forskjellige deler av verktøyet som blir festet til tool0 på roboten, Gripper, Kamera og Gaffelen, størrelse forholde mellom dem stemmer ikke. Størrelsesforholdet mellom delene stemmer ikke. 51
3.4 Testoppsettet med "Gaffelen" fra forskjellige vinkler. 52
3.5 Fire bildesett av totalt tolv som ble brukt til å kalibrere stereoriggen. . . 55
3.6 Størrelsen på kvardene på sjakkbrettet er målt til $32mm$. En trykker "ok" i figuren over og MATLAB begynner å analyse bilde. Hvor den sjekker at filnavn stemmer og finner hjørnene til alle sjakkbrettene. 56
3.7 I venstre del av bildet ser en de forskjellige bildesettene, opp kan en velge hvilke parametere som ønskes estimert hvor "skew" nå er det samme som gamma fra 2.4.2. Radial distorsjon blir valgt til å estimers til to koeffisienter i henholdt til 2.4.3. Når en fornøyd med valgene trykkes det på "calibarte" for å starte prosessen. 57
3.8 Kvaliteten på kalibreringen er blir vist i "reprojection error" vinduet. Feilen som blir vist i plottet er avstanden mellom punkt funnet i bilde og punktet som blir estimert igjennom modellen. Som er vist i neste figur. Figuren er hentet fra [10] 58
3.9 Bildet oppe til venstre er kalt "basis", mens bildet oppe til høyre er kalt "20mmx". Det nederste bildet er et sammensatt RGB-bilde som viser "basis" og "20mmx", hvor "basis" er den grønne banen og "20mmx" er den lilla banen. Fargene er endret for å lett skille bildene fra vedrandre. 60
3.10 Originalt bilde og korrigert bilde i forskjellige fargebånd. 61
3.11 Absolutt verdi |Originalt - korrigert| bilde 61
3.12 "Rectifert" bildepar i forskjellige fargebånd. 62
3.13 Som oppgitt av parameteren vises forskyvningen mellom bildene 63
3.14 Scenen er rekonstruert, hvor det ikke er tatt hensyn til forvrengingen. . . 64
3.15 Punktskyen er funnet ved å bruke `'reconstructScene'`. 65
3.16 Punkte som er satt til `'NaN'` blir grå. 66
3.17 Resultat av `im2bw` ved å sette `'NaN'` pikslene til gjennomsnittetsverdien av bildet. 66
3.18 Resultat av `im2bw` ved å sette `'NaN'` pikslene til 0. 66
3.19 Spur operasjonen er anvendt i venstre sett og resultatet i høyre sett. . . . 67

3.20 En rett linje blir brukt til å demonstrere hva skel operatoren gjør med et binært bilde . . . 68
3.21 Skel operatoren finner medianen til linjen . . . 68
3.22 Den binære operasjonen XOR blir brukt på de to øverste bilden for å fremheve at en finner medianen til den binære strukturen som var utgangspunktet. . . . 68
3.23 Figuren ser diskontinuerlig ut og er et resultalt av reduksjonen av størrelsen. 69
3.24 Bildet er tatt med venstre kamera, hvor en legger på skjelettet. Dette viser at en har klart å finne senter av banen relativt bra. . . . 70
3.25 Alle pikslene som har verdien 0 i det binære bilde blir nå korresponderende punkt satt til `'NaN'` i punktskyen. Hvor er dermed vil så igjen med et skjelett av den ønskede banen. . . . 71
3.26 Valgt startpunkt. . . . 73
3.27 Punkt $P_1 \rightarrow P_4$, blir brukt til å demonstere problemet ved å bruke euklidsk sortering. . . . 75
3.28 Visualisert punktskyen, før glatting og etter glatting . . . 76
3.29 Matrisen $[\mathbf{R}|\mathbf{t}]$ er sammenhengen mello tool0 og kameraoptiskesenter. . . . 80
3.30 Gjennom FlexPendanten får brukeren to valg; Ta bilde eller Kjør bane. . . 82

4.1 TCP punktet er plassert 17.5 cm. langs tool0 sin Z-akse. . . . 85
4.2 Distansen mellom reelt og estimert punkt . . . 86
4.3 Histogram til målt avvik, søylene er i $1[mm]$ inkrement. . . . 86
4.4 Tiden er målt som gjennomsnittet av ti tidsmålinger, med relativ liten varians. . . . 88
4.5 Lineær punktredusjon på virklig datasett. Original, Behandlet . . . 89
4.6 Lineær punktredusjon på virklig datasett, annen synsvinkel. Original, Behandlet . . . 89
4.7 K-mean punktredusjon på virklig datasett. Original, Behandlet . . . 90
4.8 K-mean punktredusjon på virklig datasett, annen synsvinkel. Original, Behandlet . . . 90
4.9 Volum-punktredusjon på virklig datasett. Orginal, Behandlet . . . 91
4.10 Volum-punktredusjon på virklig datasett, annen synsvinkel. Original, Behandlet . . . 91

Algoritmer

1 Punktglatting 26
2 Volumtransformen - punktreduksjon 41

1

Innledning

Robotprogrammering kan være en svært tid krevende oppgave. I enkelte tilfeller kan en jobbe i et koordinatsystem, mens i andre tilfeller kan det være flere koordinatsystemer. Dette gjør at det har blitt utviklet en rekke verktøy for å gjøre dette lettere (f.eks ABBs Robotstudio). Selv med disse verktøyene må en fremdeles ha kontroll på posisjon, orientering, fart og geometriske begrensinger. For personer som ikke er kjent med robotor kan disse verktøyene fremdeles virke overveldende som følge av alt man må ha kontroll på.

Ved ABB robotics på Bryne blir det for tiden eksperimentert med en sensor med seks frihetsgrader, som skal anvendes til forenklet robotprogrammering. Jeg ønsket å ta steget videre med datasyn (eng.:computer vision) anvendt til forenklet robotprogramming.

Det er nesten umulig å snakke om datasyn uten å sammenligne med naturen. Mennesker og de fleste andre dyr har to øyne som brukes til innhenting av informasjon. Denne informasjonen videresendes til hjernen hvor et bilde blir dannet. Mennesket, men også dyr har lett for å kjenne igjen mønstre. Vi er så flinke til å kjenne igjen mønstre at uten å tenke over det, kjenner vi igjen forskjellige personer, steder, ord. Vi er så gode til å kjenne igjen mønstrene at vi kan se hvor en brikke i et puslespill skal ligge, selv om vi ikke har sett løsningen på puslespillet.

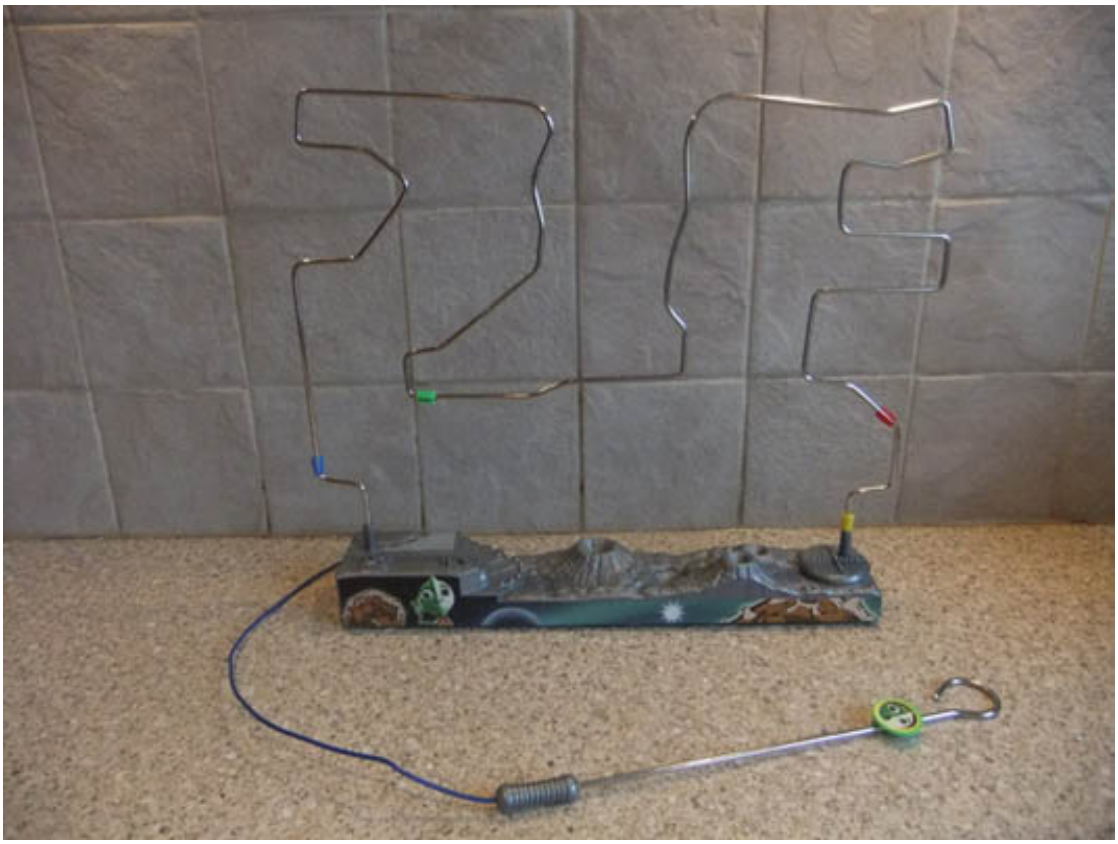
Dette har gitt oss muligheten til å få en oppfatting av verden rundt oss. Det gir oss muligheten til å observere hendelser på avstand, slik at vi kan bedømme hvilket steg vi skal ta videre. Vi oppfatter lett hva som er viktig informasjon og hva som er uvesentlig, og ha fokus på forskjellige oppgaver deretter. Disse egenskapene som vi selv tar for gitt, har vist seg å være noe av det vanskeligste man kan forsøke å lære en datamaskin. Men gjennom årevis med forskning har en klart å komme frem til en rekke metoder som gjør det bra i enkelte tilfeller. Som for eksempel ansikts gjenkjenning. Ett av problemene blir

dermed å finne hvilken del av bildet som inneholder relevant informasjon.

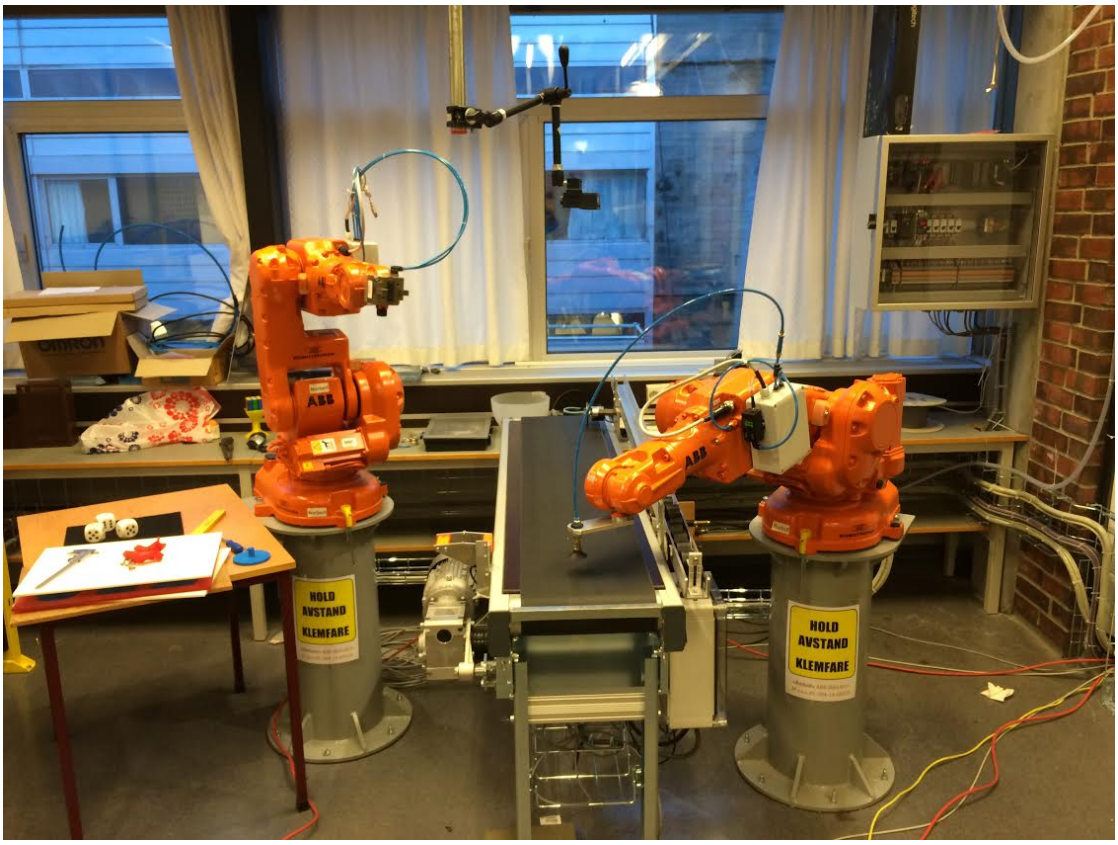
1.1 Oppgavebeskrivelse

Oppgaven som er valgt i dette prosjektet er å ta i bruk datasyn til å løse en bane som roboten skal bevege seg langs. For å prøve og lage en ''wow" faktor plukket jeg ut spillet "Don't touch the wire", hvor målet er føre en ring langs en bane uten å komme nær banen. Ettersom det er behov for å måle avstand, ble det bestemt å ta i bruk stereosyn. Det ble valgt å utføre stereosyn med et noe utradisjonelt oppsett. En benytter seg av ett kamera som er montert på robotarmen, som ved hjelp av robotens gode evne til å repetere, flyttes mellom to punkter for å få to bilder som er forskjøvet mellom hverandre. Bildebehandling må deretter til for å konstruere en 3D punktsky hvor en må plukke ut aktuelle punkter på banen. Roboten som blir brukt er en ABB IRB 140. UiS har to av denne modellen på labben, hvor jeg har valgt og bruke roboten kalt "Norbert". Dette er grunnet at gripper verktøyet og kamera allerede er montert. Dermed er også kamera for oppgaven gitt. Det vil derfor være behov for å lage et testverktøy som kan festes på roboten, samt en testbane som roboten skal løse. Det ble vurdert å kjøpe en ferdig av standard leke, men en valgte i stedet et tynt kobber rør som en har muligheten til å utforme selv.

Jeg har valgt å begrense oppgaven til at banen skal holde seg til et tilnærmet plan mellom start- og sluttpunktet. Det er videre satt som en selvfølge at spillobjektets fulle utstrekning er innenfor kameraets synsfelt. I oppgaven antar jeg også at banen som skal løses ikke har noen krapp brekning i dybden ettersom dette kan føre til at deler av banen ikke blir synlig. Som følge av robotens rekkevidde er en annen begrensing at objektet må befinne seg minst en meter fra robotens utgangsposisjon.



Figur 1.1: "Don't touch the wire" - Målet i spillet er å lede kroken fra start til slutten uten å komme inntil banen. Bilde er hentet fra `http://tinybearstoyandpartyhire.co.uk/small-party-games/dont-buzz-the-wire/`



Figur 1.2: Robotlaben på UiS hvor en har ABB robotene Norbert med griperverktøyet til venstre og Rudolf til høyre med sugekoppverktøy montert.

2

Teori

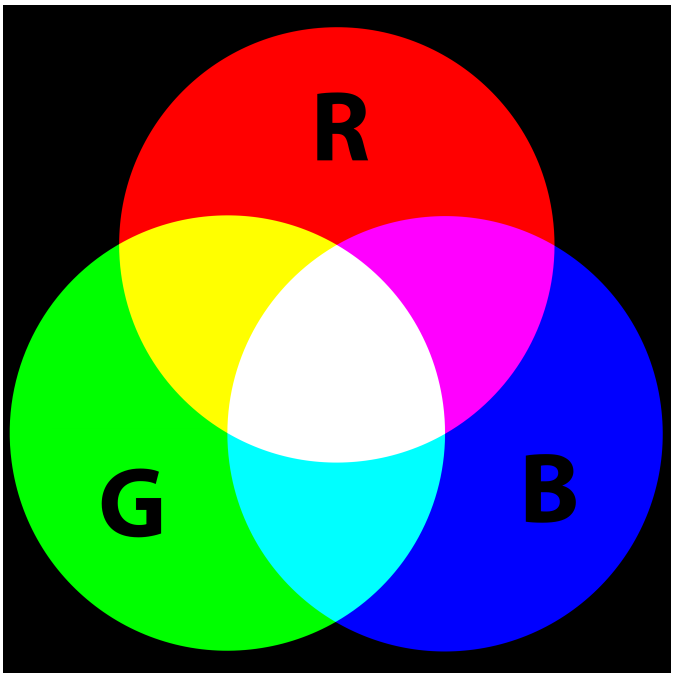
2.1 Lys og farger

Mennesker ser farge som et resultat av at øyet har utviklet seg til å ha tre typer fotoreseptor; S-cone, M-cone og L-cone. Som er forkortelser for small-, medium og large-cone. Dette er celler som sitter bak i øyeeplet og reagerer på lys av forskjellige bølgelengder. S-conene har sin maksimale følsomhet for lys av bølgelengdene 450-500nm, M-conene 530-630nm og L-conene 500-700nm. Disse bølgelegende kjenner en igjen fra fargespekteret som blått, grønn og rødt. [2]

2.2 Fargebilder

Fargemodellen RGB (Red, Green, Blue) er mye brukt i fargeteori. RGB modellen tar utgangspunkt i at alle farger kan beskrives som en vektet sum av de tre grunnfargene. I figuren under (2.1) ser en hvordan blandinger av forskjellige grunnfarger produserer sekundærfarger. Men datamaskiner kan ikke jobbe med analoge farger, ettersom datamskiner kun takler diskrete verdier. Det er derfor nødvendig med en diskretisert versjon av RGB-modellen.[2]

Tidlig på 1950-tallet ble det gjort forsøk som viste at mennesket til liten grad klarte å skille på mer enn 250 toner av samme farge. Dette blir ofte lagt til grunne i implementeringer av RGB. En gir rødfargen 8bit, grønn fargen 8bit og blåfargen 8bit. Man står dermed med 24bit som representere en piksel. Med 8bit gir 256 verdier, om en ønsker 0 som en del av tall mengden $8bit = \{0,1,2,3,..,255\}$. Totalt mulige farger per piksel blir derfor $2^{24} = 16.777.216$ farger. At man kan representere farger som en sum av tre grunnfarger er mange praktiske anvendelser som en ser i hverdagen. RGB-modellen blir brukt i LCD- og LED-skjermer som er i stor grad en del av hverdagen.[1, 3]



Figur 2.1: En kombinerer grunnfarger for å produsere sekundærfarger. Sekundærfargen er altså en vektet sum av grunnfargene. Figur er hentet fra ; *wikipedia* [1]

2.3 Rigid bevegelse

2.3.1 Rotasjon

En kan definere rotasjonsmatrisen $\mathbf{R} \in SO(n)$, Special Orthogonal Group, hvor n er dimensjon. Ofte vil $n = 3$ som spesifiserer rotasjon i 3D rommet. En matrise må oppfylle følgende kriterier for å kunne kalles en rotasjonsmatrisen.[4]

1. Rader og kolonner er gjensidig ortogonale.
2. Rader og kolonner er enhetsvektor.
3. $\mathbf{R}^{-1} = \mathbf{R}^T$, der $\mathbf{R}^T$ er den transponerte av matrisen $\mathbf{R}$.
4. Determinanten til $\mathbf{R} = 1$

Basisrotasjon er rotasjon rundt en akse. F.eks. $\mathbf{R}_{x,\theta}$ er rotasjon om x-aksen med θ. Basis rotasjonsmatrisene er :

$$\mathbf{R}_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta)) \end{bmatrix} \tag{2.1}$$

$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2.2}$$

$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Basis rotasjonsmatrisene har noen egenskaper :

1. $\mathbf{R}_{alle\ akser,\theta} = \mathbf{I}$, for $\theta = 2\pi n$, $n \in \mathbb{Z}$
2. $\mathbf{R}_{x,\theta} + \mathbf{R}_{x,\gamma} = \mathbf{R}_{x,\theta+\gamma}$
3. Fra 1 og 2 kan det vises at $\mathbf{R}_{x,\theta}^{-1} = \mathbf{R}_{x,-\theta}$

2.3.2 Eksempel

La p_1 være et punkt i rommet gitt av (x_1,y_1,z_1) i koordinatsystem K_1. En kan beskrive punktet relativt til koordinatsystem K_0 om en kjenner $\mathbf{R}_0^1$, som beskriver rotasion mellom kordinatsystem K_0 og K_1.

$$p_0 = \mathbf{R}_0^1 p_1 \tag{2.4}$$

2.3.3 Sammensatte rotasjoner

I mange tilfeller har man rotasjon om flere akser. Disse kan beskrives med en kombinasjon av basisrotasjoner. Ettersom matrisemultiplikasjon ikke er kommutativ, er det viktig at en tenker seg om før en setter opp basisrotasjons matrisene.

Hvis en rotasjon beskrives i forhold til nåværende koordinatsystem så postmultipliseres rotasjonsmatrisene. Om en dermed skulle ønske å rotere rundt et fast koordinatsystem premultipliseres det.

Eksempel

En ønsker å rotere ϕ grader om z aksen, deretter rundt NYE x aksen θ grader.

En tar utgangspunkt i basis rotasjonsmatrisene, som gir matrisen $\mathbf{R}$ som beskriver rotasion gitt i problemstillingen.

$$\mathbf{R} = \mathbf{R}_{z,\phi}\mathbf{R}_{x,\theta} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.5}$$

2.3.4 TR - Matrisen

Ettersom grunnlaget for rotasjon er lagt, kan man kombinere rotasjon og translasjon i matrisen $\mathbf{TR}$ kalt translasjons- rotasjonsmatrisen. Hvor en utformer $\mathbf{R}$ i henhold med tidligere gjennomgått teori. Translasjonen defineres ut opprinnelig koordinatsystem.

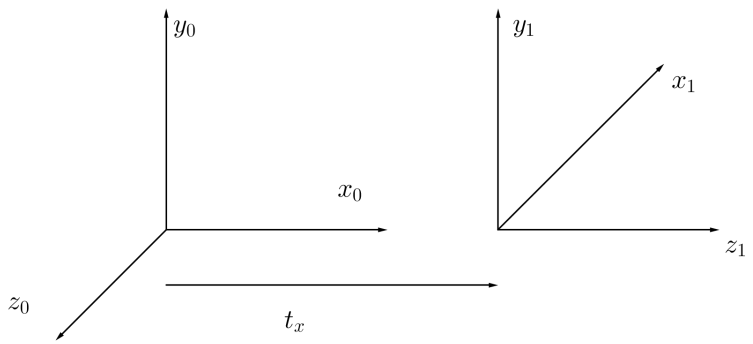
$$\mathbf{TR} = [\mathbf{R}\ \mathbf{t}] = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

2.3.5 TR - eksempel

Her et eksempel på hvordan **TR** matrisen vil se ut om en vil beskrive den rigide bevegelsen i figur 2.2. En ser to koordinatsystem, hvor forholdet mellom dem kan bli uttrykt som en **TR** matrise. Fra figuren ser en at koordinatsystemet K_1 er forskjøvet t_x i forhold til koordinatsystemet K_0, i tillegg er det rotert om y-aksen i K_0.

En kan starter med å finne translasjonen $\mathbf{t} = [t_x,0,0]^T$, samt at en ser rotasjonen rundt y-aksen i K_0. Fra basisrotasjonene passer 2.2, vinkelen er satt til å være 180° eller $\pi/2$ rad.

$$\mathbf{TR} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} = \begin{bmatrix} \cos(\frac{\pi}{2}) & 0 & \sin(\frac{\pi}{2}) & t_x \\ 0 & 1 & 0 & 0 \\ -\sin(\frac{\pi}{2}) & 0 & \cos(\frac{\pi}{2}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & t_x \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$



Figur 2.2: Rotasjon om y_0 og translasjon t_x.

2.3.6 Kvaternion

Innen robotteknologi brukes ofte kvaternioner til å beskrive orientering i rommet. Kvaternioner ble oppdaget av Sir William Rowan Hamilton på 1800 tallet. Et kvaternion har formen $z = \alpha_1 + \alpha_2 i + \alpha_3 j + \alpha_4 k$, hvor α_i er reelle koeffisienter og basiselementene i, j & k , hvor $i^2 = j^2 = k^2 = ijk = -1$. Når kvaternioner anvendes til rotasjon må koeffisientene oppfylle $\sqrt{\alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2} = 1$ for å unngå utmeisling(eng.:shearing). [5][6].

Sammenhengen melleom et kvaternionen $z = \alpha_1 + \alpha_2 i + \alpha_3 j + \alpha_4 k$ og rotasionsmatrisen $\mathbf{R}$ er gitt som[7]:

$$\mathbf{R} = \begin{bmatrix} 1 - 2(\alpha_2^2 + \alpha_3^2) & 2(\alpha_1\alpha_2 - \alpha_0\alpha_3) & 2(\alpha_0\alpha_2 + \alpha_1\alpha_3) \\ 2(\alpha_1\alpha_2 + \alpha_0\alpha_3) & 1 - 2(\alpha_1^2 + \alpha_3^2) & 2(\alpha_2\alpha_3 - \alpha_0\alpha_1) \\ 2(\alpha_1\alpha_3 - \alpha_0\alpha_2) & 2(\alpha_0\alpha_1 + \alpha_2\alpha_3) & 1 - 2(\alpha_1^2 + \alpha_2^2) \end{bmatrix} \tag{2.8}$$

Eksempel

Hvis $z = 1 + 0i + 0j + 0k$, hva blir $\mathbf{R}$?

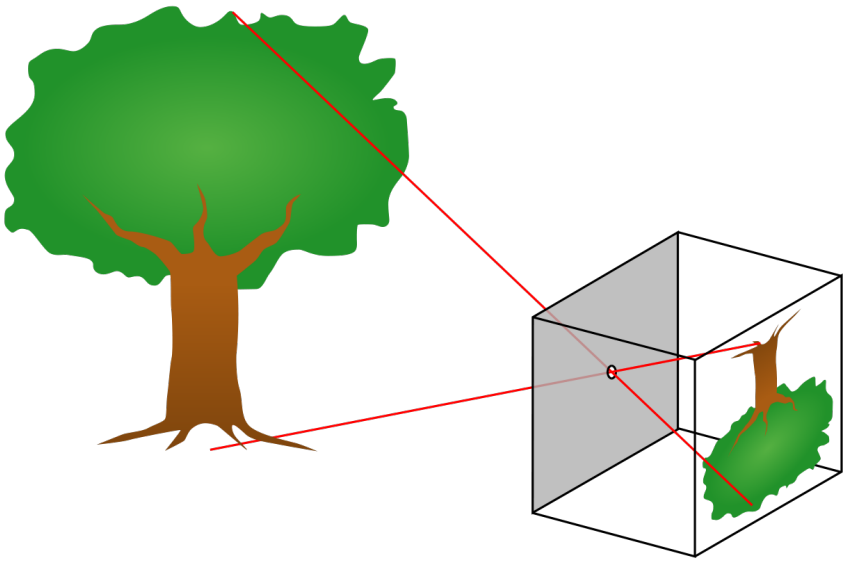
$$\mathbf{R} = \begin{bmatrix} 1 - 2(0^2 + 0^2) & 2(00 - 00) & 2(00 + 00) \\ 2(00 + 00) & 1 - 2(0^2 + 0^2) & 2(00 - 00) \\ 2(00 - 00) & 2(00 + 00) & 1 - 2(0^2 + 0^2) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{I} \tag{2.9}$$

En har vist at hvis $z = 1 + 0i + 0j + 0k$, så er dette det samme som rotasjonsmatrisen er en identitetsmatrisen.

2.4 Nålehulls kameramodell

Nålehulls kameramodellen (eng.:Pin Hole camera) er blant de mest anvendte modellene for å modellere et kamera. Modellen tar utgangspunkt i et lite hull i en lukket boks (kamerahuset), dette tvinger lys utenfra igjennom punkt O, det optiske senter. Dette medfører at punktet $P = [X,Y,Z]$ utenfor kamerahuset er direkte relatert gjennom en linje til punktet $p = [x,y]$ i bildeplanet innenfor kamerahuset. Punktet P som er i rommet blir prosjektert til et plan. Denne prosessen kalles "image formation". [3][2]

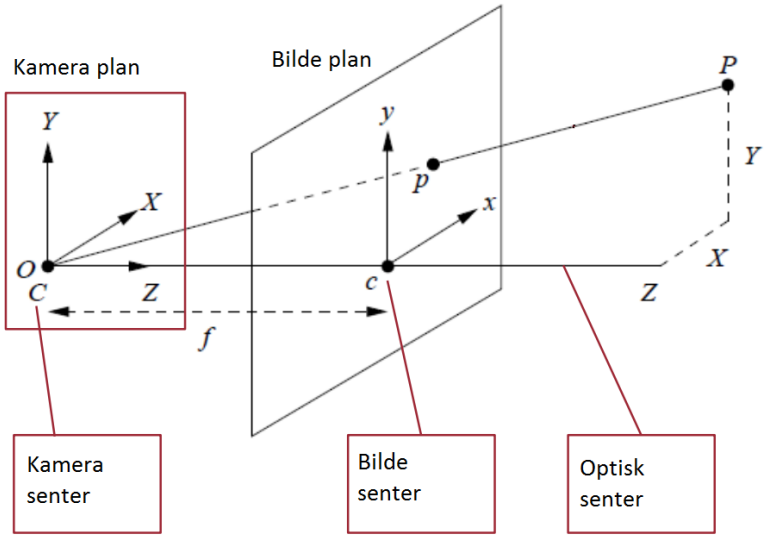
Fra figur 2.3 ser en at alle punkter blir speilvendt om O, ergo, høyre blir venstre, opp blir ned osv. Derfor velges det et bildeplan som ligger foran det optiske senter i planet $Z = f$. Dette gjør at en slipper speilvending.



Figur 2.3: Nålehulls kameraet, viser hvordan et tredimensjonalt objekt blir prosjektert ned til et plan. Bildet er hentet fra : `http://en.wikipedia.org/wiki/Pinhole_camera`.

Ettersom en nå har fått definert noen begreper, er det naturlig å se om en klarer å komme frem til noen ligninger som beskriver modellen litt mer. I figur 2.4 ser en et kamerakoordinatsystemet er markert med X,Y,Z samt et bildeplan med koordinater x,y. Ved hjelp av formlikhet og å tenke seg at en står et sted langs x-aksen, kan en enkelt vise at ligning 2.11 er sann. Det er alltid viktig å tenke over hvilke begrensinger en modell har. Denne tar ikke hensyn til linsefeil, objekt ute av fokus, samt at de fleste kamera nå til dags har distre bildebrikkekoordinater.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{2.10}$$



Figur 2.4: Nålehulls kameramodell. Legg merke til at en har definert et venstrehånds koordinatsystem, ettersom jeg ønsker at Z skal peke ut fra kamera.

2.4.1 Internmatrisen K

Som nevnt over tar ikke pin hole modellen hensyn til en del parametere, spesielt skal vi se nærmere på interne parametere. Internmatrisen **K** inneholder parametere som relaterer en forskyvning av bildesenter i forhold til kamerasenter. Ofte legger en ikke origo i senter av bildet, men oppe i venstre hjørne i f.eks MATLAB. Samt en skalering som sier noe om forholdet mellom piksler og meter, ergo brennvidde i antall pixler. Eksternmatrisen **[R t]** inneholder tre parametere, som ble snakket om i 2.3. Mens kameramatrisen **M** er produktet av **K[R t]**[1]. Sammenhengen mellom et homogentpunkt i 3D $P = [X,Y,Z,1]^T$ og $p = [x,y,1]^T$ er gitt av ligning 2.11. Bildebrikken skal i utgangspunktet være et kvadrat eller rektangel med hjørner som er 90°, som følge av fabrikkering som ikke er optimal, kan det i noen tilfeller være avvik som γ fanger opp.

$$s\mathbf{p} = \mathbf{K}[\mathbf{R}\ \mathbf{t}]\mathbf{P} = \mathbf{M}P \qquad K = \begin{bmatrix} \alpha & \gamma & x_0 \\ 0 & \beta & y_0 \\ 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad t = \begin{bmatrix} t_x \\ t_y \\ y_z \end{bmatrix} \tag{2.11}$$

2.4.2 Metode for kalibrering

I denne oppgaven benyttes MATLAB til sentrale beregninger, jeg har derfor valgt å se nærmere på hvordan intern matrisen **K** estimeres. I MATLAB blir det referert til artikkelen "A Flexible New Technique for Camera Calibration"[8] skrevet av Zhengyou Zhang. Som det nå vil gjennomgås med raske steg.

Ideen er å ta et mange bilder av et sjakkbrett med kameraet som ønskes kalibrert. Det er viktig at sjakkbrettet ligger i et plan. Det kan være greit å vite at det ikke er noen forskjell på om du flytter kameraet eller sjakkbrettet. Derfor kan en velge om en vil feste kameraet og flytte brettet eller omvendt.

Anta at planet er fast i $Z = 0$, dette er et valg jeg tar. Derfor har en muligheten til å la kameraet flytte seg vilkårlig rundt $Z = 0$, ut fra dette vil alle punkter på planet ha homogene koordinater $[X, Y, 0,1]^T$. Ut fra ligning 2.11 vil en nå ha følgene sammenheng (2.12) mellom **P** og **p**. Hvor **p** er homogene bildebrikke koordinater og **P** er homogene

[1] **K** er en 3×3 matrise, mens **[R t]** er en 4×4 matrise. Med litt misbruk av notasjon menes dermed $\mathbf{M} = \mathbf{K}\Lambda[\mathbf{R}\ \mathbf{t}]$ hvor

$$\Lambda = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

koordinater på planet $Z = 0$.

$$s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{r}_1 \ \mathbf{r}_2 \ \mathbf{r}_3\mathbf{t}] \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{r}_1 \ \mathbf{r}_2 \ \mathbf{t}] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{2.12}$$

Hvor $\mathbf{r}_i$ er kolonnene til $\mathbf{R}$ og $\mathbf{H} = \mathbf{K}[\mathbf{r}_1 \ \mathbf{r}_2 \ \mathbf{t}]$.

Deretter sitter en igjen med et forhold mellom punkter i verdenskoordinater og bildekoordinater. Som skulle gi mening ettersom alle punktene $\mathbf{P}$ nå er i samme plan($Z = 0$).

For alle kameraets posisioner har man en projeksjonsmatrixe(3×3) $\mathbf{H}_i$ hvor det nå antas at kameraets interne parametere er konstant, ergo samme kamera, focal length og oppløsnigen på bildet er det samme. Det eneste som endrer seg er kameraes fysiske posisjon

$$\mathbf{H}_i = \lambda_i \mathbf{K}[\mathbf{r}_1^i \ \mathbf{r}_2^i \ \mathbf{t}^i] \tag{2.13}$$

kan skrives som

$$[\mathbf{r}_1^i \ \mathbf{r}_2^i \ \mathbf{t}^i] = \frac{1}{\lambda_i}\mathbf{K}^{-1}[\mathbf{h}_1^i \ \mathbf{h}_2^i \ \mathbf{h}_3^i] \tag{2.14}$$

To av egenskapene til rotasjonsmatrisen er at den ikke skal ha skalering dermed $||r_j^i|| = 1$. Vektorene er basisvektror som står ortogonalt på hverandre, dermed er $(r_1^i)^T(r_2^i) = 0$. Ved å anvende dette på ligningen over får en:

$$(\mathbf{r}_1^i)^T(\mathbf{r}_1^i) = (\mathbf{r}_2^i)^T(\mathbf{r}_2^i) \tag{2.15}$$

$$(\mathbf{r}_1^i)^T(\mathbf{r}_2^i) = 0 \tag{2.16}$$

Fra ligning 2.15 kan det nå vises at :

$$(\mathbf{h}_1^i)^T(\mathbf{K}\mathbf{K}^T)^{-1} = (\mathbf{h}_2^i)^T(\mathbf{K}\mathbf{K}^T)^{-1}(\mathbf{h}_2^i) \tag{2.17}$$

Fra ligning 2.16 kan det nå vises at :

$$(\mathbf{h}_1^i)(\mathbf{K}\mathbf{K}^T)^{-1}(\mathbf{h}_2^i) = 0 \tag{2.18}$$

I ligning 2.17 og 2.18 kjenner en $\mathbf{h}_1^i$ og $\mathbf{h}_2^i$ mens $\mathbf{K}$ er ukjent, men en kjenner strukturen til $\mathbf{K}$.

$$(\mathbf{K}\mathbf{K}^T)^{-1} = \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & \frac{x_0\gamma - y_0\beta}{\alpha^2\beta} \\ -\frac{\gamma}{\alpha^2\gamma} & \frac{\gamma^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & -\frac{\gamma(x_0\gamma - y_0\beta)}{\alpha^2\beta^2} - \frac{x_0}{\beta^2} \\ \frac{x_0\gamma - y_0\beta}{\alpha^2\beta} & -\frac{\gamma(x_0\gamma - y_0\beta)}{\alpha^2\beta^2} - \frac{x_0}{\beta^2} & -\frac{\gamma(x_0\gamma - y_0\beta)}{\alpha^2\beta^2} + \frac{x_0}{\beta^2} + 1 \end{bmatrix} \tag{2.19}$$

Studerer en matrisen nærmere legger en merke til at matrisen er symmetrisk om diagonalen, en definerer derfor vektoren $\mathbf{k} = [K'_{11}, K'_{12}, K'_{22}, K'_{13}, K'_{23}, K'_{33}]^T$, hvor $\mathbf{K}' = (\mathbf{K}\mathbf{K}^T)^{-1}$. La så $\mathbf{h}_i = [\mathbf{h}_{i1}, \mathbf{h}_{i2}, \mathbf{h}_{3i}]^T$, hvor $\mathbf{h}_i$ er kolonnene til $\mathbf{H}$. Hvor nå følgene kan vises:

$$\mathbf{h}_i^T \mathbf{K}' \mathbf{h}_j = \mathbf{v}_{ij}^T \mathbf{k} \tag{2.20}$$

med

$$\mathbf{v}_{ij} = [\mathbf{h}_{i1}\mathbf{h}_{j1},\ \mathbf{h}_{i1}\mathbf{h}_{j2} + \mathbf{h}_{i2}\mathbf{h}_{j1},\ \mathbf{h}_{i2}\mathbf{h}_{j2},\ \mathbf{h}_{i3}\mathbf{h}_{j1} + \mathbf{h}_{i1}\mathbf{h}_{j3},\ \mathbf{h}_{i3}\mathbf{h}_{j2} + \mathbf{h}_{i2}\mathbf{h}_{j3},\ \mathbf{h}_{i3}\mathbf{h}_{j3}] \tag{2.21}$$

Restriksjonene fra ligning 2.15 og 2.16 gir nå

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = 0 \Rightarrow \mathbf{V}\mathbf{b} = 0 \tag{2.22}$$

$\mathbf{V}$ er matrise med dimension $2n \times 6$, hvor n er antall observerte bilder. Løsningen til $\mathbf{V}\mathbf{b} = 0$ er eigenvektorene til $\mathbf{V}^T\mathbf{V}$. Som følge kan en nå estimere $\mathbf{b}$. Resultatet er nå at en kan analysere hva som skjer med likningssettet for forskjellige verdier av n.

- Hvis $n \geq 3$, $\mathbf{b}$ har en unik løsning.
- Hvis $n = 2$,$\mathbf{b}$ unik løsning hvis en antar $\gamma = 0$
- Hvis $n = 1$, $\mathbf{b}$ kan kun estimere α og β gitt at x_0 og y_0 er kjent, samt $\gamma = 0$.

Det nå følger at hvis $\mathbf{b}$ er kjent så er følgene sammenheng $\mathbf{b}$ og elementene i $\mathbf{K}$ sant.

$$x_0 = (k_2k_4 - k_1k_5)/(k_1k_3 - k_2^2)$$

$$\lambda = k_6 - [k_4^2 + v_0(k_2k_4 - k_1k_5]/k_1$$

$$\alpha = \sqrt{\frac{\lambda}{k_1}}$$

$$\beta = \sqrt{\lambda \frac{k_1}{k_1k_3 - k_2^2}}$$

$$\gamma = -k_2\alpha^2\beta/\lambda$$

$$y_0 = \gamma x_0/\beta - k_4\alpha^2/\lambda$$

Med dette er matrisen $\mathbf{K}$ estimert, ut fra 2.12 kan en nå finne $[\mathbf{R}\ \mathbf{t}]$ som

$$\mathbf{r}_1 = \lambda \mathbf{K}^{-1}\mathbf{h}_1$$

$$\mathbf{r}_2 = \lambda \mathbf{K}^{-1}\mathbf{h}_2$$

$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$$

$$\mathbf{t} = \lambda \mathbf{A}^{-1}\mathbf{h}_3$$

Nå er interne- og ytreparametere kjent og kameraet anses som kalibrert. Jeg anser dette som hovedinnholdet i artikkelen til Zhengyou Zhang. For mer detaljer kan leseren lese artikkelen.[8]

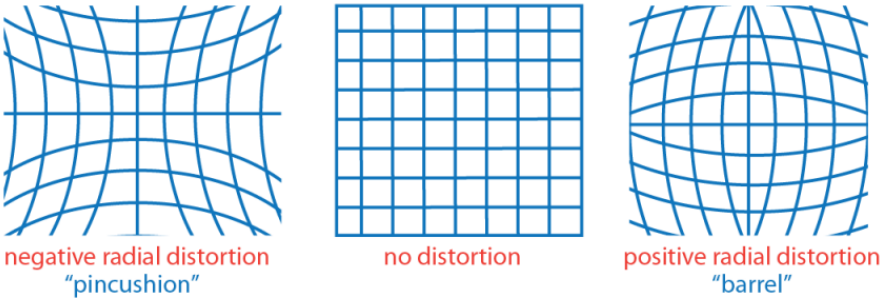
2.4.3 Linseforvregning

De fleste har badet og erfart at når en ser gjennom vannoverflaten er ting under vann ofte større en det de er virkeligheten. Dette skyldes brytningsindeksen mellom de forskjellige medium som fører til en bøyning av lys. Dette er en parallell til hva som skjer når en sender lys gjennom en linse. Billige linser vil ofte ha en stor grad av bøyning før lyset treffer bildebrikken. Avbøyning lyset får gjennom linsen er ofte en funksjon av avstanden til optiskesenter. Utviklerne av MATLAB har valgt følgene modell for linsen.[9, 10]

$$x_{dist} = x(1 + k_1 r^2 + k_2 r^4) \tag{2.23}$$

$$y_{dist} = y(1 + k_1 r^2 + k_2 r^4) \tag{2.24}$$

Hvor x,y er korrigerte pixelkoodinater,x_{dist},y_{dist} er forvrengte koordinater, k_1 & k_2 er koeffisienter som beskriver linseforvrengingen og $r^2 = x^2 + y^2$. For å estimere koeffisientene kan en lese "A Four-step Camera Calibration Procedure with Implicit Image Correction"[9].



Figur 2.5: Bilde viser to typer linseforvrengning nålepute og tønneforvrengingen. Hvor høyre bilde er forvrengingen som oppstår i kamera brukt i denne oppgaven. Bilde er hentet fra `http://se.mathworks.com/help/vision/ug/single-camera-calibrator-app.html`

2.5 Stereo Kamera

Kunnskapen skulle være til streking til at en kan utvide til flere kamera som tar bilde av samme scene. Håpet er at ved å introdusere enda et kamera at en skal kunne klare å komme frem til noen ligninger som beskriver dybden til punkter i scenen.

En starter med å ta utgangspunkt i figur 2.6 hvor en ser to kamera kalt R og L. Begge kamera har sitt eget interne koordinatsystem, men en skal så se at en kan finne en sammenheng mellom bildebrikke koordinatsystemet som gjør at en kan bergene dybden.

En observerer at koordinatsystemet til begge kameraene har samme orientering. Rotasjonsmatrisen blir derfor en identitetsmatrise $\mathbf{R} = \mathbf{I}$. Avstanden fra de to kameraene til P er det samme, $Z_l = Z_r = Z$, jeg antar at det er kun translasjon langs x-aksene, sammenhengen mellom x-aksene er derfor $X_r = X_l - t$, formlikhet gir dermed:

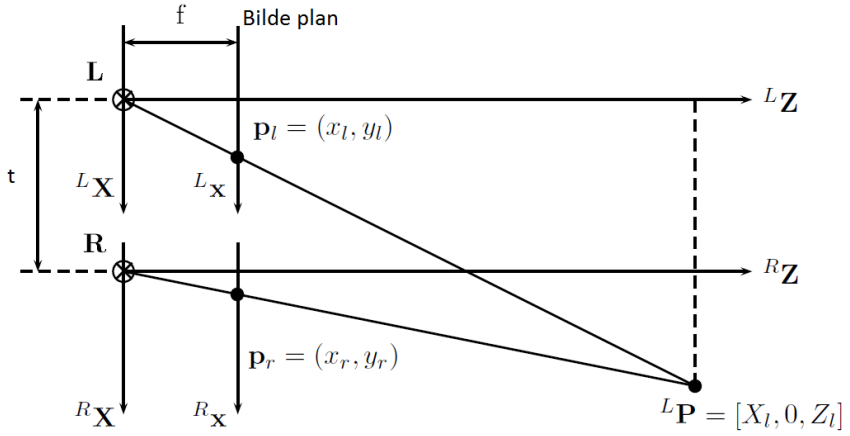
$$\frac{Z}{f} = \frac{X_l}{x_l} \quad \Rightarrow \quad X_l = \frac{Z}{f} x_l \tag{2.25}$$

$$\frac{Z}{f} = \frac{X_l - t}{x_r} \quad \Rightarrow \quad X_l = \frac{Z}{f} x_r + t \tag{2.26}$$

Fra 2.25 og 2.26

$$\frac{Z}{f}(x_l - x_r) = t \quad \Rightarrow \quad Z = f\frac{t}{d} \tag{2.27}$$

En velger dispariteten $d = x_l - x_r$. Oppsette som blir analysert nå er veldig forenkelt oppsett, i de fleste tilfeller vil $d = [d_x, d_y, d_z]$. Hvor en kan anta $d_z = 0$ om $d_x \gg d_z$ og $d_y \gg d_z$. Hvor antagelsen om $\mathbf{R} = \mathbf{I}$ vil som offtest være dårlig. Men hensikten var å vise at dybden kan nå beregnes i henhod til 2.27.



Figur 2.6: Venstre- og høyre kamera med avstand t.

2.5.1 Kalibrere stereorigg

En stereokamerarigg anses som kalibrert når en kjenner translasjonen og orienteringen som beskriver kamera 2 i forhold til kamera 1, samt internmatrisen til begge kameraene. En må derfor bestemme vektoren $\mathbf{t}$ og rotasjonsmatrisen $\mathbf{R}$. En mulighet er å finne et sett av felles punkter i bildene for å estimere parameterne $\mathbf{R}$ og $\mathbf{t}$.

Rotasjonsmatrisen er 3×3 derfor 9 parametere, translasjonsvektoren er 1×3, som gir 3 parametere. Dette gir totalt $9 + 3 = 12$ parametere. En må derfor ha minimum 12 felles punkter i kamera 1 og kamera 2 for å kunne løse ligningssettet. For mer detaljer kan en lese [9].

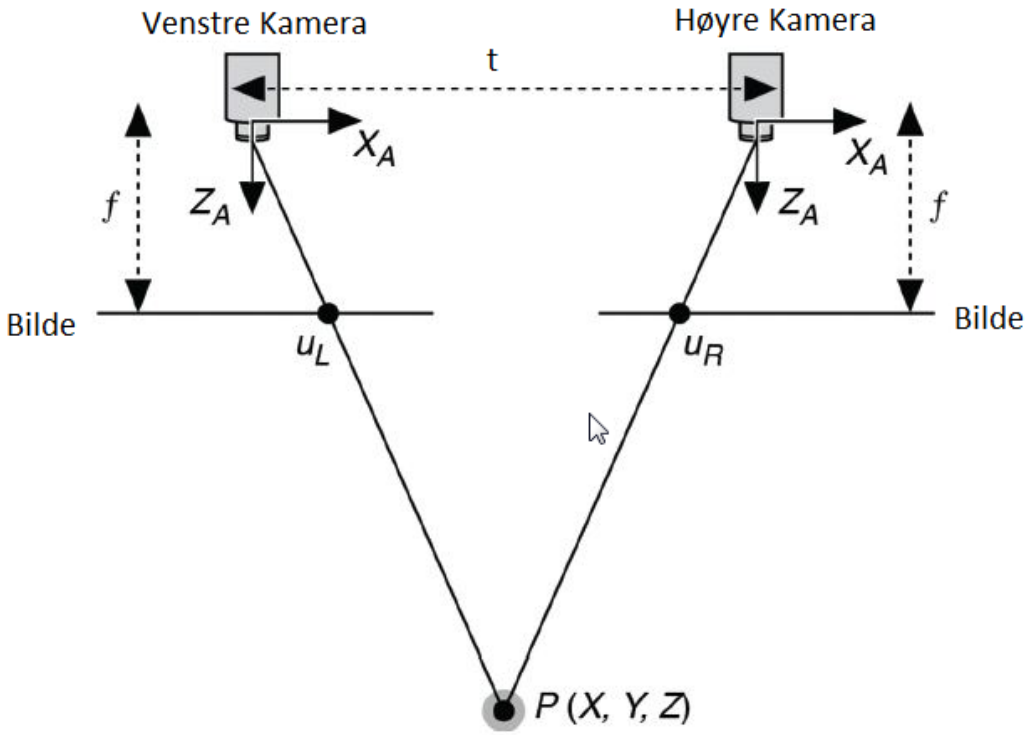
MATLAB har implementert en ferdig app for å kalibrere en sterokamerarigg, samt internmatrisen kaldt; "stereoCameraCalibrator". Hvordan appen brukes blir vist i implementerings delen av oppgaven.

Eksempel

I stereoriggen er t blitt valgt til $t = [20,\ 0,\ 0]^T mm$. (Det har blitt testet med forskjellige t, mens valgte t ser ut til å gi best resultat på den korte avstanden det jobbes med). Punktet P i scenen observet fra venstre kamera er gitt av $\mathbf{TR}$ med $t = [0,\ 0,\ 0]^T$, mens fra høyre kamera $t = [0.020,\ 0,\ 0]^T$. Orienteringen ser en ganske lett fra figur 2.7, at er $\mathbf{R} \approx \mathbf{I}$.

$\mathbf{TR}$ matisen for høyre kamera blir nå :

$$\mathbf{TR} = \begin{bmatrix} 1 & 0 & 0 & t_x|_{t_x=20mm} \\ 0 & 1 & 0 & t_y|_{t_y=0} \\ 0 & 0 & 1 & t_z|_{t_z=0} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0.020 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Figur 2.7: Venstre- og høyre kamera med oppsett t.

Valget av liten **t** er kanskje litt merkelig ettersom når **t** øker, vil som regel få større presisjon i rekonstruksjon senere. Men en har kommet frem til t fra følgene antagelser.

1. Roboten har en rekkevidde på ca $1m$. Objekter som er lenger borte er av ingen interesse. Som betyr at punkter med $Z > 1m$ er av ingen interesse.
2. f, focal length er kjent i antall pixel og er funnet til omtrent 2000pixel, $f \approx 2000pix$
3. Jeg velger d ved $1m$ til ca $40pixel$, $d \approx 40pix$

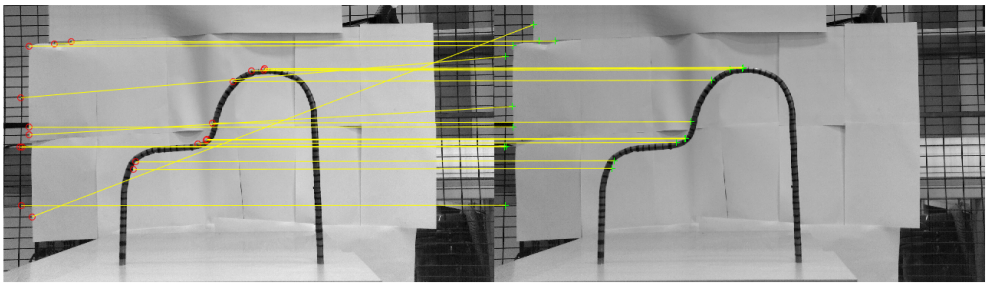
Ut fra dette kan en benytte seg av 2.27 løse for **t**, som blir optimal for dette oppsettet.

$$t = \frac{Zd}{f} \approx \frac{1m \cdot 40pix}{2000pix} = 0.02m = 2cm \tag{2.28}$$

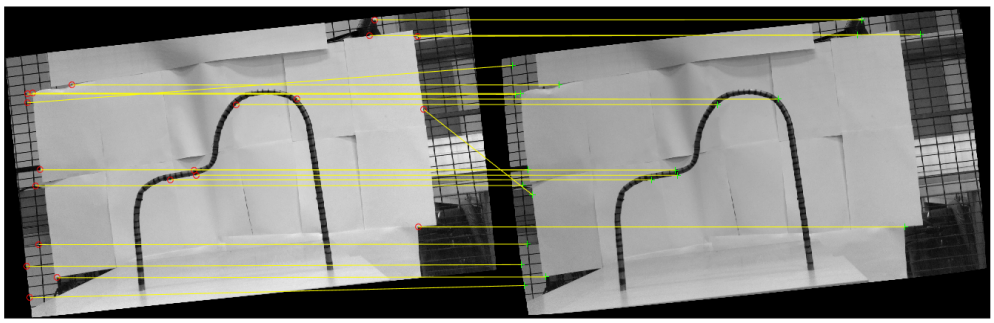
2.5.2 Bilde likeretting

For å kunne bergene disparitet er en avhengig av å finne punkter som matcher i begge bildeplan. Ved Brute force metoder er søket et to dimensjonalt. Beregningene blir derfor ofte betydelig enklere når en kan anta at bildene er likerettet. Det vil si at en transformerer de to originale bildene til et felles bildeplan. Det valgte bildeplanet har nå to friheter, posisjon og orientering. Posisjon er som oftest irrelevant fordi dette blir en skalar, som

enkelt kan løses med multiplikasjon av den inverse skalaren i bildekoordinatsystemet. Orienteringen er derimot litt mer vrien, men et fornuftig valg er som oftest en parallell til basislinjen mellom O_l og O_r. Som siste begrensing ønsker en å minimere endringen som blir påført bildene og derfor maksimere felles synsfelt. I [11, s433-446] blir Bouguet's aloritme forklart i detalj, som er implementert i MATLAB. [12]



Figur 2.8: I denne oppgaven er avstanden mellom kameraene nesten bare en ren translasjon å dermed vil de epipolare linjene være nesten horisontale. Men dette er ikke godt nok hvis en skal løse korrespondanse problemet effektivt er en avhengig av perfekt horisontale epipolare linjer.

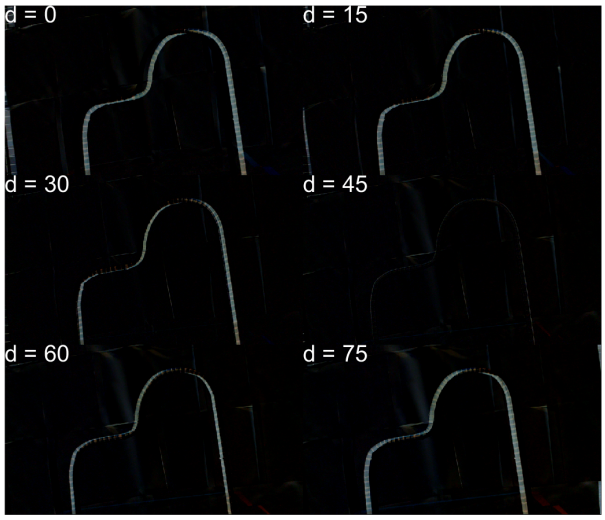


Figur 2.9: Studerer en de to bildene ovenfor vil en se at den epipolare linjene er mer horisontale i nederste bildet en øverste. Med unntak av to av linjene som er feilklassifisert.

2.5.3 Disparitet

Tidligere i 2.27 ble formelen $Z = f\frac{t}{d}$ utledet, hvor f og t er konstanter mens d kalt disparitet må estimeres. Estimering av disparitet er i faglitteraturen referert til som korrespondanse-problemet (eng.: Correspondence problem). Ved å la det ene bildet gli forbi det andre, å regne ut differansen mellom dem, blir deler av bildet tilnærmet null. I områder der bildet blir null har en nå funnet dispariteten. Et filmklipp som viser dette finne en i linken i fotnoten.[2]

[2] `http://www.ux.uis.no/~lovland/abb/disparityDemo.avi`

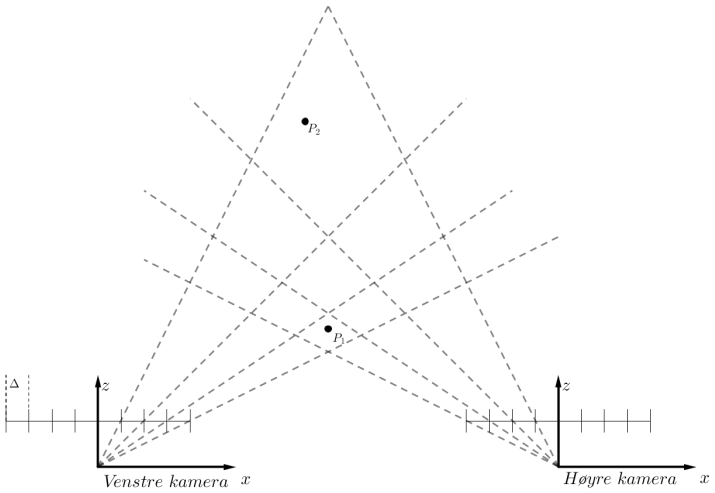


Figur 2.10: Bildeserien viser hvordan en kan finne disparitet til et objekt ved forskyvning av det ene bildet i forhold til det andre. Legg merke til at banen er nesten helt svart, som vil si lik null. Ved en forskyvning på 45 piksler finnes banen. Linken i fotnote 2 på side 20 anbefales for en bedre visuell fremstilling.

2.5.4 Rekonstruksjonsfeil

Jeg har laget en figur hvor jeg prøver å få frem problemet som oppstår når avstanden mellom kameraene og et punkt i scenen blir stor. I figur 2.11 er Δ størrelsen på en piksel. En ser tydelig at når avstanden øker blir området som et punkt ligger innenfor, mye større. En kan derfor med større nøyaktighet si hvor P_1 er i forhold til P_2.

Problemstilling til figur 2.11 blir dermed; gitt usikkerheten til lokalisering av tekstur i bildeplanet. Hva blir feilen i dybdeberegningene?
Feilen kan ses som en funksjon av ligning 2.27, hvor det da er åpenlyst at feilen i dypdeestimeringen Z' er avhengig av f, t Z, og usikkerheten knyttet til lokaliseringen x'_r & x'_l bildeplanet. Hvor jeg amtar at feilen kun ligger i d.



Figur 2.11: Pikselen som er assosiert med P_2 gir større usikkerhet en pikselen som er assosiert med P_1

En starter med å finne feilen som følge av estimering i disparitet, som er assosiert ved å lokalisere felles trekk mellom bildene, kaldt x_l og x_r. Fra tidligere har en $d = x_l - x_r$, som en partiell deriveres med hensyn på(mhp) d, x_l, x_r. Resultatet er d' som er endring i d som følge av ending i x_l og x_r. En antar x'_l og x'_r som uavhengige og null middelverdi.

$$d(d) = d(x_l) - d(x_r) \quad \Rightarrow \quad d' = x'_l - x'_r \tag{2.29}$$

$$\mu = \mathrm{E}\left[d'\right] = \mathrm{E}\left[x'_r\right] - \mathrm{E}\left[x'_l\right] = 0 \tag{2.30}$$

$$\begin{aligned} \mathrm{Var}\left[\Delta d\right] &= \mathrm{E}\left[(\Delta d - \mu)^2\right] = \mathrm{E}\left[(\Delta d)^2\right] \\ = \mathrm{E}\left[(\Delta x_l - \Delta x_r)^2\right] &= \mathrm{E}\left[\Delta x_l^2 - \underbrace{2\Delta x_l \Delta x_r}_{0} + \Delta x_r^2\right] \\ = \mathrm{E}\left[\Delta x_l^2\right] + \mathrm{E}\left[\Delta x_r^2\right] &\quad \Rightarrow \quad \sigma_d^2 = \sigma_l^2 + \sigma_r^2 \end{aligned} \tag{2.31}$$

Deriverer av $Z(d) = \frac{ft}{d}$, mhp d får en ending i Z som følge av d, $Z' = -f\frac{t}{d^2}d'$. Som nevnt tidligere antar en at $\mu_z = 0$, som følge blir $\mu_z = \mathrm{E}\left[\Delta Z\right] = 0$. Husk $\mathrm{E}[aX + b] = a\mathrm{E}[X] + b$.

$$\begin{aligned} \sigma_Z^2 = \mathrm{E}\left[(Z' - \mu_z)^2\right] = \mathrm{E}\left[(Z')^2\right] &= \left(\frac{ft}{d^2}\right)^2 \mathrm{E}\left[(-d')^2\right] \\ = \left(\frac{ft}{d^2}\right)^2 \sigma_d^2 \quad \Rightarrow \quad \sigma_Z &= \underbrace{\frac{ft}{d}}_{Z}\frac{1}{d}\sigma_d = Z\frac{\sigma_d}{d} \end{aligned} \tag{2.32}$$

Eksempeler

Gitt $d = 40pix$, focal length $f = 2000pix$ og $t = 2cm$, hva er Z?

$$Z = f\frac{t}{d} = 2000pix\frac{2cm}{40pix} = 100cm$$

Hva er usikkerheten relatert til dybden om, $\sigma_r = \sigma_l = 0.25pix$?

$$\sigma_Z = Z\frac{\sigma_d}{d} = Z\frac{\sqrt{\sigma_l^2 + \sigma_r^2}}{d} = 100cm\frac{\sqrt{0.25^2 + 0.25^2}pix}{40pix} \approx 0.9cm$$

Fra forskjellige feilmarginer er bildekoordinatene x_l og x_r 1% lavere enn sin sanne verdi, hva er usikkerheten i punktet $\mathbf{P} = [0.1m,0.2m,1m]^T$ når f og t er som i forrige eksempel.

Fra 2.5 har en $x_l = f\frac{X_l}{Z}$ og $x_r = f\frac{X_l - t}{Z}$

$$x_l = 2000\frac{0.1}{1}0.99 = 180pix \quad x_r = 2000\frac{0.1 - 0.02}{1}0.99 = 158pix$$

$$Z_{error} = \frac{2000 \cdot 0.02}{180 - 158} = 1.82cm \quad \Delta Z = Z_{erorr} - Z = 1.8 - 1 = 0.82cm$$

Fra 2.4 har en sammenhengen mellom bildekoordinater.

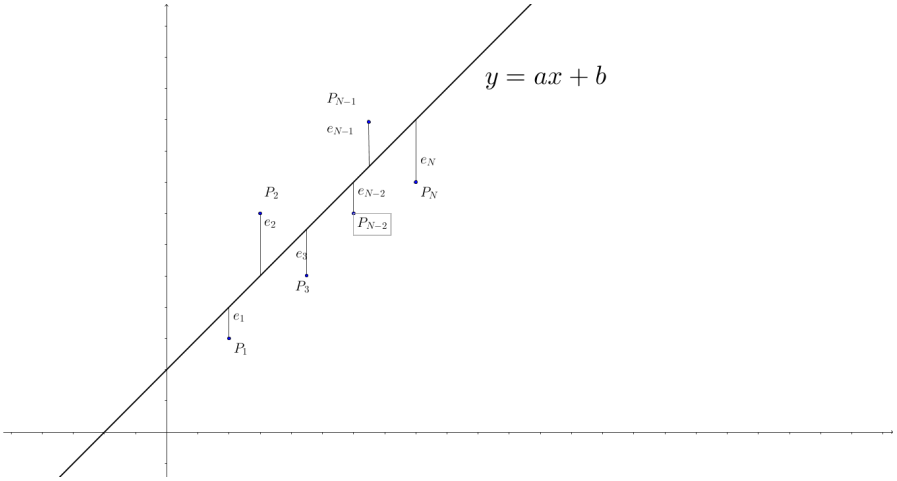
$$\Delta\begin{bmatrix}x\\y\end{bmatrix} = \frac{1}{0.99}\cdot\frac{1}{2000}\begin{bmatrix}0.1\\0.2\end{bmatrix} - \frac{1}{2000}\begin{bmatrix}0.1\\0.2\end{bmatrix} = \frac{1}{2000}\begin{bmatrix}2\\4\end{bmatrix}[pix] \Rightarrow \begin{bmatrix}X\\Y\end{bmatrix} = \frac{1}{2000}\begin{bmatrix}2\\4\end{bmatrix} = \begin{bmatrix}0.1\\0.2\end{bmatrix}cm$$

Koordinatakse	Avvik[cm]
X_{error}	$0.1[cm]$
Y_{error}	$0.2[cm]$
Z_{error}	$0.82[cm]$

Figur 2.12: Dette viser at ved relativt liten feilmargin i x_l og x_r er det ca. 4 ganger større feilmargin i dybdeestimeringen en i XY-planet.

2.6 Minste kvadraters metode, for en linje

Problemstillingen er som følger; gitt et sett med punkter, som man ønsker å modellere som $y = ax + b$. Hvordan finner en a og b som gjør at den totale kvadrerte feilen SE (end:.squrred error) blir minst ?



Figur 2.13: Punktene P_1 til P_N og tilhørende avvik.

$$SE = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} (y_i - (ax_i + b))^2 = (y_1 + (ax_1 + b))^2 + ... + (y_N + (ax_N + b))^2 \quad (2.33)$$

e_i^2 er den kvadrerte feilen, ut fra figuren 2.13 ser at $e_i^2 = (y_i - (ax_i + b))^2$. SE er det totale kvadrerte avviket, som da er summen av alle kvadrerte e_i. Utvider en alle parentesene og samler felles faktorer kan en vise at SE kan skrives som:

$$\begin{aligned} SE = &\left(y_1^2 + ... + y_N^2\right) - 2a\left(x_i y_i + ... + x_N y_N\right) - 2b(y_i + ... + y_N) \\ &+ a^2(x_i^2 + ... + x_N^2) + 2ab(x_i + ... + x_N) + Nb^2 \end{aligned} \quad (2.34)$$

Her kommer et triks. La gjennomsnittet $\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \Rightarrow \overline{x}N = \sum_{i=1}^{N} x_i$, og tilsvarende for x^2, x, y, y^2, xy så kan ligningen nå skrives som:

$$SE = N\overline{y^2} - 2aN\overline{xy} - 2bN\overline{y} + a^2N\overline{x^2} + 2abN\overline{x} + Nb^2 \quad (2.35)$$

For å finne minste SE er man nå kun avhengig av å finne a og b som minimerer SE. Fra matematikken kjenner man igjen SE som en (elliptisk) paraboloide. Fortegnene viser at ved partiell derivasjon vil man finne et minimum. Minimum finnes nå som $\frac{\partial SE}{\partial a} = \frac{\partial SE}{\partial b} = 0$.

$$\frac{\partial SE}{\partial a} = -2N\overline{xy} + 2a\overline{x^2}2N + 2b\overline{x}N = 0 \tag{2.36}$$

$$\frac{\partial SE}{\partial b} = -2N\overline{y} + 2aN\overline{x} + 2Nb = 0 \tag{2.37}$$

Faktoren $2N$ kan fjernes ved å multiplisere begge ligninger med $\frac{1}{2N}$,$N > 0$. Ligningen kan skrives på matrise vektorform $\mathbf{Ax} = \mathbf{b}$, med løsning $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$.

$$\begin{bmatrix} \overline{x^2} & \overline{x} \\ \overline{x} & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \overline{xy} \\ \overline{y} \end{bmatrix} \Rightarrow \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \frac{\overline{xy} - \overline{x}\,\overline{y}}{(\overline{x})^2 - \overline{x^2}} \\ \overline{y} - a\overline{x} \end{bmatrix} \tag{2.38}$$

Legg merke til at $(\overline{x})^2 \neq \overline{x^2}$. Løsningen en har kommet frem til gir a og b som minimerer $e_1^2 + ... + e_N^2$.

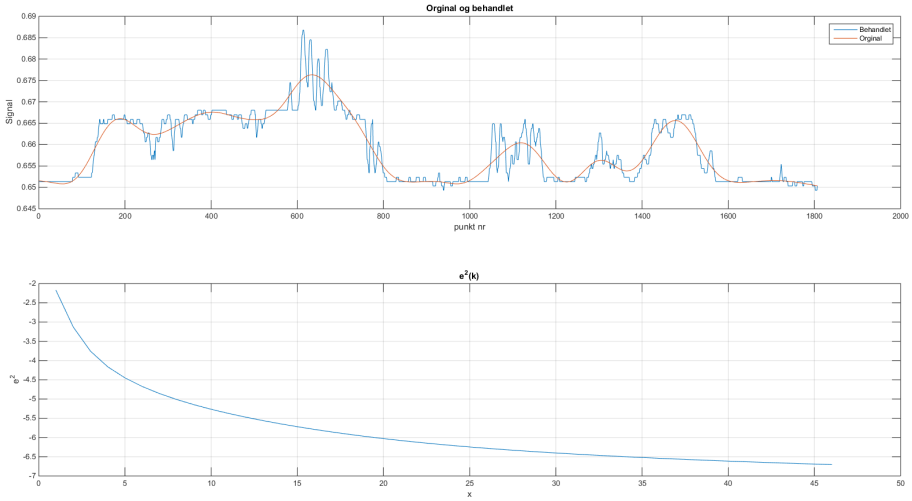
2.6.1 Minste kvadrat filtrering

Tanken med kvadrat filtreringen[KF] er at linjer forblir linjer mens krappe avbøyninger blir fjernet å prøvd tilpasset en av funksjonen $y = ax + b$. En tar dermed utgangspunkt i hvordan a og b estimers fra forrige avsnitt. Det plukkes så ut et gitt antall punkter fra datasettet. For datasettet som har blitt plukket ut blir parameterne a og b estimert. Man har dermed laget en modell av formen $ax + b$ som passer for underdatasettet. Gjennomsnittet av modellen og underdatasettet blir regnet ut å lagt tilbake i det originale datasettet. Kvadrat filtreringen er utviklet av forfatter å ser ut til å løse tiltenkte oppgave bra.

Prosessen som er beskrevet over kjøres helt til virkningen er minimal. Dette er gjort ved at en tar differansen mellom nyeste- og forrige glatting, kvadrerer å summerer. Hvis denne verdien er tilnærmet null anses filtrert for å ha konvergert å prosessen avsluttes.

Algorithm 1 Punktglatting

procedure POLY1LMS($data$,N,tr)
  $N \leftarrow (N-1)/2$
  $k \leftarrow 0$
  while *true* **do**
    $old \leftarrow data$
    for $i = N+1 : length(data) - (N+1)$ **do**
      $Y \leftarrow data(i-N : i+N)$ ▷ Plukk ut punkter fra $i-N$ til $i+N$
      $x \leftarrow linspace(data(i-N),data(i+N),2N+1)$
      $\Phi \leftarrow [x',ones(N*2+1,1)]$ ▷ $2 \times N$ matrise
      $\theta \leftarrow \Phi^{\dagger}Y$ ▷ $\Phi^{\dagger} = (\Phi^T\Phi)^{-1}\Phi^T$[13]
      $data(i-N : i+N) \leftarrow (theta. * [x,ones(1,2 \cdot N+1)] + data(i-N : i+N))/2$
    ▷ Utsnittet av data variabelen blir tildelt gjennomsnittet av gammel- og ny verdi.
      $change \leftarrow (old - data)^2$
      if $change \leq tr$ **then**
        Break
  return *data* ▷ Ferdig behandlet signal.

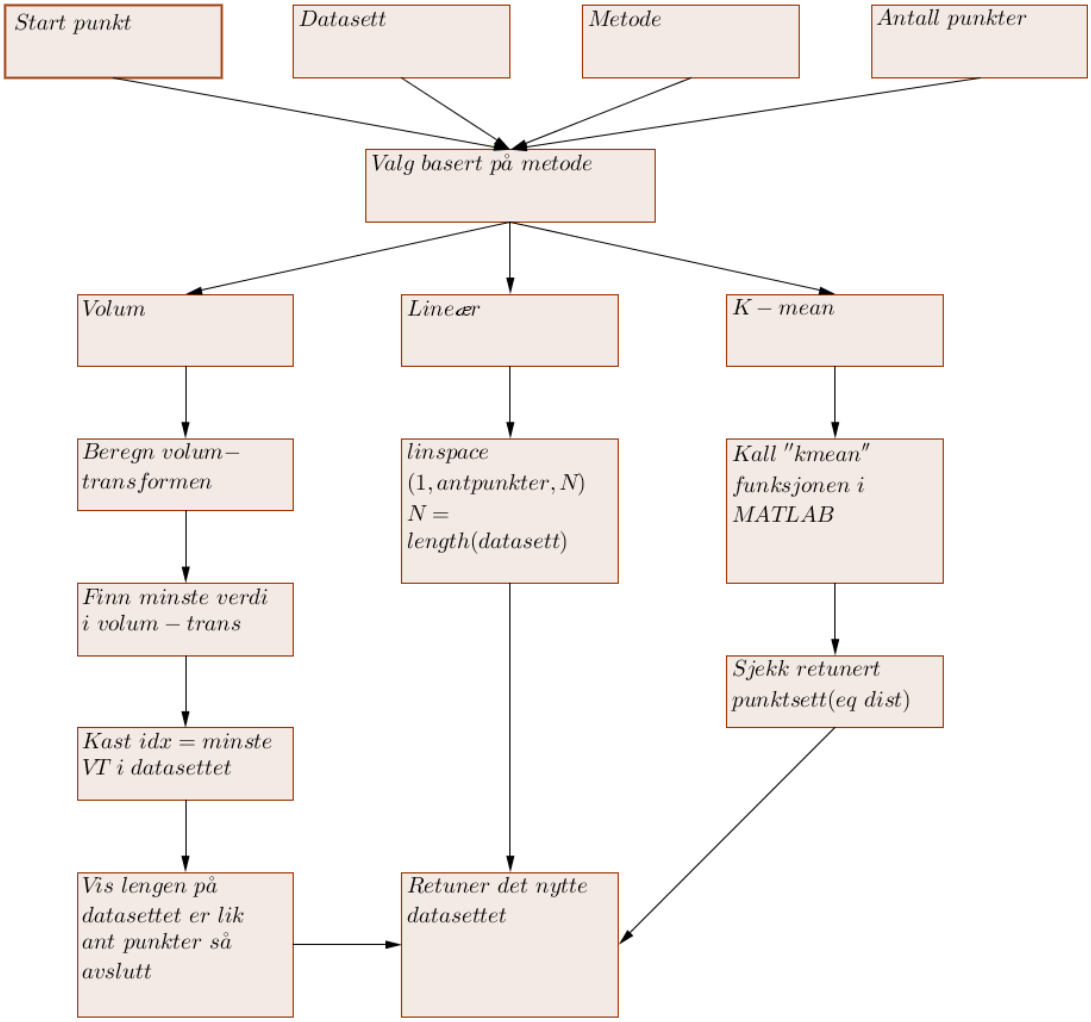


Figur 2.14: Første plot viser originalt- og behandlet signal, hvor en ser at områder hvor det er liten ending eller linjer beholder egenskapene sine. Mens områder med mye ending blir prøvd tilnærmet linje. Andre plot viser at filteret konvergerer etter ca. 45 iterasjoner

2.6.2 Flytskjema for punktRed funksjon

Flytskjemaet viser hvordan de forskjellige modulene til i `punktRed` henger sammen. Som nevnt blir funksjonen kalt med 3-4 argumenter hvorav hvilken metode som ønskes brukt

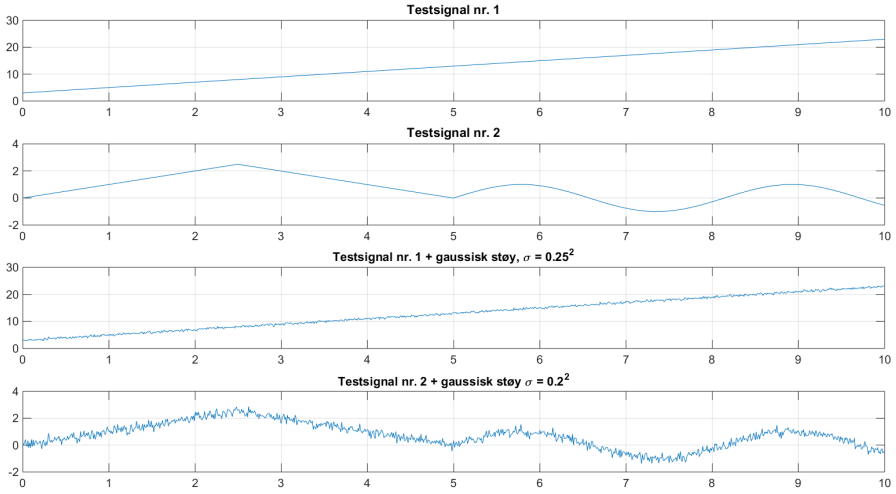
blir funnet i andre blokk fra toppen. Herfra blir de punktskyen sendt til et av de 3 grenene basert på valget som er gjort i forrige blokk. På slutten av grenene blir redusertpunktsky returnert.



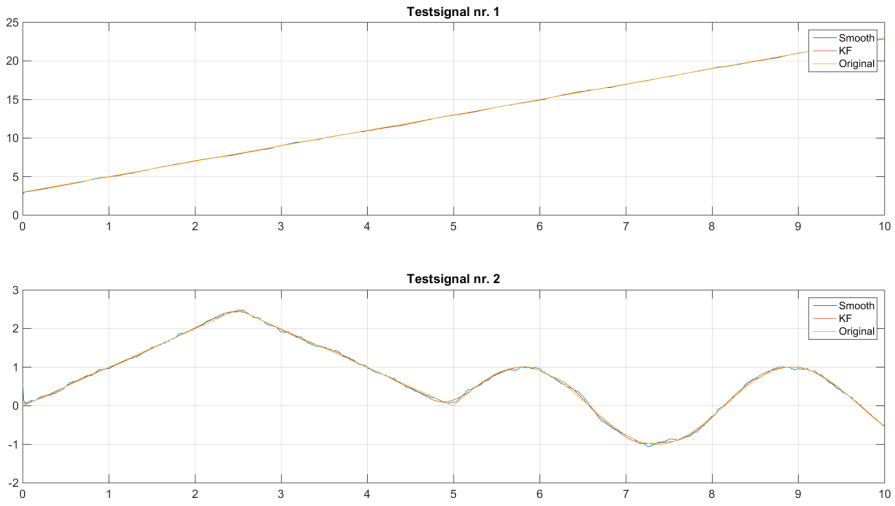
Figur 2.15: Flytskjema til punktRed-funksjonen

2.6.3 Eksperimentell sammenligning av KF og smoothing-filter

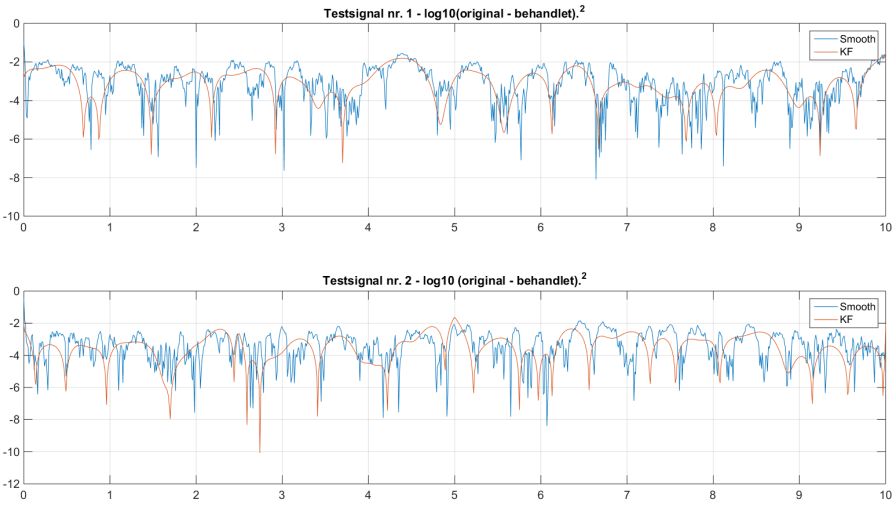
Jeg lager to testsignal som blir filteret og hvor en etterpå ser på hvor godt dem treffer det opprinnelige signalet. Testsignal nummer en er på formen ax+b+ gaussisk støy. Testsignal nummer to er en trekant med en sinus kurve lagt til på slutten. Testsignelene er plottet i figuren under. Smoothing filtere er satt opp som et '' moving average" filter med lengde 32.



Figur 2.16: Plottene viser de forskjellige testsignalene



Figur 2.17: Signalene som er behandlet smootingfilter og KF. Studerer man grafen nøye ser en tendenser til at smooth vandrer rundt det originale signalet.



Figur 2.18: Plottene viser tier-logaritmen til original- minus behandletsignal i andre potens. KF filtrere ser ut til å være mer stabilt og en hel del glattere

2.7 Punktreduksjon

I dette oppsettet får en omtrent 2000 punkter som beskriver banen, hvor et piksel tilsvarer et punkt i rommet. Når banen er omtrent 50 cm lang vil dette medføre $\frac{2000punkter}{50cm} = 40\frac{punkter}{cm}$. Som er "overkill", fordi hvis en skal oppnå en imponerende hastighet må robotkontrolleren klare å planlegge banen i forkant av sin nåværende posisjon.

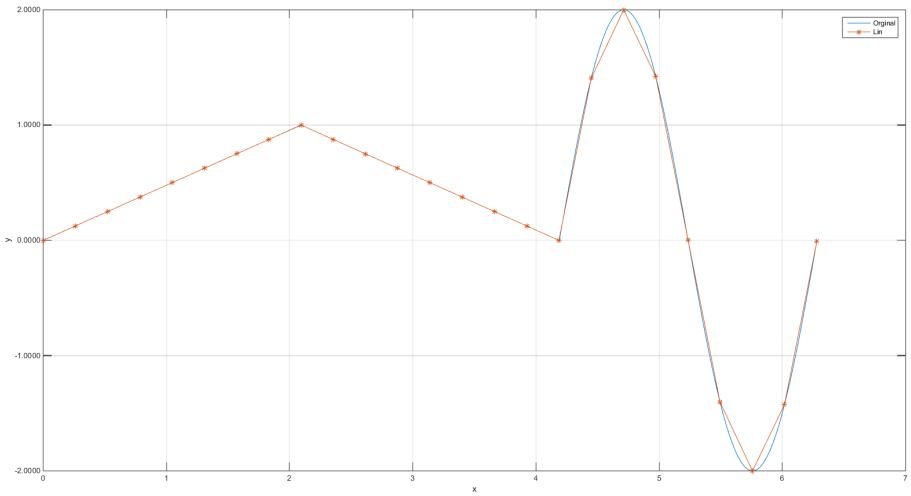
Når en har et ønske om at roboten skal følge banen nøyaktig, så en må inngå et kompromiss mellom antall punkter og nøyaktighet. Fra teorien har roboten $1cm$ å jobbe med kan jeg gå fra 2000 punkter til 50 punkter, samt fra teorien i forrige avsnitt har man funnet omtrent 1cm feilmargin. Ved å velge 100 punkter har en ca. fire ganger (2.5.4) så god presisjon i xy-koordinatene som i z-koordinatene.

2.7.1 Lineærreduksjon

Når jeg snakker om lineær punktreduksjon legger jeg til grunn at jeg har matrisen **wP** som inneholder N elementer. Som skal reduseres til k elementer som er lineært fordelt. Slik at $\mathbf{wP}_{ny} = \mathbf{wP}(u,v), \quad v = \{1,2,3\}, \quad u = \{1...step...N\}$, hvor step er steget mellom hvert punkt og N er lengden av vektoren.

Eksempel

I figuren under har en redusert signalet fra 1800 punkter til 25 ved og kun beholde vært 72 punkt.



Figur 2.19: Figuren viser opprinnelig signal og redusert signal fra 1800 punkter til 25 punker.

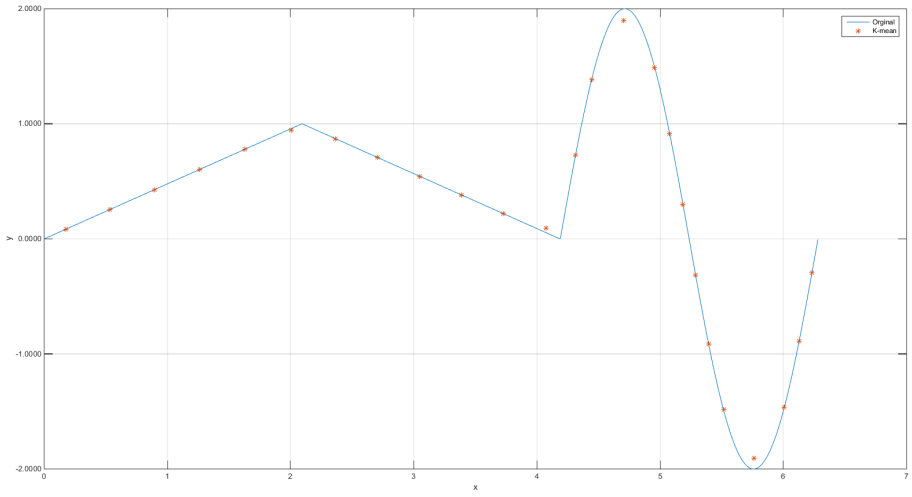
2.7.2 K-means

Her prøver en å finne et sett med punkter som gjør at avstanden mellom det nye settet og det originale punktsettet blir minst mulig. Punktene som blir stående igjen skal gjøre at distansen fra k utvalgte punkter til resten av punktene blir minst mulig. Distansen er målt som Euklidsk avstand (eng:. Euclidean distance). Ideen er at steder hvor en har mange punkter, har disse en tendens til å bli plukket ut.

En starter som regel med en tilfeldig gjetting av hvor punktene c_1,c_2,..,c_k ligger, også regner en ut distansen fra alle punktene som er nærmest c_1, så c_2 opp til c_k. Punktene som er nærmest sin respektive c_i blir klassifisert som en del av c_i. Så flytter en c_i til midtpunktet av dataclustren[3]. For så å state på en ny runde med distanseberegning og klassifisering. Dette gjør en helt til c_i ikke flytter seg lenger. [2, s202][14]

Eksempel

Som i eksempelet med lineærreduksjon, prøver en og gå fra 1800punkter til 25. I figuren under er dette gjort med k-means algoritmen.



Figur 2.20: Figuren viser opprinnelig signal og redusert signalet fra 1800 punkter til 25 punkter med K-mean clustering.

[3] Tenkt senter av gravitasjon

2.7.3 Krumning

I kalkulus er krumning et vel definert begrep som et mål på hvor krapp svingene langs en kurve C er. Krumning er en egenskap til kurven, som dermed er uavhengig av hastigheten til eventuelle partikler som skulle følle banen.

En antar at kurven C er gitt ved en buelengdeparametrisering $\mathbf{r} = \mathbf{r}(t)$. Enhetsvektoren $\mathbf{T}$ viser retningen til C i ethvert punkt. Dermed vil $\mathbf{T}$ endre seg i forhold til t jo mer kraftigere svingen er. En definerer krumning som endrinsraten til $\mathbf{T}$(t).

I Kalkulus[17] er krumningen til den glatte kurven $\mathbf{r}$=$\mathbf{r}$(t) i punktet i punktet $\mathbf{r}$(t) er definert ved:

$$\kappa = \left| \frac{d\mathbf{T}}{dt} \right| \tag{2.39}$$

Eksempel fra [17]

Finn krumningen til en sirkel i origo med radius R.

En sirkel kan med radius R kan parametriseres ved

$$r(t) = [R\cos(t), R\sin(t)] \quad for \quad 0 \leq t \leq 2\pi$$

Som følge får en $v(t) = r'(t) = [-R\sin(t), R\cos(t)]$ og $|v(t)| = R$, dermed er $\mathbf{T}(t) = [-\sin(t), \cos(t)]$.

Av kjerneregelen får man:

$$\frac{d\mathbf{T}}{dt} = [-\sin(t), \cos(t)] \cdot \frac{1}{R}$$

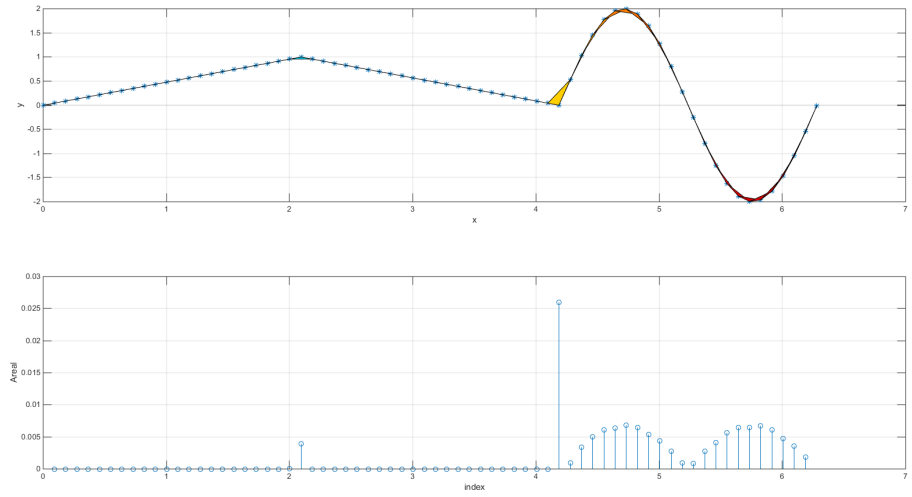
Fra 2.39 finner en κ som:

$$\kappa = \left| \sqrt{\cos^2(t) + \sin^2(t)} \cdot \frac{1}{R} \right| = \frac{1}{R}$$

Resultatet skulle ikke være spesielt overraskende, alle punkter på sirkelen har samme krumning. En ser også at om R øker blir κ mindre.

2.7.4 Punkter og areal

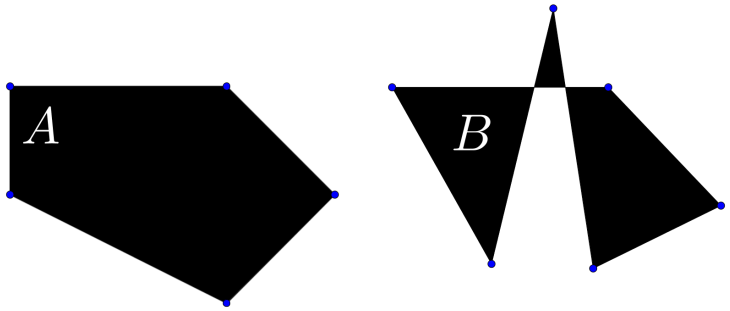
Med inspirasjon fra forrige delkapittelet om krumning prøver en og få til en diskreditert utgave, hvor en ser på arealet som er gitt av 3 punkter på kurven. Ideen med å bruke arealet er at om en har 3 punkter på en linje vil arealet fra disse punktene bli null. Litt skjevhet gir litt mer areal, mer skjevhet gir enda litt mer areal osv. Om en studerer figur 2.21 ser en tydelig at deler av grafen er mer interessant en noen deler. Start og slutt, er veldig viktige, toppen av trekanten, samt noen punkter av sinusen. Studerer man sinusfunksjonen litt nærmere kan en se at funksjonen er litt mer lineær fra starten og blir mer ulineært mot toppen. Dette er ting jeg ønsker å fange opp å legge til grunne for å kaste eller beholde et punkt.



Figur 2.21: Den øverste grafen viser hvordan arealet mellom 3 punkter utvikles. Grafen under viser arealet som er fremhevet i farger. Fargen har ikke noe med verdien å gjøre, bare estetisk.

Skolisseformelen

Med skolisseformelen kan en finne arealet til et enkelt polygon hvis en vet alle hjørnekoordinatene som utgjør et polygon.[16] Et enkelt polygon er definert ved at ingen av kantene krysser en annen kant i samme polygon. I figur 2.22 ser en to polygon A og B, hvor A er et enkelt polygon fordi det ikke krysser sine egne kanter.
Arealet til polygon A, vil være rett frem å kalkulere ut fra Skolisseformelen, men polygon B har kanter som krysser internt, som gjør at den minste trekanten vil gi et negativt bidrag til det totalte arealet og en vil derfor ikke ende opp med en korrekt verdi. For å unngå dette velges det kun 3 punkter som skal utgjøre arealet som blir undersøkt. Skolisseformelen er et spesialtilfelle av Green's therom. [15]



Figur 2.22: A er et simpelt polygon, B er ikke.

Arealet til et polygon vil da være gitt av formelen 2.40, hvor x_i er x-koordinaten til punktet P_i og y_i er y-koordinaten i samme punkt.

$$A = \frac{1}{2}\left|\sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n\right| \tag{2.40}$$

Som nevnt tidligere kan en få problemer om punktene ligger uheldig, derfor velges 3 punkter ($n = 3$). Dette vil føre til at linjene mellom punktene ikke krysser.

$$A_s = \frac{1}{2}\left|x_1 y_2 + x_2 y_3 + x_3 y_1 - x_2 y_1 - x_3 y_2 - x_1 y_3\right| \tag{2.41}$$

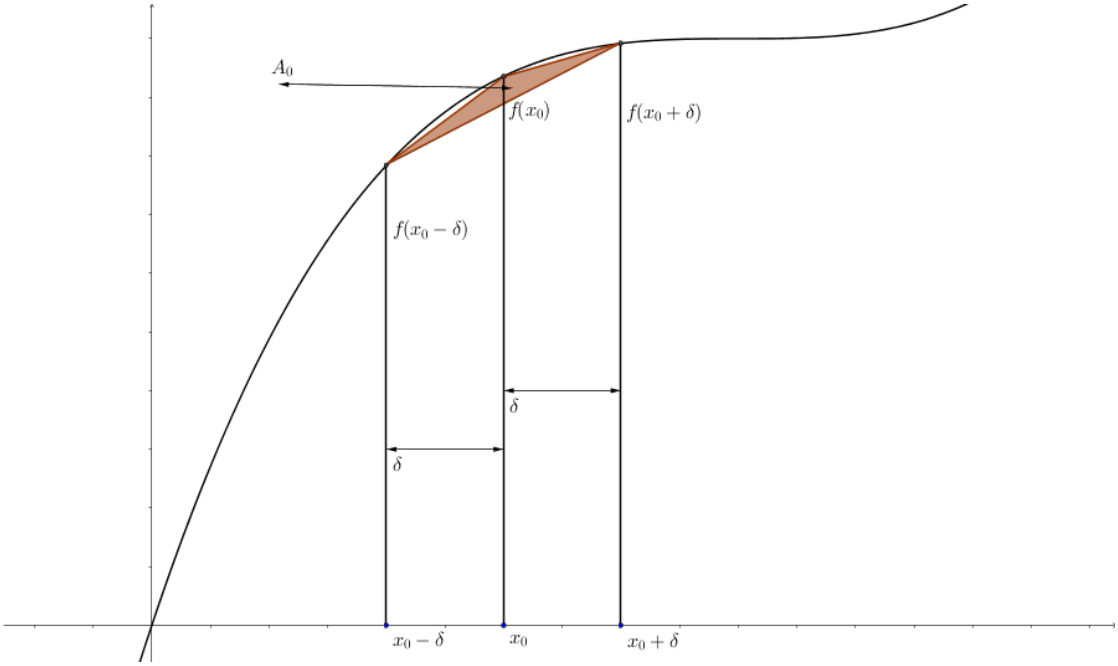
En må dermed undersøke om hva skjer om punktene ligger på linje. Intuitivt skulle dette medføre at arealet blir null, men jeg undersøker for å sjekke at 2.41 ikke bryter sammen. Når punktene ligger på linje er dette det samme som at $P_1 = c_2 P_2 = c_3 P_3$, hvor $c_{2,3}$ er reelle positive konstanter.

$$A = \frac{1}{2}\left|x_1 y_1 c_2 + x_1 y_1 c_2 c_3 + x_1 y_1 c_3 - x_2 y_1 c_2 - x_1 y_1 c_2 c_3 - x_1 y_1 c_3\right| = 0$$

Det er atså ikke noe problem at punktene er på linje, formelen gir rett resultat.

2.7.5 Arealtransformen til f(x)

Jeg velger ut 3 punkter på funksjonen $f(x)$. Punktet i midten blir kaldt $f(x_0)$, punktet til venstre $f(x_0 - \delta)$ og punktet til høyre $f(x_0 + \delta)$. Som vist i figuren 2.23. Arealet som de 3 punktene gir, kaller jeg A_0.



Figur 2.23: $A(x_0)$ som en funksjon av $f(x_0 \pm \delta)$

Jeg lar følgene punkter i 2.41 bli byttet ut med punkter i funksjonen $f(x)$ i figuren over.

$$x_0 - \delta = x_1, \quad x_0 = x_2, \quad x_0 + \delta x = x_3 \tag{2.42}$$

$$f(x_0 - \delta) = y_1, \quad f(x_0) = y_2, \quad f(x_0 + \delta) = y_3 \tag{2.43}$$

$$\tag{2.44}$$

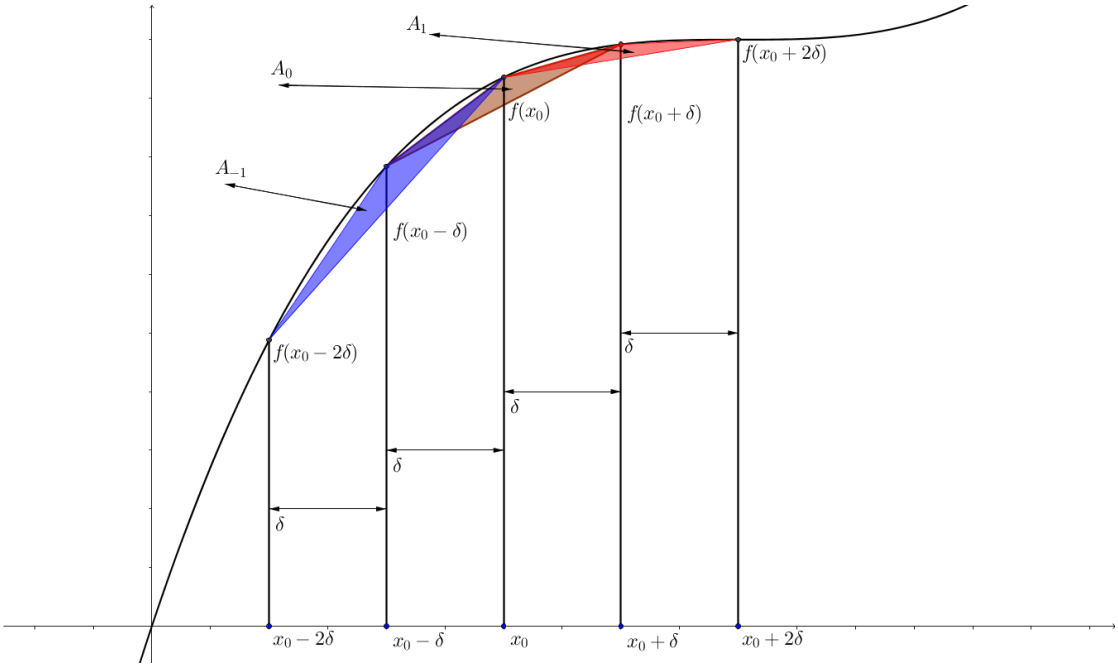
Sett inn ligningene fra 2.42 inn i 2.41

$$\begin{aligned} A(f(x_0)) = \frac{1}{2}|&(x_0 - \delta) \cdot f(x_0) + x_0 \cdot f(x_0 + \delta) + (x_0 + \delta) \cdot f(x_0 - \delta) \\ &-x_0 \cdot f(x_0 - \delta) - (x_0 + \delta) \cdot f(x_0) - (x_0 - \delta) \cdot f(x_0 + \delta)| \\ &= \frac{1}{2}| - 2f(x_0) + f(x_0 - \delta) + f(x_0 + \delta)\ |\delta \end{aligned} \tag{2.45}$$

En har dermed kommet frem til det jeg kaller arealtransformasjonen A.

$$A\{f(x)\} = \frac{1}{2}|-2f(x_0) + f(x_0 - \delta) + f(x_0 + \delta)|\delta \tag{2.46}$$

Man har nå en formel som beskriver arealene til polygonene som er begrenset av funksjonen $f(x)$. Som en ser er denne vil arealet variere hvor en er hen på kurven.



Figur 2.24: Arealene $A_{-1 \to 1}$ er gitt av 2.46. Studerer en figuren litt nærmere legges det merke til at A_{-1} er en del større en A_0. Dette har sitt opphav i at funksjonen blir mer lineær om x_0.

Jeg undersøker om transformen virkelig har de egenskapene som er ønsket, for $f(x) = ax + b$ skal transformen gi null.

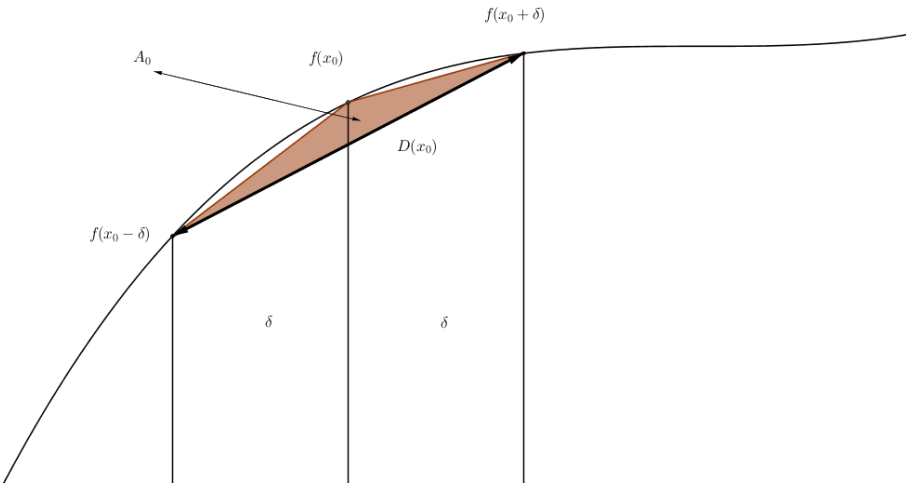
$$\begin{aligned} A\{f(x) = ax + b\} =& \frac{1}{2}|-2(ax+b) + a(x-\delta) + b + a(x+\delta) + b| \\ =& \frac{1}{2}|-2ax - 2b + ax - a\delta + b + ax + a\delta + b| \\ =& \frac{1}{2}|0|\delta = 0 \end{aligned} \tag{2.47}$$

At arealtransform gir null som resultat skulle ikke komme som noen overraskelse ettersom punktene er på en linje og vil gi et areal som er null. Om en ser på en ulineær funksjon som $f(x) = ax^3$ kommer det frem noe som er interessant.

$$\begin{aligned} A\{f(x) = ax^3\} =& \frac{1}{2}| - 2ax^3 + a(x-\delta)^3 + a(x+\delta)^3| \\ =& \frac{1}{2}| - 2ax^3 + a(6\delta^2 + 2x^2)| \ , \\ =& 3|ax\delta^2|\delta = 3\delta^3|ax| \end{aligned} \tag{2.48}$$

En ser at ved $x = 0$ kommer arealet til å bli null uavhengig av δ. Man vil kunne si at punktene er på rette linjer og derfor begrense seg til to punkter som beskriver hvordan en kan få en fullkommen bevegelse fra punkt $((x-\delta),f((x-\delta)))$ til $((x+\delta),f((x+\delta)))$ gitt at $x \approx 0$

Et problem med transformasjonen er hvis punktene er langt fra hverandre kan et lite avvik fra linjen de ligger på gi et stor utslag i arealet. Derfor velger jeg å dele på den totale avstanden D mellom endepunktene. Tanken er at, hvis punktene er nærme vil D bli liten, om de er langt fra hverandre blir D stor. Dette vil bli en form for normalisering av arealet. Arealtransform blir nå 2.49. Fra figuren under er det åpenlyst at D er gitt som $D(x_0) = \sqrt{(2\delta)^2 + (f(x_0-\delta) - f(x_0+\delta))^2}$.[4]



Figur 2.25: D som funksjon av δ og $f(x_0 \pm \delta)$

En deler ligning 2.46 med D å får uttrykket under.

$$A(f(x_0)) = \frac{1}{2D(x_0)}| - 2f(x_0) + f(x_0-\delta) + f(x_0+\delta)|\delta \tag{2.49}$$

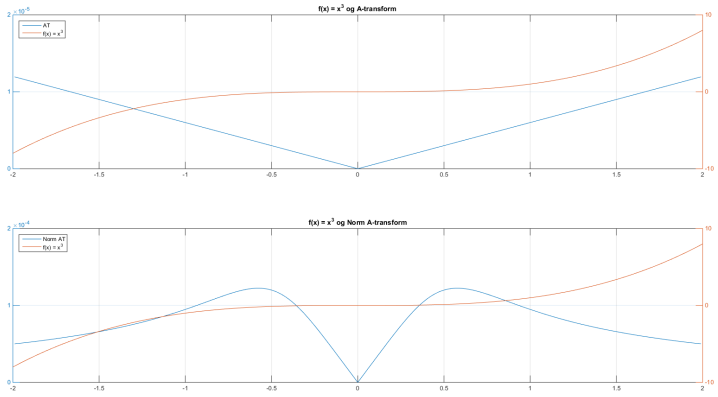
Hvor

$$D(x_0) = \sqrt{(2\delta)^2 + (f(x_0-\delta) - f(x_0+\delta))^2}$$

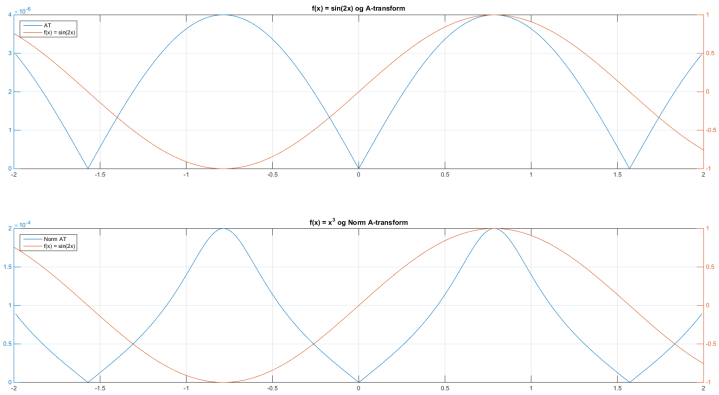
[4]Pytagoras' læresetning

2.7.6 Eksperimentell sammenligning av A-transform og normalisert A-transform

Hensikten med normaliseringen er at hvis punktene er langt fra hverandre kan de fremdeles gi et stort areal om det er liten skjevhet mellom punktene. Dette prøves å fange opp av normaliseringen. Under kommer det to eksempel av A-transformen og normalisert A-transformen hvor $f = x^3$ og $f = \sin(2x)$. Den øverste viser sammenhengen mellom AT og funksjonen, hvor en ser at om $x = 0$ blir a-transformen liten som tyder på at f blir mer lineær i dette området. I 2.27 ser en samme tendenser rundt $x = \frac{\pi}{2} \cdot N,\ N \in \mathbb{Z}$. Det viktigste forholdet er at AT og norm AT er enige om topp- og bunnpunkt i transformen, ettersom det er bunnpunktet som blir plukket ut når AT blir brukt til punktreduksjon.



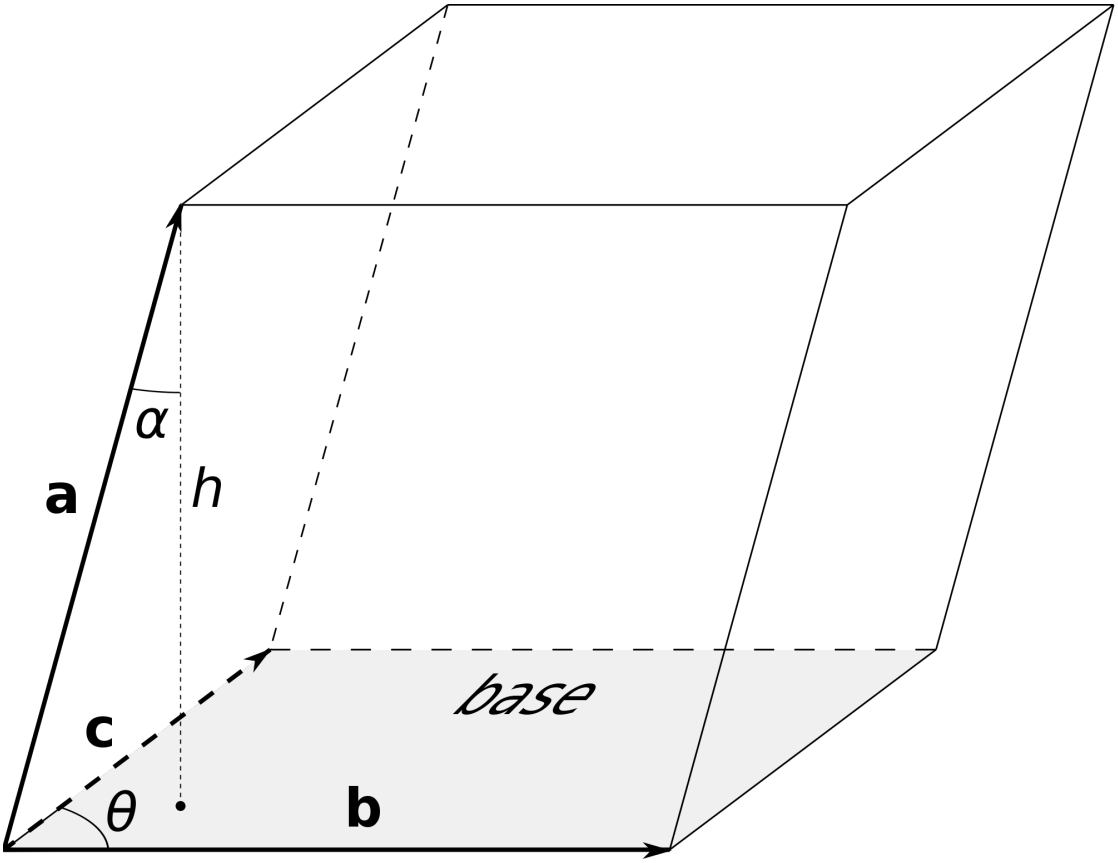
Figur 2.26: $f(x) = x^3$



Figur 2.27: $f(x) = sin(2x)$

2.7.7 Volumtransform

Til nå har det vært snakk om punkter i ett plan, det er sjeldent en bare jobber med et plan. Derfor utvides det til rommet. En vil fort se at mange av de samme konseptene som har blitt presentert for planet også vil gjelde for rommet. For å klare å definere et lukket volum i rommet må en i utgangspunktet ha fire punkter, 3 for grunnflaten og 1 som sier noe om høyden. Dette vil danne en 3-kantet pyramide. I Kalkulus[17] finner en at volumet til et parallellepiped kan finnes ut fra 3 vektorer som vist i figur 2.28.



Figur 2.28: Vektorene **a,b,c** definerer et parallellepiped. Bilde er hentet fra [`http://en.wikipedia.org/wiki/Parallelepiped`]

Volumet til parallellepiped er gitt som:

$$V_p = \left| \det \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \right| \tag{2.50}$$

Hvor $a_{1,2,3}$ er komponentene til vektoren **a**, tilsvandre for **b** og **c**. En vet at volumet til

en tetraederet er 1/6 av parallellepiped. Dermed blir V_t

$$V_t = \frac{1}{6}\left|\det\begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}\right| \tag{2.51}$$

Ut fra dette definerer jeg nå volumtransformen for en multivariabel funksjon $f(x,y)$ som:

$$V\{f(x_0,y_0)\} = \frac{1}{6D(x_0,y_0)}\left|\det\begin{bmatrix} x_0-\delta & y_0-\delta & f(x_0-\delta,y_0-\delta) \\ x_0 & y_0 & f(x_0,y_0) \\ x_0+\delta & y_0+\delta & f(x_0+\delta,y_0+\delta) \end{bmatrix}\right| \tag{2.52}$$

Hvis en lar :

$$\mathbf{a} = P(x_0-\delta), \quad \mathbf{b} = P(x_0), \quad \mathbf{c} = P(x_0+\delta)$$

Så kan 2.52 skirve som :

$$V\{f(x_0,y_0)\} = \frac{1}{6D(x_0,y_0)}\left|\det\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \end{bmatrix}\right| \tag{2.53}$$

Volumtransform har noe av de samme defektene som arealtransformen. Derfor normaliserer en, volumtransformen normalisert med $D(x_0,y_0)$.

$$D(x_0,y_0) = \sqrt{\mathbf{(a\text{-}c)} \cdot \mathbf{(a\text{-}c)}^T}$$

Som i arealtransformen om V blir liten, tyder det på at punktene er på en linje og en kan kaste punktet som er i midten. Oppsummeringen av det er:

- Liten A, kast punktet i midten.
- Stor A, behold punktet i midten.

2.8 Volumtransform punktreduksjon

Med kunnskapen man har tilegnet seg i delkapitlene om areal- og volumtransformen skulle man nå være i stand til å implementere dette i MATLAB. Det har blitt skrevet en MABLAB funksjon kalt *punktRed*. Algoritmen som er brukt for reduksjon med volumtransformen blir nå godt igjennom.

`punktRed` funksjonen trenger i utgangspunktet 3 argumenter (4 om en bruker K-means). Hvorav disse er oppgitt i rekkefølgen: punktskyen, antall punkter som ønskes redusert til og metode som skal brukes. Punktskyen er oppgitt som en matrise på formen $3 \times N$, antall punkter er et positivt heltall mindre en N.

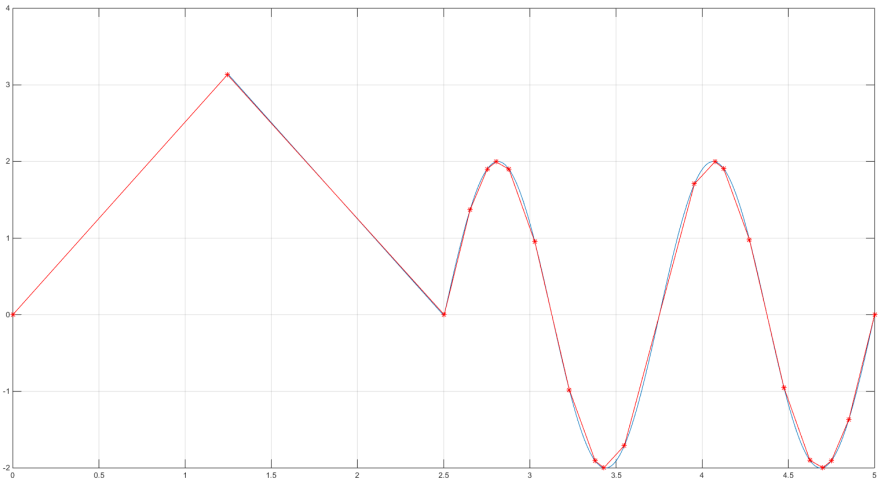
Når en kaller *punktRed* med volumtransformargumentet blir volumtransformen beskrevet i henhold til 2.52. Så finner en minsteverdi i transformen. Indeksen som samsvarer til minsteverdi i volumtransformen blir fjernet fra den originale punktskyen. Dette blir gjort helt til man har oppnådd en punktsky som er $3 \times k$. Hvor k er antall punkter en ønsket og redusere til.

Algorithm 2 Volumtransformen - punktreduksjon

```
1: procedure VOL(data,k)
2:     while size(data) >= k do
3:         V ← volumtransformen(data)        ▷ Bergen volumtransformen til data.
4:         idx ← min(V)                      ▷ Finn minste V, returnerer indexen.
5:         data(idx, :) = []                 ▷ Punktet idx i data blir kastet.
6:     return data                           ▷ Ferdig behandlet datasett.
```

Eksempel

Som i de tidligere eksemplene med punktreduksjon reduserer man fra 1800 til 25 punkter. I figuren under er dette gjort med Volumtransformen algoritmen ovenfor.



Figur 2.29: Volumpunktredusjon anvendt på testsignal. Hvor fargene gjenspeiler originalt og behandlet datasett. Hvor en bør legge merke til at punktene nå har blitt lagt litt mer strategisk med tanke på at toppene har fått flere punkter som beskriver denne delen av kurven.

Om en tar en titt på de tidligere eksemplene innen punktreduksjonen, er det min påstand at voltransformen gjør en mye bedre jobb ved og finne områder på kurven trenger mer informasjon i form av punkter en andre områder på kurven.

2.9 Eksperimentell sammenligning av Lineær-, Kmeans- og Arealpunktreduksjon

Denne delen blir i hovedsak undersøkt i MATLAB. Det blir laget et signal som blir brukt som testfunksjon[5], som blir redusert ved å velge ut punkter i henhold til teorien i 2.7.1, 2.7.2 og 2.7.7.

I denne testen blir et signal av 800 punkter redusert til 30 punkter (`Ns = 30`).

Når testsignalet er laget, benytter jeg meg av `punktRed`-funksjonen som ble derfinert tidligere i oppgaven. Når en kaller funksjonen med argumentet `'lin'`,`'kmean'` og `'vol'`.

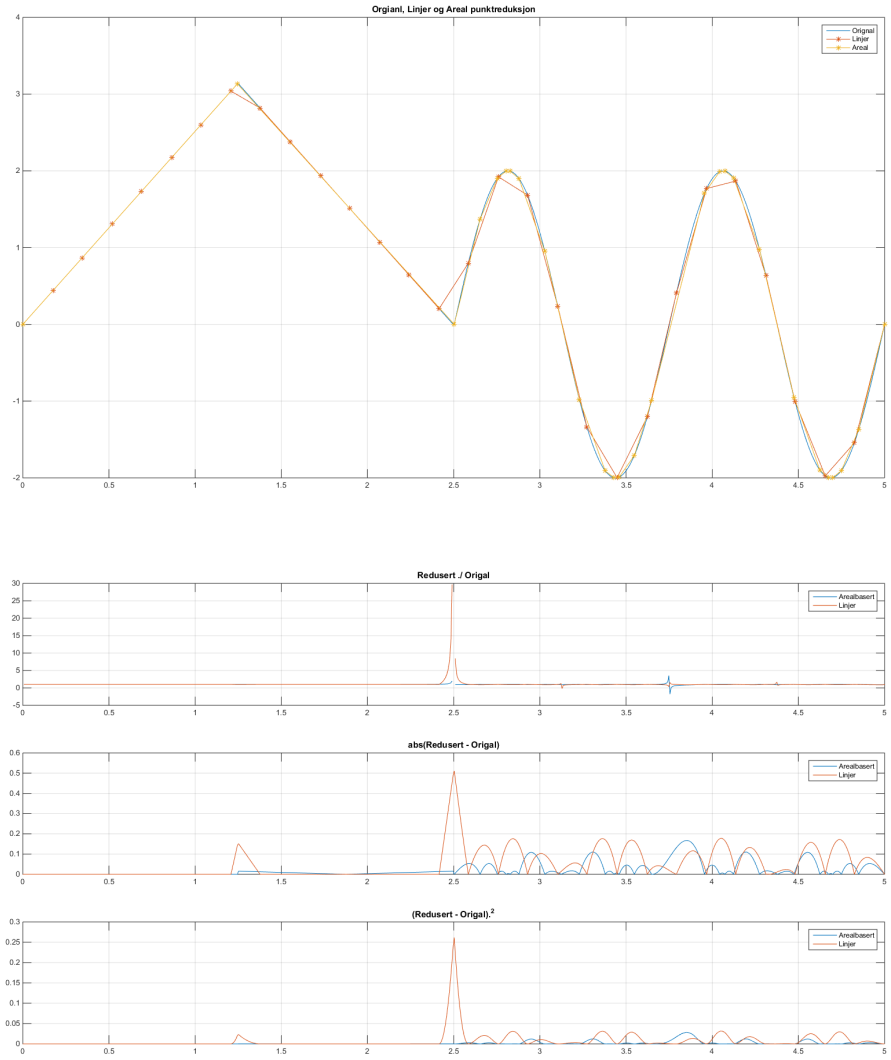
De behandlete datasettene blir lagret for sammenligningen med det originale datasettet. I tenkte om at roboten skal bevege seg i rettelinjer mellom punktetene passer det perfekt å benytte seg av lineær interpolering. Hvor en da finner linjen $y = ax + b$ mellom to punkter. Dette blir gjort for alle reduksjons metodene.

Kvaliteten på reduksjonen er blir målt som sum av kvadrert differanse, hvor en dette bør ha et tall så nærmer null som mulig. Under følger MATLAB koden som er brukt.

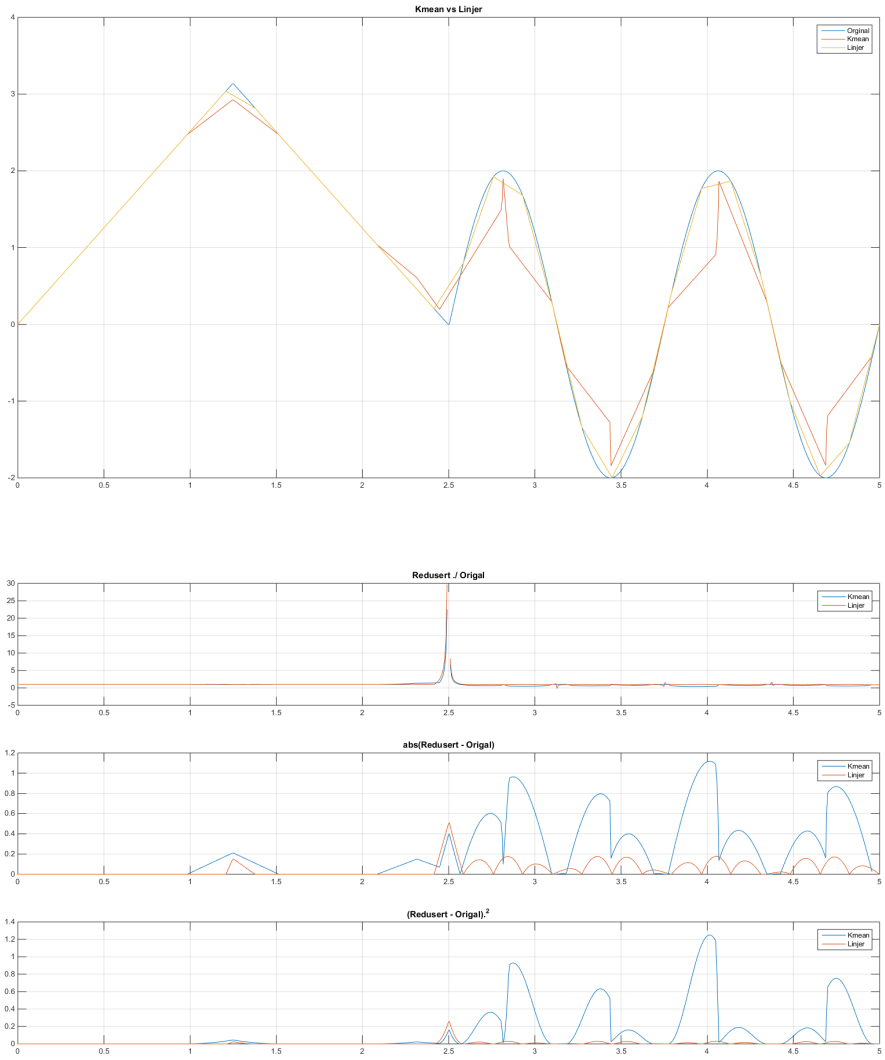
```
1  %% Bygging av testsignal.
2  N = 400; %Antall punkter
3  t = linspace(0,2*pi,N); %Tidsvektor.
4
5  y = [t(1:end/2), flip(t(1:end/2)), 2*sin(2*t)]; %Lag trekantpuls legg ...
       paa sinus paa slutten.
6  t = linspace(0,5,length(y)); % Lag ny tidsvektor som er reskalert.
7
8  wpSort = [t',y',ones(1,length(t))']; % Legg inn i matrise [800x3]
9  plot(wpSort(:,1),wpSort(:,2)),grid on %Plot
10
11 Ns = 30; %Antall punkter en oonsker og redusre til
12 % Punktreduksjon
13 wpVol = punktRed(wpSort,'vol',Ns,[0 0 1]); % Punktreduksjon, volum er ...
       valgt
14 wpVol = wpVol(:,1:2); % Kast siste kolonnen i matrisen.
15 wpLin = punktRed(wpSort,'lin',Ns,[0 0 1]); % Punktreduksjon, Linear er ...
       valgt
16 wpLin = wpLin(:,1:2); % Kast siste kolonnen
17 wpKm = punktRed(wpSort,'kmean',Ns,[0 0 01]); % Punktreduksjon, kmean ...
       er valgt
18 wpKm = wpKm(:,1:2); % Kast siste kolonnen
19 % Velger linear interpolering
20 vol = interp1(wpVol(:,1),wpVol(:,2),t);
21 lin = interp1(wpLin(:,1),wpLin(:,2),t);
22 km = interp1(wpKm(:,1),wpKm(:,2),t);
```

[5]Testsignlet er det samme som ble brukt tidligere (2.6.1) men uten støy.

```
% Plotting
figure(1)
subplot(311)
plot(t,(vol./y),t,(lin./y)),title('Redusert ./ Original')
legend('Arealbasert','Linear')
grid on
subplot(312)
plot(t,abs(vol-y),t,abs(lin-y)),title('abs(Redusert - Original)')
legend('Arealbasert','Linear')
grid on
subplot(313)
plot(t,(vol-y).^2,t,(lin-y).^2),title('(Redusert - Original).^2')
legend('Arealbasert','Linear')
grid on

figure(2)
plot(t,y,wpLin(:,1),wpLin(:,2),'*-',wpVol(:,1),wpVol(:,2),'-*')
grid on,title('Orginal, Linear og Arealpunktreduksjon')
legend('Original','Linear','Areal')

figure(3)
plot(t,y,t,km,wpLin(:,1),wpLin(:,2)),grid on,title('Kmean vs Linear')
legend('Orginal','Kmean','Linear')

figure(4)
subplot(311)
plot(t,(km./y),t,(lin./y)),title('Redusert ./ Original')
legend('Kmean','Linear')
grid on
subplot(312)
plot(t,abs(km-y),t,abs(lin-y)),title('abs(Redusert - Original)')
legend('Kmean','Linear')
grid on
subplot(313)
plot(t,(km-y).^2,t,(lin-y).^2),title('(Redusert - Original).^2')
legend('Kmean','Linear')
grid on

sumSElin = sum((lin-y).^2); % Sum av feil kvadrert
sumSEvol = sum((vol-y).^2); % Sum av feil kvadrert
sumSEkm = nansum((km-y).^2); % Sum feil kvadrert

disp(['Sum av kvadrert feil volum : ' num2str(sumSEvol)])
disp(['Sum av kvadrert feil linear : ' num2str(sumSElin)])
disp(['Sum kvadrert feil kmean : ' num2str(sumSEkm)])

Sum av kvadrert feil volum : 1.3943
Sum av kvadrert feil linear : 6.2852
Sum kvadrert feil kmean : 114.3263
```



Figur 2.30: Resultatet fra MATLAB-koden gitt på forrige side, det blir vist Areal vs Lineær punktreduksjon.



Figur 2.31: Resultatet fra MATLAB-koden gitt på forrige side, det blir vist K-mean vs Lineær punktreduksjon.

Punktreduksjons metode	kvadrert differanse
Volum	1.3943
Lineær	6.2852
K-mean	114.3263

Figur 2.32: I tabellen over er resultatene fra MATLAB-koden.

I et punktsett uten støy er min konklusjon at punktreduksjon ved bruk av volumtransformen er det beste valget ut fra testene som er blitt gjort.

Det har blitt gjort noen tester ved å legge på støy på testfunksjonen for deretter å glatte den. Resultatet er overraskende bra for volumtransformen.

Når det gjelder k-means redusering er denne testen nesten litt urettferdig og vil heller vise sitt fulle potensial når en får 3-dimensjonal data. Jeg har dessverre ikke hatt tid til å gå dypere inn i analysen av dette.

```
1 Sum av kvadrert feil volum : 2.4893
2 Sum av kvadrert feil linear : 7.0546
3 Sum kvadrert feil kmean : 105.6668
```

Figur 2.33: Resultat ved at punktreduksjon $SNR = 28.8$

3

Implementering

I denne delen av oppgaven gås det gjennom hvordan teorien som er presentert i forrige kapittel blir anvendt. Det kommer i hovedsak en forklaring i forkant av MATLAB-koden som blir presentert sammen med figurer som skal illustrere hva koden gjør og eventuelle problemer som kan oppstå. Mot slutten vil det vises hvordan koordinatene som har blitt estimert i MATLAB, blir lastet i robotkontrollen. Fullstending MATLAB-kode er tilgjenlig i appendix A og Rapid-kode i appendix B.

Hovedtrekkene i implementasjonen er :

1. Rapid
   (a) Gå til punkt P_1, Ta bilde I_1
   (b) Gå til punkt P_2, Ta bilde I_2
2. MATLAB
   (a) Last inn bildene I_1 og I_2
   (b) Bildebehandling
   (c) Punktreduksjon
   (d) Send koordinater til robotkontroller
3. Rapid
   (a) Les posisjon fra fil
      i. Gå til punkt lesete posisjon, hvis ikke EOF(End of file)
      ii. Gjenta de to forrige punktene
   (b) Gå til hardkodet startposisjon
4. Ferdig

3.1 Kamera brukt i oppgaven

Kameraet som er brukt i oppgaven er produsert av IDS (Imaging Development Systems) av typen uEye XS2. Dette er et av de minste USB kameraene som er tilgjengelig på markedet for tiden. uEye2 kan ta bilder opp til 5Mpix i 8bit oppløsning og har en fysisk utstrekning på $26.6mm \times 23.00mm \times 21.5mm$. Kameraet sto fastmontert på ABB roboten Rudolf v/UiS før oppgave start å ble dermed valgt til implementeringen.

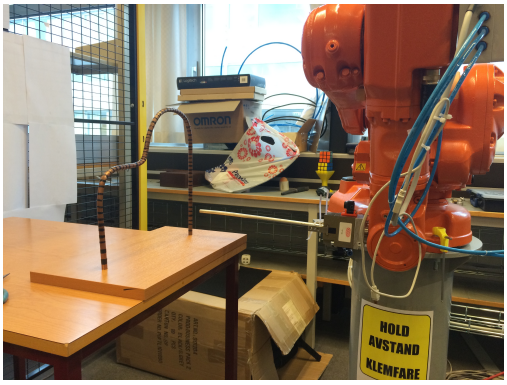
Figur 3.1: μEye2 utviklet av IDS her sammen med M-sjokolade for å illustrere den lille størrelsen på kameraet. Bilde er hentet fra `http://en.ids-imaging.com/`

3.2 Utviling av testverktøy

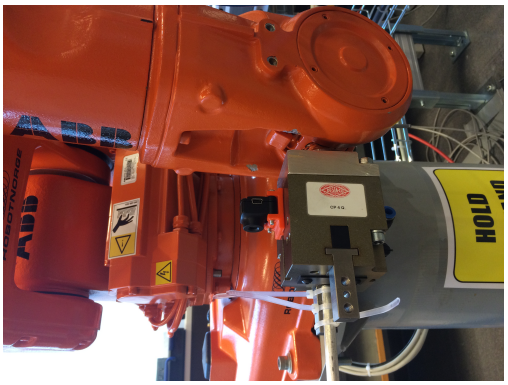
Før det endelige testverktøyet ble produsert, ble enkelt midlertidig testverktøy laget og montert på Rudolf. Testverktøyet versjon 1 er et aluminiumstag på ca. 25 cm som lar seg lett feste på gripperen med strips. Hvor kameraet også står montert. Det ble dermed laget et *tooldata* objekt i Rapid som er definert som *tStag := [TRUE, [[-5.26933,-12.3905,393.892] ,[1, 0, 0, 0]],[0.001, [0, 0, 0.001],[1, 0, 0, 0], 0, 0, 0]];*. Enden av staget er forskjøvet $[-5.2, -12.4, 494]mm$ i forhold til tool0. Dette oppsettet er ganske primitivt, men en kan teste mesteparten av oppgaven utenom å måtte bruke mye tid på design. Etterhvert ble en U-formet metallplate festet til staget. Dens funksjon var å demonstrere at beregningene knyttet til orientering var korrekt.

3.2.1 Versjon 1 og 2

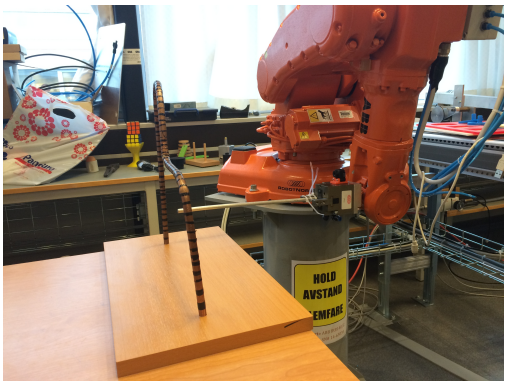
(a) Testverktøy montert på griperen



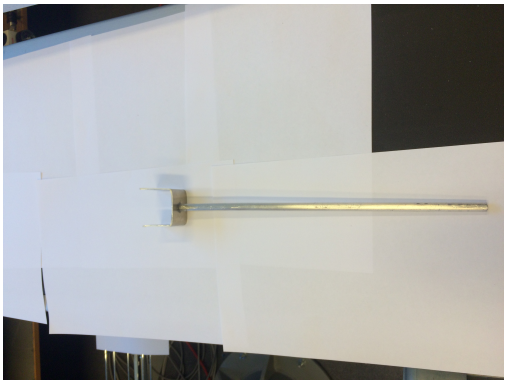
(b) Testverktøy montert på griperen



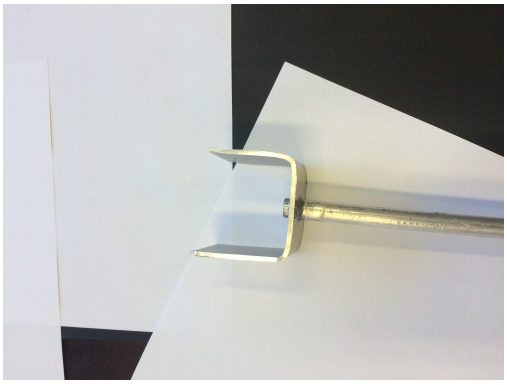
(c) Testverktøy montert på griperen, nærbilde



(d) Testverktøy montert på griperen sammen med banen som skal løses.



(e) Testverktøy versjon - 2



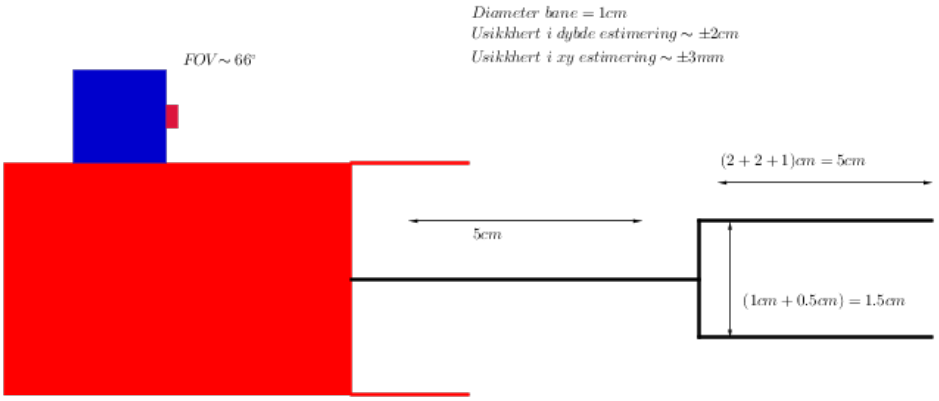
(f) Testverktøy versjon - 2, nærbilde

Figur 3.2: Testoppsettet fra forskjellige vinkler

3.2.2 Versjon 3 - "Gaffelen"

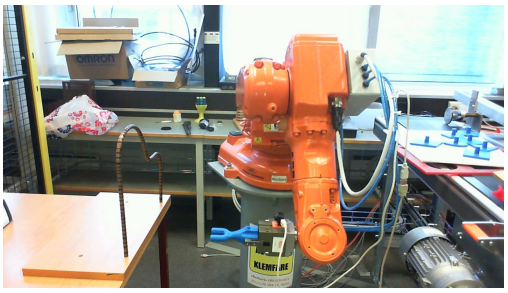
Det var behovet for større presisjon i verktøyet samt et ønske om å bruke de eksistensene parameterne og rask montering. Valget falt på en løsning som kan gripes av griperverktøyet som allerede er montert på roboten. Dimensjonene på testverktøyet beregnet ut fra teorien om feilmargin. Et utkast ble overlevert til Ståle Freyer som tegnet og 3D-printet "Gaffelen" i plast. For å unngå at testverktøyet ble ødelagt, ble testløypen byttet ut med et tynnere($5mm$)[1] kobberrør og dermed mer fleksibel.

Gaffelen som er laget i plast har fordelen med at den knekker som svakeste ledd ved en eventuell kollisjon. I tilegge vis en skulle være uheldig og ødelegge gaffelen kan det relativt enkelt produseres et nytt eksemplar

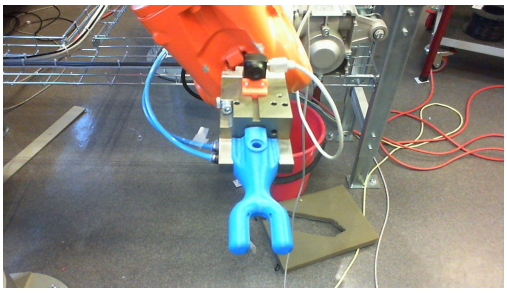


Figur 3.3: Skissen ble laget med utgangpunkt i 10mm ytre diameter på røret. Det er også lagt til en liten sikkerhetsfaktorer i usikkerhetsberegningene. Fargene representerer forskjellige deler av verktøyet som blir festet til tool0 på roboten, Gripper, Kamera og Gaffelen, størrelse forholde mellom dem stemmer ikke. Størrelsesforholdet mellom delene stemmer ikke.

[1] Den første testbanen hadde en ytre rør diameter på $10mm$



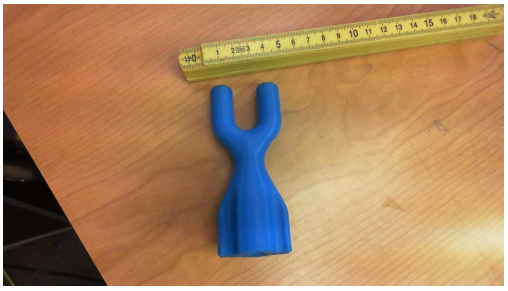
(a) Gaffelen montert på griperen



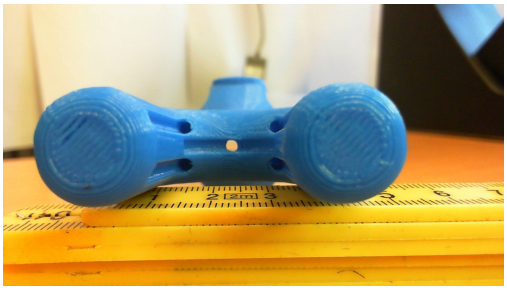
(b) Gaffelen montert på griperen, nærbilde

(c) Gaffelen montert på griperen, nærbilde

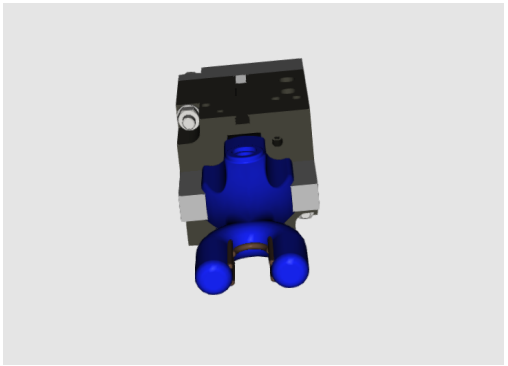
(d) Gaffelen ved siden av tommestokk



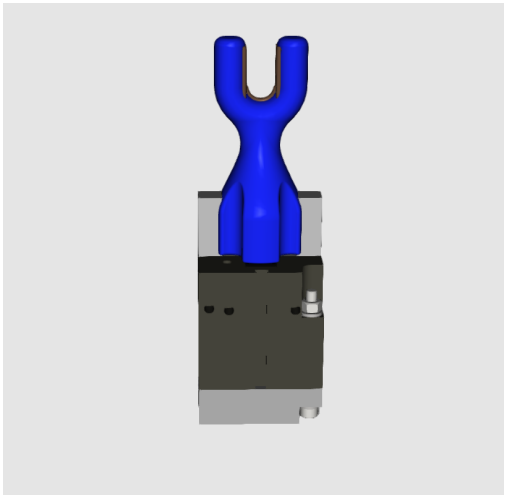
(e) Gaffelen ved siden av tommestokk



(f) Gaffelen nærbilde.



(g) CAD tegning, fra Freyer



(h) CAD tegning, fra Freyer

Figur 3.4: Testoppsettet med ”Gaffelen” fra forskjellige vinkler.

3.3 Ta bilder

For å ta bilder brukes et tredjeparts program μeye (levert av produsenten til kameraet). Dette brukes fordi en gjennom studentversjonen ikke har tilgang til "image acquisition toolbox" i MATLAB. Denne brukes selv om en bedre løsning hadde vært foretrukket.

Her opprettes et nytt koordinatsystem hvor en nå plasserer "tGripperCam" i dette koordinatsystemets origo. "tGripperCam" er funnet av Martin Tykhelle[18] som i sin hovedoppgave brukte to uker på å finne gode parametere som beskriver *tGripperCam* i forhold til tool0. Hvor en tidligere har vist at kvaternionene $q = [1, 0, 0, 0] \Rightarrow \mathbf{R} = \mathbf{I}$.

> Fra Martin Tykhelle sin hovedoppgave [18]: To uker med prøving, feiling, måling og korrigering gjorde til slutt at entooldata for kameraet ble funnet: tGripperCam:=[TRUE, [[53.69 ,0 ,41.19], [0.997969, 0.00541994, 0.0171534, -0.0611147]], [1, [0, 0, 1], [1, 0, 0, 0], 0, 0, 0]]; Denne peker på sentrum i kameralinsen med riktig rotasjon og viser at sentrum er 53.69 mm forskjøvet i X-aksen og 41.19 mm forskjøvet i Z-aksen (i forhold til festepunktet for verktøyet). Rotasjonen er uttrykt i quaternioner og representerer $-7°$ om Z-aksen, $2°$ om X-aksen og $0.5°$ om Y-aksen.

Når en nå velger "Ta bilde" på FlexPendaten vil roboten bevege seg slik at "tGripperCam" nå er å betrakte som origo. Roboten beveger seg til første posisjon. Når roboten har kommet frem benytter en seg av μeye til å ta bilde og å lagre. En lar roboten gå til neste posisjon og gjentar prossen ved å ta bilde.

Når bildene er tatt starter en på MATLAB-koden som er gått gjennom tidligere, hvor en får koordinatene som beskriver banen.

3.4 Estimere av Stereo Parametere

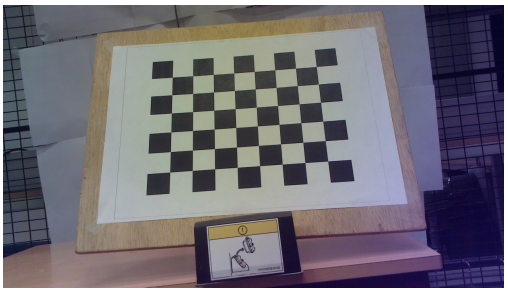
I MATLAB har en tilgang til en app som gir brukeren mulighet til å kalibrere kamera og stereoriggen samtidig. Dette er en geometrisk kalibrering ettersom interne og ytre parametere blir estimert ut fra geometriske betraktninger av et sjakkbrett. Sjakkbrettet er å betrakte som et plan hvor rutene må være kvadratiske og størrelsen må være kjent i *mm*, *cm* eller *inc*. Appen vil da bergene translasjon og orientering til kamera 2 i forhold til kamera 1. Samt internmatrisen til begge kameraene, som i vårt tilfelle skal være like. I tillegg blir også k_1, k_2 i ligning 2.23 og 2.24 estimert i henhold til [9]. Bildene tas ved at en velger ta bilder på flexpeneanten. I det man trykker vil roboten gå til første posisjon hvor den venter noen sekunder før den går videre til neste. Det er absolutt kritisk at scenen ikke endrer seg i tidsrommet mellom bildene blir tatt. Prosedyren til MALTAB-appen `'stereoCameraCalibrator'` er:

1. Ta bilder (Bør lagres som .png fil, for å unngå feil som følge av komprimering)

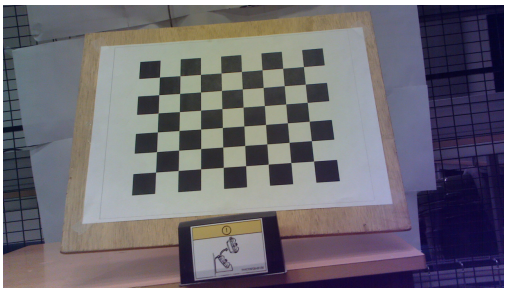
2. Last inn bilder med sjakkbrett som motiv
3. Kalibrer
4. Sjekk resultater og fjern eventulle bilder som gir dårlig estimering
   (a) Hvis dårlig, kalibrer på nytt
5. Lagre parametre i et `'stereoparameters'` objekt kaldt `'sP'`. Som en kan betrakte som kontante så ingen tuller mer kamera eller hvordan det er montert.

MATLABs - "stereoCameraCalibrator"

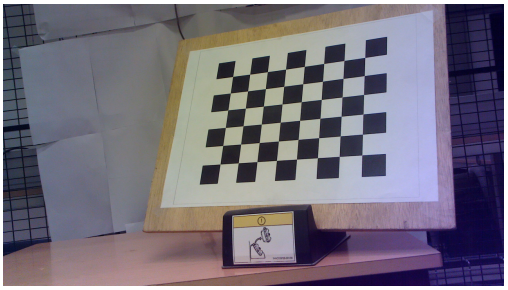
Det førte en gjør er å skrive ut et sjakkbrett mønster som en fester på flat overflate. Ved å feste sjakkbrettet til en overflate kan forsikre seg om at sjakkmønstret er å betrakte som et plan. En av forutsetning som blir lagt til grunne i Zhangs metode er at bildene som blir tatt har sjakkbrett som motiv. Scenen må altså være statisk i tidsperioden hvor bilde paret blir tatt. I Zhangs ble det lagt frem antall bilder som trenges for å finne parametere, men Zhangs anbefaler et titalls bilder for optimal kalibrering som følge av antagelser om støy.



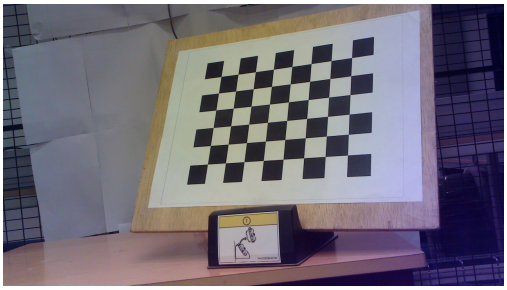
(a) Venstere kamera - bilde sett nr. 1



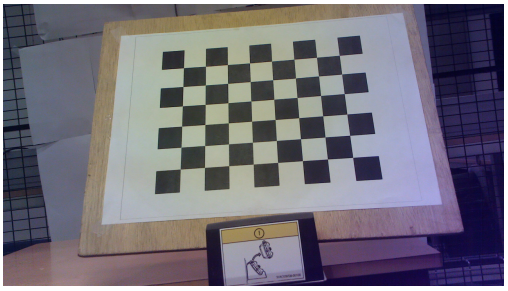
(b) Høyre kamera - bilde sett nr. 1



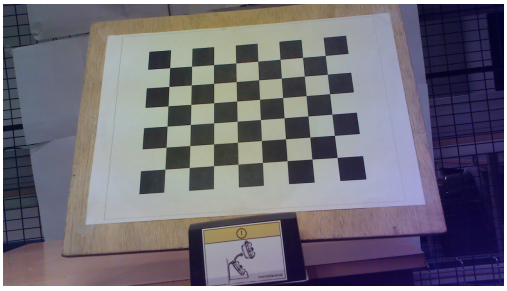
(c) Venstere kamera - bilde sett nr. 2



(d) Høyre kamera - bilde sett nr. 2



(e) Venstere kamera - bilde sett nr. 3

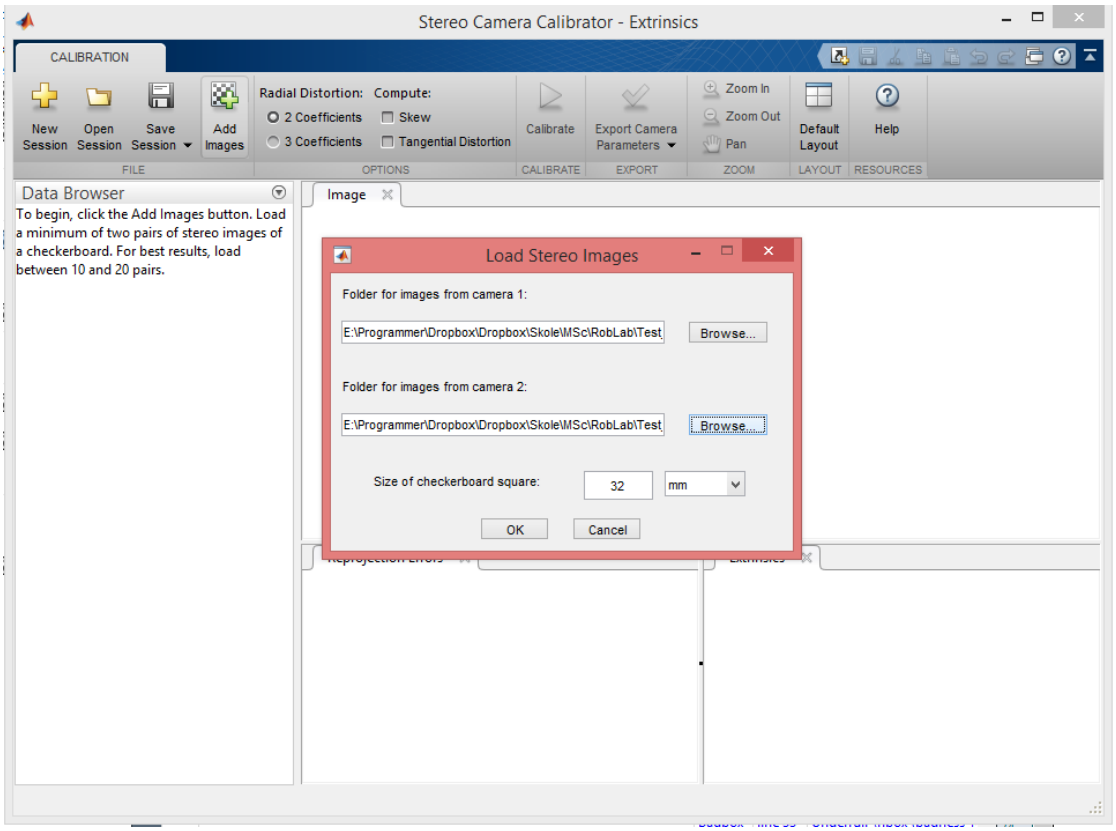


(f) Høyre kamera - bilde sett nr. 3

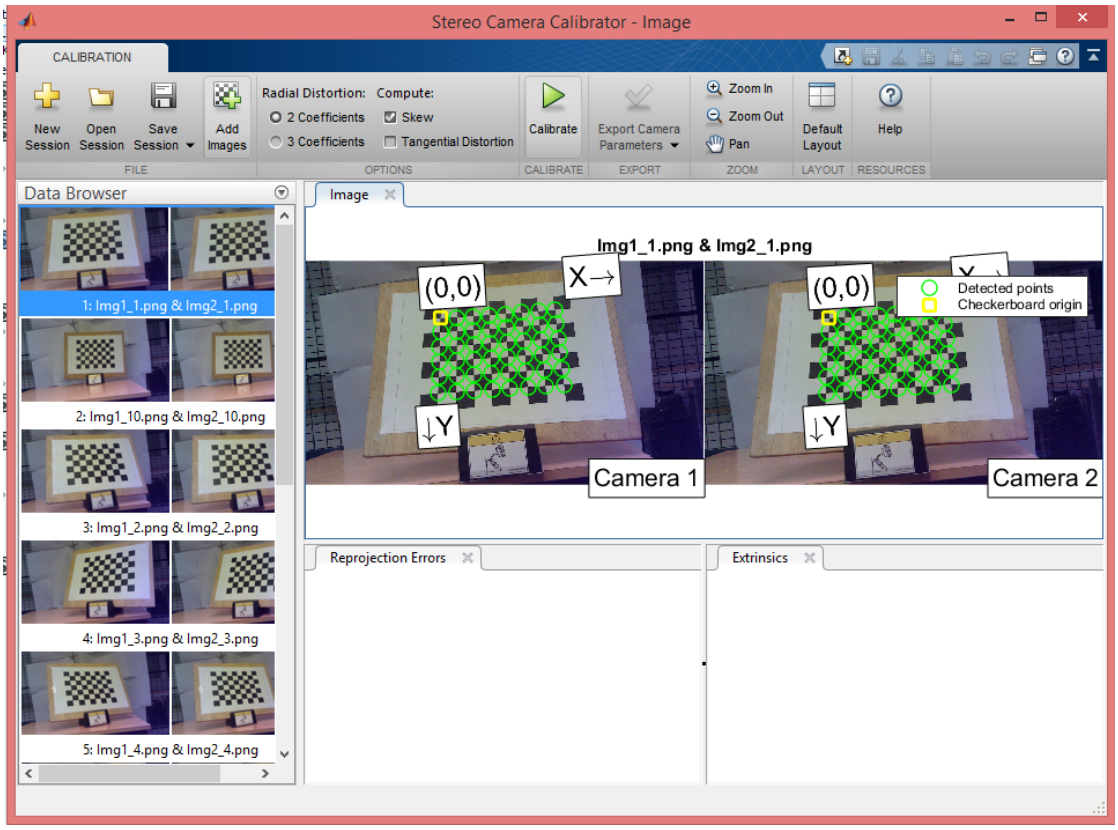
(g) Venstere kamera - bilde sett nr. 4

(h) Høyre kamera - bilde sett nr. 4

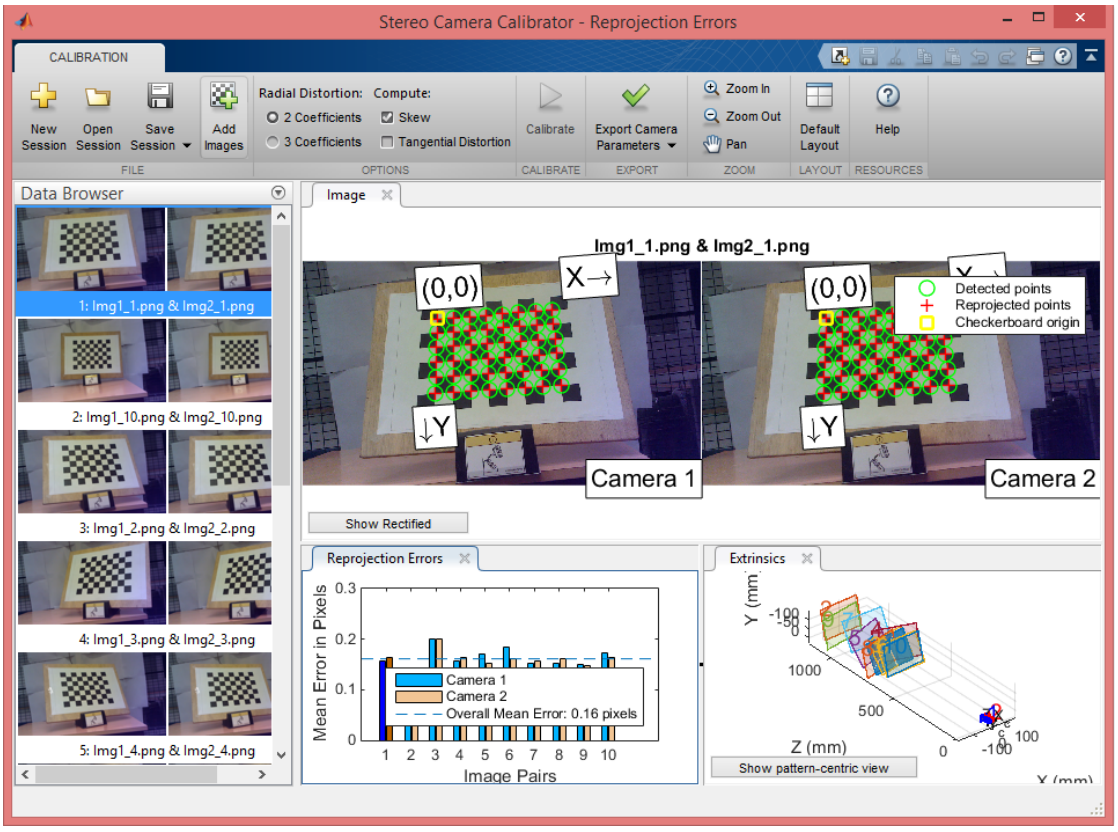
Figur 3.5: Fire bildesett av totalt tolv som ble brukt til å kalibrere stereoriggen.



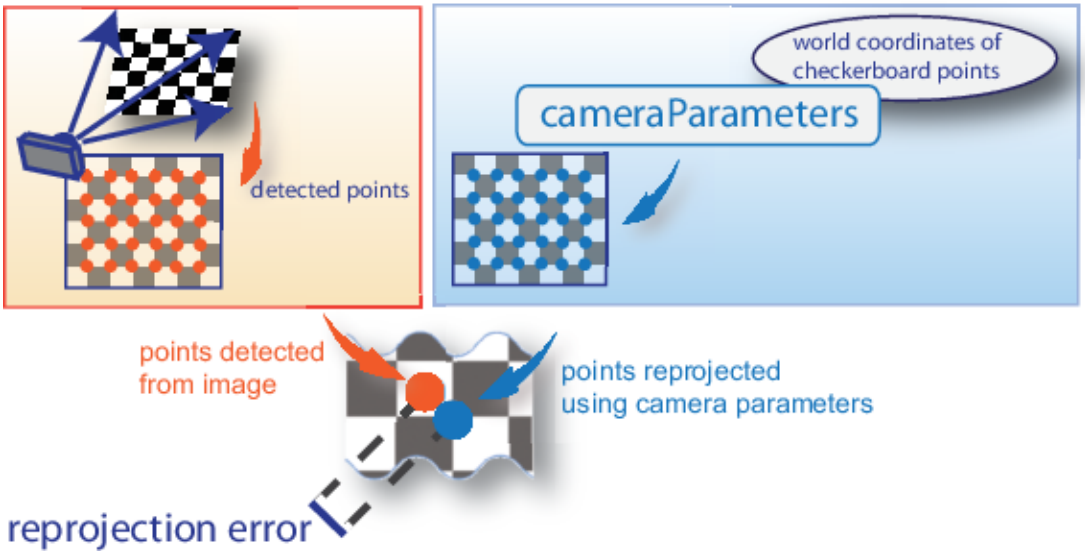
Figur 3.6: Størrelsen på kvardene på sjakkbrettet er målt til $32mm$. En trykker "ok" i figuren over og MATLAB begynner å analyse bilde. Hvor den sjekker at filnavn stemmer og finner hjørnene til alle sjakkbrettene.



Figur 3.7: I venstre del av bildet ser en de forskjellige bildesettene, opp kan en velge hvilke parametere som ønskes estimert hvor ”skew” nå er det samme som gamma fra 2.4.2. Radial distorsjon blir valgt til å estimers til to koeffisienter i henholdt til 2.4.3. Når en fornøyd med valgene trykkes det på ”calibarte” for å starte prosessen.



Figur 3.8: Kvaliteten på kalibreringen er blir vist i "reprojection error" vinduet. Feilen som blir vist i plottet er avstanden mellom punkt funnet i bilde og punktet som blir estimert igjennom modellen. Som er vist i neste figur. Figuren er hentet fra [10]



3.5 Bruk av Stereo Parametere

I forkant blir stereoriggen kalibrert og interne og ytre parametere for kameraet blir estimert og lagret i variabelen `'sP'` (Stereo Parameters). Variabelen er et objekt hvor en kan hente ut matriser som translasjon-, rotasjon-, fundemental- og essensialmatriser.

I kapittel 2 ble det gjennomgått hvordan rotasjon- og translasjonsmatrisene burde se ut for kamerariggen i oppgaven.

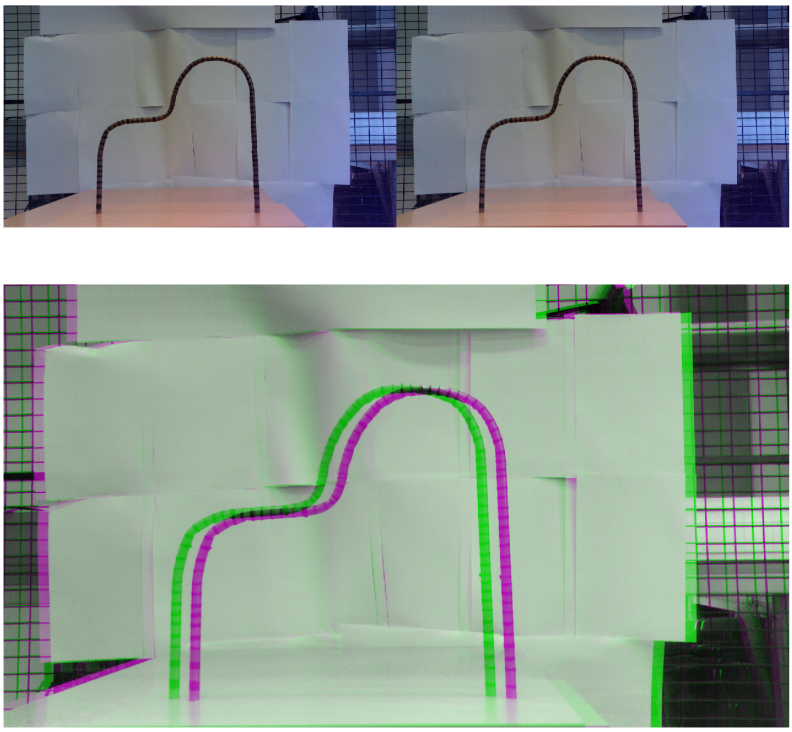
```
%% Last inn kamera parametre
load sP
sP = stereoParameteres(sP.CameraParameters1,sP.CameraParameters1,...
    sP.RotationOfCamera2,sP.TranslationOfCamera2);

%% Last inn bilder
sP.RotationOfCamera2 =
         1.0000      -0.0000      -0.0002
         0.0000       1.0000       0.0004
         0.0002      -0.0004       1.0000
sP.TranslationOfCamera2 =
        -19.8248      -2.0806      -0.2423
sP.WorldUnits =
         'mm'
```

Legg merke til at $||sP.TranslationOfCamera|| = 19.9352mm$ som er tilnærmet $20mm$ (langs x-aksen) som er tilnærmet det samme som roboten er programmert til å bevege seg. Samtidig er orienteringen mellom venstre og høyre kamera estimert til $\mathbf{R} \approx \mathbf{I}$.

3.6 Last inn bildepar der banen er motivet

To bilder som er tatt med kameraet som er festet på enden av robotarmen blir lastet inn.

```
%% Last inn bildepar
Img1 = imread('Basis.png');
Img2 = imread('20mmx.png');

% Vis bildepar
imshowpair(Img1,Img2)
```

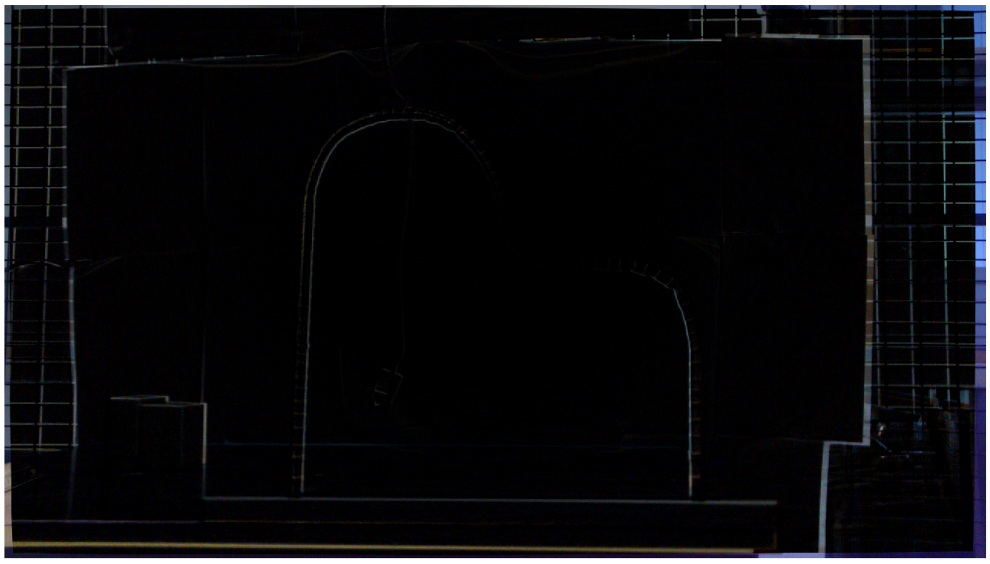


Figur 3.9: Bildet oppe til venstre er kalt ”basis”, mens bildet oppe til høyre er kalt ”20mmx”. Det nederste bildet er et sammensatt RGB-bilde som viser ”basis” og ”20mmx”, hvor ”basis” er den grønne banen og ”20mmx” er den lilla banen. Fargene er endret for å lett skille bildene fra vedrandre.

3.7 Bilde likeretting

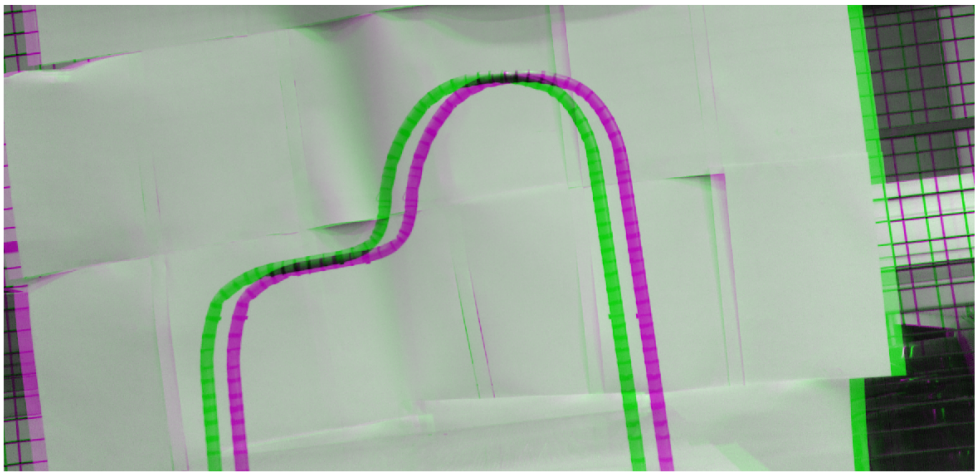
I 2.5.2 ble begrepet bilde likeretting introdusert, hvor en legger to bilder i samme bildeplan for å gjøre et eventuelt søk lettere når en skal estimere disparitet mellom bildene. Linseforvrenging blir også korrigert.

Figur 3.10: Originalt bilde og korrigert bilde i forskjellige fargebånd.



Figur 3.11: Absolutt verdi |Originalt - korrigert| bilde

```
1 %% Rectify Stereo Images
2 [Img1Rect,Img2Rect]= rectifyStereoImages(Img1,Img2,sP);
3 imshowpair(Img1Rect,Img2Rect)
```



Figur 3.12: "Rectifert" bildepar i forskjellige fargebånd.

3.8 Disparitet

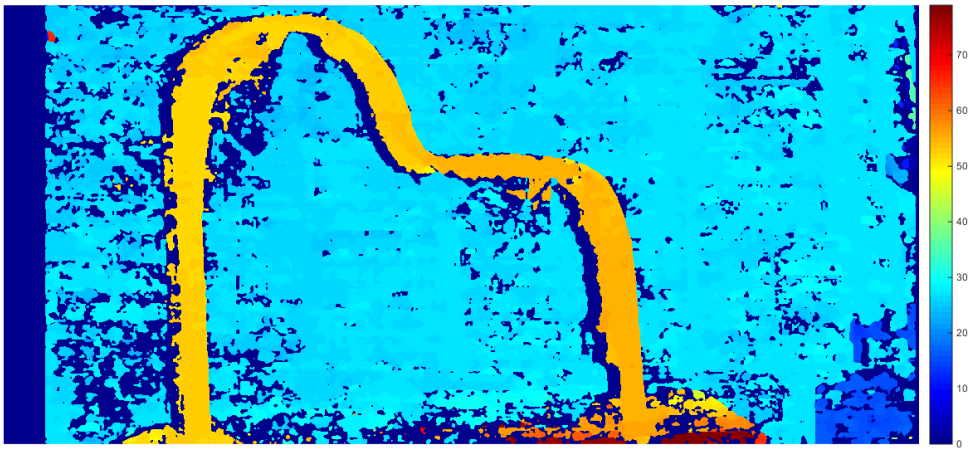
I 2.5.3 ble begrepet disparitet introdusert. Det blir benyttet en del argumenter til MATLAB metoden som blir gjennomgått.

Parameteren "Method", gir brukeren muligheten til å velge mellom to algoritmer. "BlockMatching" som benytter seg av summen av absolutt forskjell (sum of absolute differences (SAD)) for hver pixelblock gitt av blocksize.

BlockSize er størrelsen som blir sammenlignet når en lar bildene gli over hverandre.

DisparityRange er hvor langt i begge retninger algoritmen skal søke. Prøv å feile teknikken blir ofte brukt for å bestemme denne parameteren. En kan også prøve å finne den ved noen antagelser.
Løypen er målt på forhånd til å være ca. $80cm$ fra kamera. Ved å anvende ligning 2.27 kan en prøve på forhånd å estimere d. Som i teori biten har f blitt estimert til ca. $2000pix$ og $t = 2cm$. Dette gir $d = \frac{2000pix \cdot 2cm}{80cm} \approx 50pix$. Dermed kommer en seg unna den tunge prøve og feile metoden.

```
%% Disparity Map, bergnet at den burde bli ca. 50, gir meg selv ca ...
    1.5x margin.
disparityMap = disparity(rgb2gray(Img1Rect), rgb2gray(Img2Rect), ...
    'BlockSize',9,'DisparityRange', [0 80], 'Method','SemiGlobal');

marker_idx = (disparityMap == -realmax('single'));
disparityMap(marker_idx) = min(disparityMap(~marker_idx));

% Filtrer disparitymap
disparityMap = medfilt2(disparityMap,[5 5]);

imshow(disparityMap,[min(disparityMap(:)), max(disparityMap(:))])
colormap(jet(80)),colorbar
```



Figur 3.13: Som oppgitt av parameteren vises forskyvningen mellom bildene

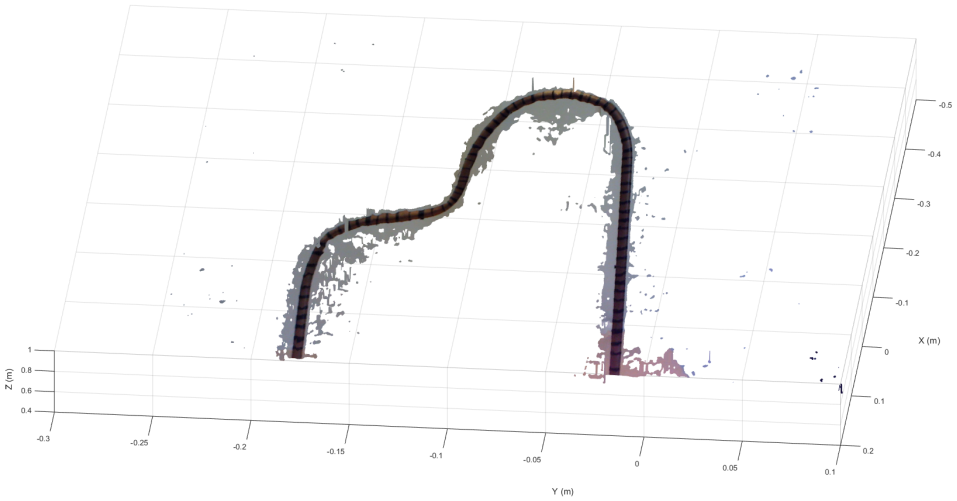
3.9 Reconstruct Scene

Disparitetbilde er estimert fra stereobildene som brukes til rekonstruksjon av scenen. Tidligere (2.5) ble ligningen $Z = f\frac{t}{d}$ utledet. Nå er høyre siden av ligningen kjent og en kan rekonstruere scenen.

Det er lagt begrensinger etter dybden i bildet. Roboten ABB IRB140 Type C på labben til UiS har ca. en meter rekkevidde. Av den grunn velger jeg at punkter som er lengre enn dette, er uinteressant og setter disse til `NaN`.

En mulig måte å rekonstruere punktskyen på uten å ta hensyn til forvrengingen, er å anvende forholdet fra ligning $Z = f\frac{t}{d}$ på alle elementene i `'disparityMap'`. Konstanten t er i millimeter, men jeg velger å gjøre den om til meter.

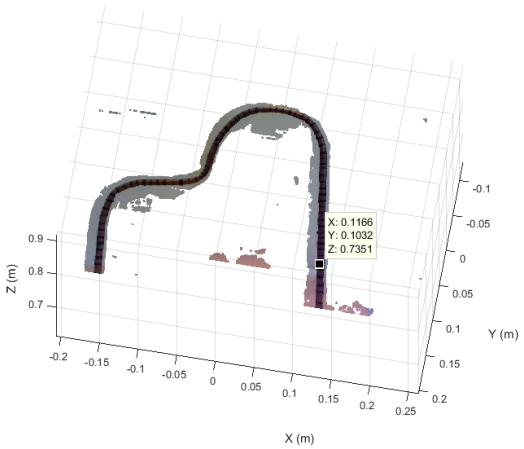
Sett punkter med større Z-komponent en 1 meter til `'NaN'`. Finn minimums- og maksimumsverdier i xy-planet i meter fra internmatrisen. Lag så et meshgrid og plot pikslene ved sin respektive farge fra `'Img1Rect'`. Dette er gjort i MATLAB-koden i appendix A under ”meshDemo”. Demokoden produserer figur 3.14.



Figur 3.14: Scenen er rekonstruert, hvor det ikke er tatt hensyn til forvrengingen.

I MALTAB finnes det en funksjon som rekonstruerer scenen og kompenserer for forvrenging som kommer som følge av bilde likeretting . Funksjonen heter `'reconstructScene'` hvor en oppgir `'disparityMap'` og `'stereoParams'`. Scenen blir rekonstruert ut fra kameraets venstre koordinatsystem, og den geometriske transformasjonen mellom `'Img1'` og `'Img1Rect'` blir også korrigert.

```
%% Lag punktsky
pC = reconstructScene(disparityMap,sP); %Rekonsturer scenen
pC = pC./1000; % Gjort om fra millimeter til meter
z = pC(:,:,3);

z(z < 0 | z > 1) = NaN;
pC(:,:,3) = z;
showPointCloud(pC,Img1Rect),axis equal
xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)')
```



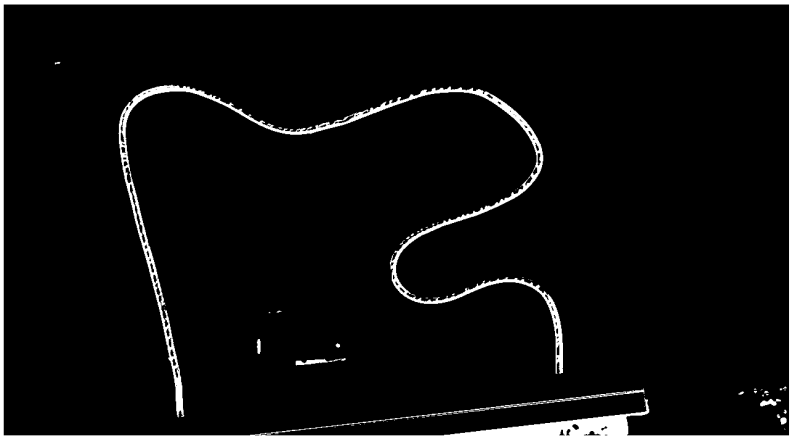
Figur 3.15: Punktskyen er funnet ved å bruke `'reconstructScene'`.

3.10 Binært bilde

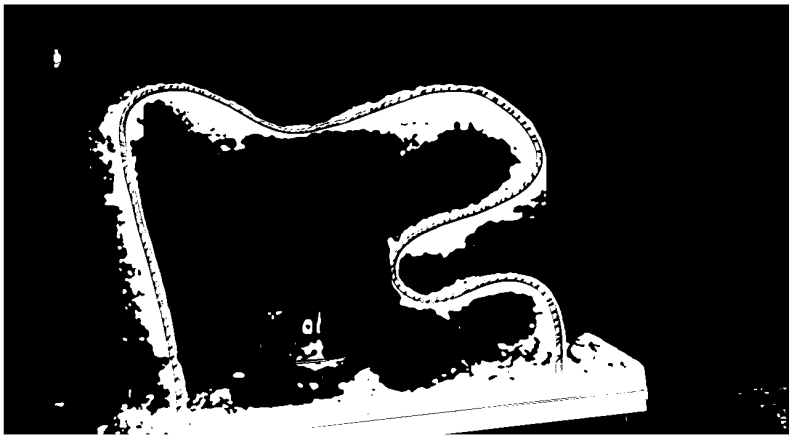
I denne delen gjøres bildet om til et binært bilde, hvor en bruker `graythresh` kommandoen. Jeg vet på forhånd hvilke punkter av bildet som er av interesse. Dette er punktene som ikke ble satt til NaN i 3.9. Dermed vet jeg hvilke piksler i `RectImg1` som skal "nulles". Et bedre valg er da å sette dem til gjennomsnittsverdien av bildet, slik at `graythresh` finner en optimal verdi for meg.

```
%% Kast punkter i bildet som er lengre bort enn 1 meter
mask = repmat(z > 0 & z < 1, [1, 1, 3]);
Img1Rect(~mask) = mean(Img1Rect(:));
imshow(Img1Rect)
```

Figur 3.16: Punkte som er satt til `'NaN'` blir grå.



Figur 3.17: Resultat av `im2bw` ved å sette `'NaN'` pikslene til gjennomsnittetsverdien av bildet.



Figur 3.18: Resultat av `im2bw` ved å sette `'NaN'` pikslene til 0.

3.11 Finne skjelettet

Nå som en har et bilde som i hovedsak kun inneholder informasjon om banen en skal løse i "dont touch the wire", er det en realtiv smal sak å hente ut senterlinjen til banen. Dette gjøres ved hjelp av de tre morfologiske operasjonene; skel, imfill og spur.

Skel- eller "median akse"-operasjonen tynner ut brede binære objekter til en pikselbred. I bildet under vises det hvordan en strek som er 30 pikselbred blir krympet til den er en pikselbred. Det siste bildet er XOR (Exclusive or) mellom de to første bildene. XOR operasjonen er for å fremheve at det er median til linjen som blir stående igjen.

Imfill fyller hullene i det binære bildet. Et hull er et sett av bakgrunns piksler som ikke kan nåes uten å krysse et annet sett av piksler.

Spur fjerner utstikkere i det binære bildet. Det blir noen ganger referert til som hår. 3.1 viser hvordan denne operasjonen fungerer med flere detaljer.

I denne oppgaven sitter en normalt igjen med ca. 1800 piksler som beskriver banen i xy-planet. Videre vet jeg at banen er målt til 60cm lang. Dette gir nå oppløsning på $\frac{1800pixel}{600mm} = 3pixel/mm$, når banen står ca. 80cm fra kamera.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.1}$$

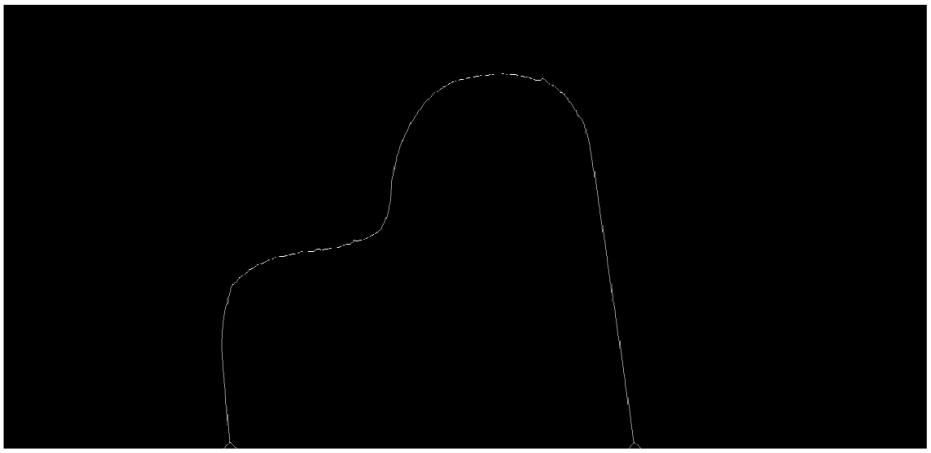
Figur 3.19: Spur operasjonen er anvendt i venstre sett og resultatet i høyre sett.

Figur 3.20: En rett linje blir brukt til å demonstrere hva skel operatoren gjør med et binært bilde

Figur 3.21: Skel operatoren finner medianen til linjen

Figur 3.22: Den binære operasjonen XOR blir brukt på de to øverste bilden for å fremheve at en finner medianen til den binære strukturen som var utgangspunktet.

```
%% Plukk ut intresange punkter fra pC
BW = ~im2bw(rgb2gray(Img1Rect),graythresh(rgb2gray(Img1Rect)));
BW = bwareaopen(BW, 2000); % Kast pixelobjekter mindre en 2000.
% Finner soorste objekt
stat = regionprops(BW,'Centroid','Area','PixelIdxList');
[~,idx] = max([stat.Area]);
BW = zeros(size(BW));
BW(stat(idx).PixelIdxList) = 1;
% Morph til skel
BW = bwmorph(BW,'skel',Inf);
% fill
BW = imfill(BW,'holes');
% Morph til skel
BW = bwmorph(BW,'skel',Inf);

% Morph spur, fjerning av "haar" som sitter igjen paa skel.
BW = bwmorph(BW,'spur',50);
disp(['Det er ' num2str(length(find(BW==1))) ' intresange punkter i BW'])
BW = bwmorph(BW,'clean',Inf);

imshow(BW)
```



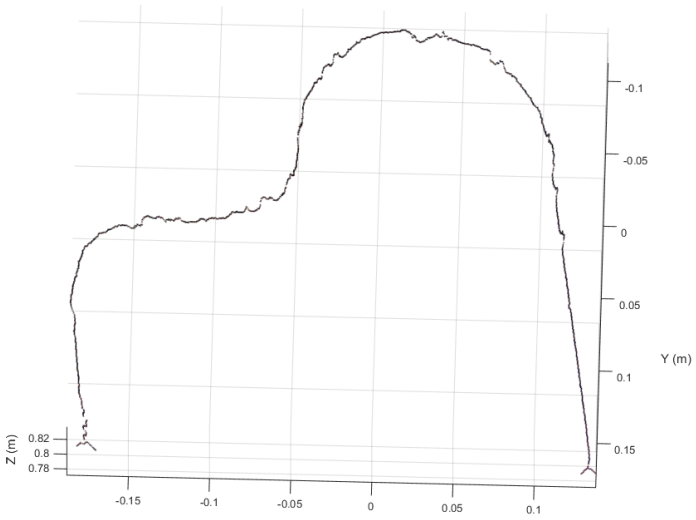
Figur 3.23: Figuren ser diskontinuerlig ut og er et resultalt av reduksjonen av størrelsen.

Figur 3.24: Bildet er tatt med venstre kamera, hvor en legger på skjelettet. Dette viser at en har klart å finne senter av banen relativt bra.

3.12 Sjelettet i punktskyen

Nå som en har definert senter av banen i et plan, må en transformere koordinatene til rommet. Litt tidligere ble det laget en punktsky ut fra `ImgRect1`, dette er samme bildet som ble brukt til å lage skjelettet. Ved å hente ut z komponenten av punktskyen å fjerne alle punkter som ikke er en del av `BW` vil en stå igjen med en bane i rommet som vist i figuren på neste side.

```
%% Plukk ut skel fra punktskyen
z = pC(:,:,3); %Hent ut z-komponenten
z(~BW) = NaN; %Sett alt som ikke er en del av BW til NaN.
pC(:,:,3) = z; %Legg tilbake i punktskyen.

showPointCloud(pC,Img1Rect),axis equal % Vis punktskyen
xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)') % Legg paa label.
```



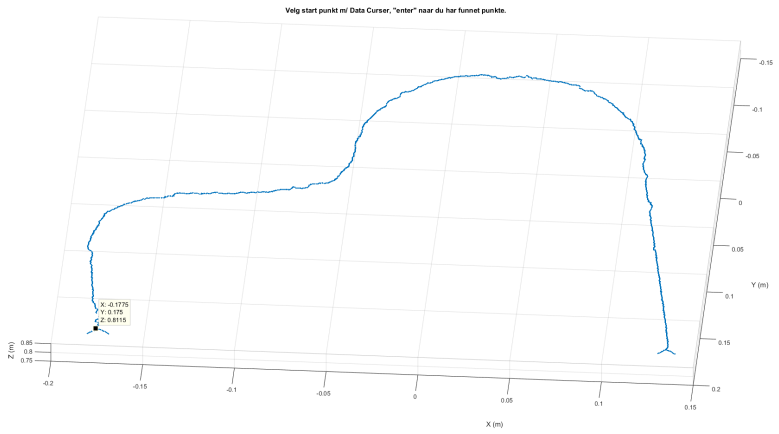
Figur 3.25: Alle pikslene som har verdien 0 i det binære bilde blir nå korresponderende punkt satt til `'NaN'` i punktskyen. Hvor er dermed vil så igjen med et skjelett av den ønskede banen.

3.13 Koordianter i felles matrise

Frem til nå har punktene lagt i en stor grid, som er tilnærmet størrelsen av det originale bildet (1080x1920x3). Ettersom jeg vet akkurat hvilke punkter som hører til banen som skal løses, kan jeg hente punktene ut fra et maskegitter (eng.:mashgrid) og legge dem inn i egne matrise som representerer x,y og z. Matrisen `wP` (world points), inneholder xyz komponentene.

I utgangspunktet har banen to løsninger; ved at man beveger seg fra venstre til høyre eller fra høyre til venstre. Dette er et valg brukeren skal ta. I figuren under viser jeg at roboten skal starte på venstre side av banen.

```
%% Plukk ut punkter, og i XYZ koordiantsystem
x = pC(:,:,1);
y = pC(:,:,2);
z = pC(:,:,3);

wP = double([x(BW),y(BW),z(BW)]); %Legg i felles matrise.

% Kast wP med NaN (Nesten aldri noen).
wP(isnan(wP(:,3)),:) = [];

% Vis data aa velg startpunkt paa banen.
fig = figure;
plot3(wP(:,1),wP(:,2),wP(:,3),'*','MarkerSize',1);,grid on, % Legg ...
    merke til '*' argumenetet.
camorbit(0, -125, 'camera', [0 1 0]),
xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)')
title('Velg start punkt m/ Data Curser, "enter" naar du har funnet ...
    punktet.')

% Hent info fra data curser.
dcm_obj = datacursormode(fig);
set(dcm_obj,'DisplayStyle','datatip',...
    'SnapToDataVertex','off','Enable','on')
pause
c_info = getCursorInfo(dcm_obj);
close(fig)

% Start punkt XYZ
startP = c_info.Position;
disp(['Startpunktet er : ' num2str(startP)])
Startpunktet er : -0.17749       0.175     0.81151
```



Figur 3.26: Valgt startpunkt.

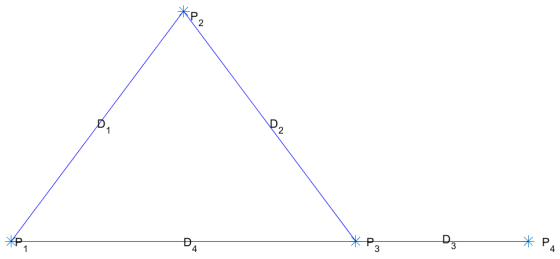
3.14 Sortere datasett

Punktskyen er ikke sortert og det trengs en sortert samling av punkter for å kunne gjøre noe videre. Derfor velger jeg punktet som er nærmest `startP` i `wP` og legger det først i matrisen `wPsort`. Dette punktet blir deretter slettet fra `wP`. Det nye punktet i `wPsort` blir valgt som nytt utgangspunkt og det nærmeste punktet blir på nytt funnet i `wP` og lagt til i `wPsort`. Nærmeste punkt er valgt som minste euklidsk avstand (eng:.euclidean distance).

```
%% Trenger aa sotere datasett, staar i startpunkt wP(1,:) vil naa ...
    finne ...
% wP(x,:) som gjor at D = sum((P-wP(x,:)).^2 er minst mulig.
P = startP;
wPsort = nan(size(wP));

g = waitbar(0,'Sorterer');
l = length(wP);
for i=1:size(wP,1)
    D = nan(size(wP,1),1); %Dist vektor.
    for k=1:size(wP,1)
        D(k) = sum((P-wP(k,:)).^2); %Beregn sq eq dist mellom et punkt ...
            og alle andre.
    end
    [~,idx] = min(D); % Finn minste dist

    P = wP(idx,:); %Plukk ut
    wPsort(i,:) = P; %Legg inn
    wP(idx,:) = NaN; %Sett gamle punkter til nan.
    waitbar(i/l) % Oppdater statusbar
end
close(g))
```

3.15 Kaste dårlig punkter

Sortering vil ikke alltid fungere helt som planlagt. Et senario som oppstår en del ganger under sorteringen er vist under. Hvis en starter i P_1 og antar at $D_4 < D_1$ så er nærmeste punkt P_3. I P_3 er nærmeste punkt P_4. Nå er det kun P_2 som er igjen i datasettet og blir derfor plukket ut. Jeg betrakter P_2 ikke lengre som en del av settet og fjerner det. Koden under prøver å finne slike punkter i `wPsort`. Tanken er nå at avstanden mellom punktene vil få en drastisk økning mellom P_4 og P_2.



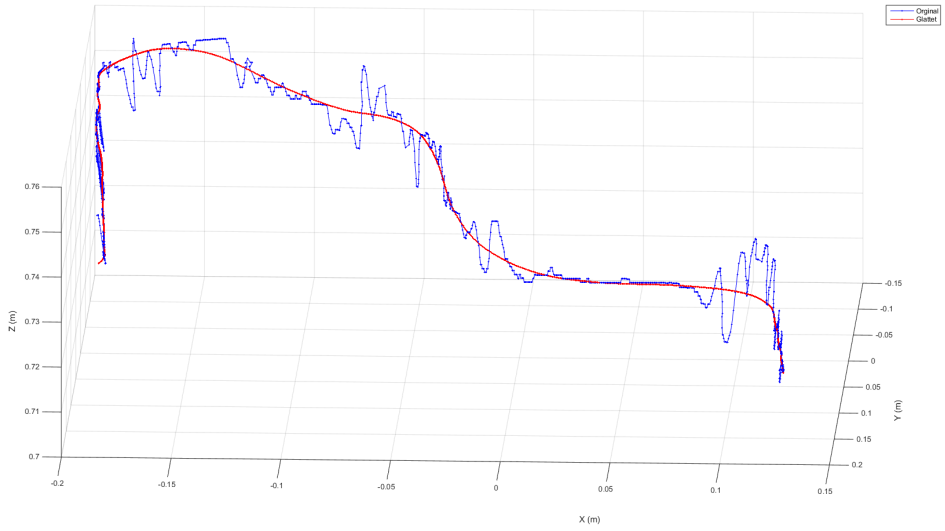
Figur 3.27: Punkt $P_1 \rightarrow P_4$, blir brukt til å demonstere problemet ved å bruke euklidsk sortering.

```
%% Distansetesting, regen ut D mellom punktene.
D = zeros(1,length(wPsort));
for i=1:length(wPsort)-1
    D(i) = (wPsort(i+1,:)-wPsort(i,:))*(wPsort(i+1,:)-wPsort(i,:))';
end
idx = find(std(D) < D); %Finn punkt som er lenger borte enn std.
idx = min(idx); %Finn minste idx
wPsort(idx:end,:) = []; %Kaste fra minste idx til slutten
```

3.16 Punkt glatting

Glattingen blir gjennomført ved å estimere en linje som beskriver et visst antall punkter som er valgt. Teorien for hvordan en kan estimere linjen som ble lagt frem i 2.6 samt en beskrivelse av KF.

```
wPsort = temp;
plot3(wPsort(:,1),wPsort(:,2),wPsort(:,3),'b-*','MarkerSize',1),hold on
r = [50,50,100]/2;
for i = 1 : 3
    wPsort(:,i) = poly1LMS(wPsort(:,i)',r(i),40)';
end

plot3(wPsort(:,1),wPsort(:,2),wPsort(:,3),'r-*','MarkerSize',1)
xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)')
grid on,set(gca,'YDir','reverse'),legend('Original','Glattet')
```



Figur 3.28: Visualisert punktskyen, før glatting og etter glatting

3.17 Punktreduksjon

Teorien som ligger til grunne for hvordan `punktRed` funksjonen fungerer, ble diskutert i 3.9. For detaljert m-kode kan en se appendix A.

Funksjonen `punktRed` fungerer ved at man sender inn en sortert punktsky[2]. Deretter oppgir en metoden en ønsker at punktreduksjonen skal skje på; vol, kmean eller lin. Så finner en antall punkter en ønsker å redusere punktskyen til, samt et startpunkt som brukes til å sjekke punktskyen for feil etter redusering.

Andre linje i koden under er; `nywP = punktRed(wPsort,'vol',50,startP)`. Første argument er punktskyen. Andre argument er å bruke volumtransformen til punktreduksjonen. Tredje argument er at man ønsker 50 punkter hvor startpunktet er gitt av startP.

```
1  %% Punktreduksjon
2  nywP = punktRed(wPsort,'vol',50,startP); % Punktreduksjon
3  figure(2),clf %Figur
4  plot3(nywP(:,1),nywP(:,2),nywP(:,3),'-r*','MarkerSize',8),hold on ...
       %plot redusert punktsky
5  plot3(wPsort(:,1),wPsort(:,2),wPsort(:,3),'*','MarkerSize',1) % plot ...
       original data
6  plot3(nywP([1 end],1),nywP([1 end],2),nywP([1 ...
       end],3),'y*','MarkerSize',10),grid on, %Marker start og sluttpunkt
7
8  %Legg paa tekst
9  text(nywP(1,1),nywP(1,2),nywP(1,3),...
10 '\leftarrow\fontname{times}Start punkt','FontSize',14)
11 text(nywP(end,1),nywP(end,2),nywP(end,3),...
12 '\leftarrow\fontname{times}Slutt punkt','FontSize',14)
13
14 %Label og akser
15 xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)')
```

[2]Unntak hvis en bruker ”kmean” argumentet

3.18 Orientering

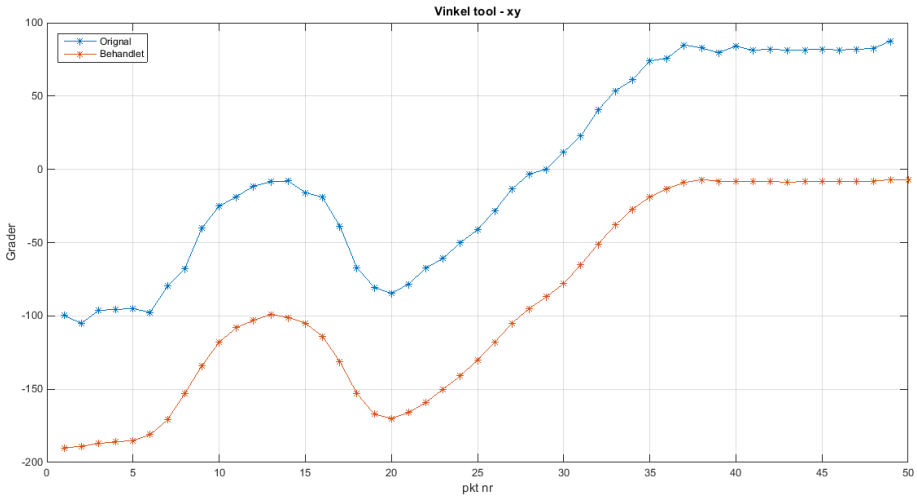
Frem til nå har en klart å finne et sett med punkter som gir en beskrivelse av banen som roboten skal løse. For at roboten skal kunne klare å løse banen uten å komme borti banen, må en også regne ut vinkelen som verktøyet skal ha. Vinkelen som er av størst interesse er gitt av xy-planet.

$$\theta_i = \arctan 2 \left(\frac{P_{i,y} - P_{i+1,y}}{P_{i,x} - P_{i+1,x}} \right) + \theta_2 \tag{3.2}$$

P_x er x-komponenten og P_y er y-komponenten til punkt P. Det er en rotasjon på 90° mellom kameraets koordinatsystem og roboten blir dermed $\theta_2 = 90°$. Det velges å jobbe i grader ettersom en oppgir vinkeler i grader i Rapid.

Når en regner ut differansen mellom punkt numerisk vil en alltid få et element mindre en utgankspuntet. Dette har jeg valgt å løse ved å legge til et ekstra element som er lik det siste opprinnelige elementet.

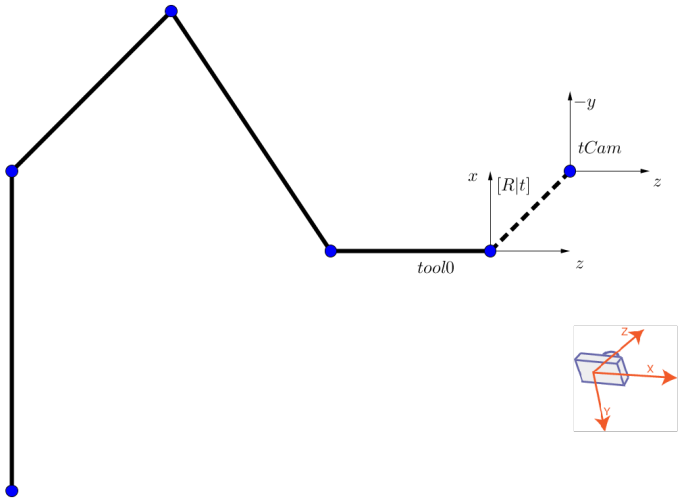
```
%% Ut fra plottet over ser en at xy-planet kan brukes til aa finne vinkel
% toolen maa ha for aa faa rett orientering i forhold til xy. Vinkelen ...
    en finner er
% gitt som forholdet mellom p(i,:) og p(i+1,:). Hvor en igjen kun er
% interessert i foorste og andre element i p.

deltax = diff(nywP(:,1)); %Endring i x
deltay = diff(nywP(:,2)); %Endring i y

theta = atan2d(deltay,deltax); % Finn vinkel

figure(1),clf
plot(theta,'*-'),hold on, %Original

theta = medfilt1(theta); %Median filtering.
theta = poly1LMS(theta,10,1); % 1-orden tilnerming mellom punkter,lms.
theta = [theta; theta(end)]; % Diff fjener et punkt, antar at to siste ...
    punkt er like.
theta = round(theta-90); % Rotajon i forhold til kamera og tool0.
plot(theta,'*-') % Behandlet theta
grid on,legend('Orignal','Behandlet')

title('Vinkel tool - xy'),ylabel('Grader'),xlabel('pkt nr')
```



3.19 Overføring til robot

Jeg har valgt å bruke File Transfer Protocol (FTP) til å overføre en textfil som innholder X,Y,Z og orientering til robotkontrollen som da leses i Rapid. En annen mulighet var å benytte seg av ABB's PC SDK som gir muligheten til å overføre data direkte til variabler i Rapid. Jeg har valgt FTP metoden fordi jeg ønsker muligheten til å gjøre Rapid-koden uavhengning av en datamaskin. Bruk av PC SDK drar også med seg noen andre ulemper. En kan ikke lenger jobbe med en 64bit's versjon av MATLAB i tillegg til en begrening på arrays størrelse til 1024byte.[3]

Bildet under viser hvordan en del koordinatsystemer er realtert til hverandre. Det er gjort en ending ved at en velger arbeidsobjektet *tWork* som koordinatsystem. Det kommer frem at en må rotere kamerakoordinatsystemet med $\frac{\pi}{2}$ om z-aksen.



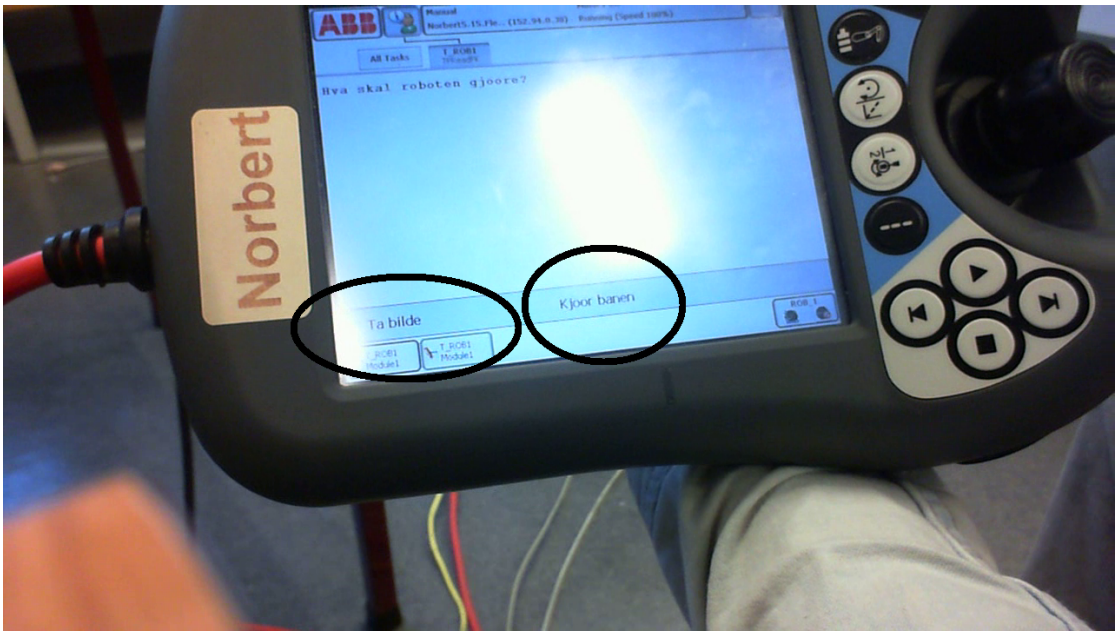
Figur 3.29: Matrisen [**R**|**t**] er sammenhengen mello tool0 og kameraoptiskesenter.

[3]Det har vist seg i etterkant at dette er fikset i den nyeste versjonen av PCSDK. Ettersom man også må oppdatere robotkontrollen ble dette lagt til side, i tilfelle oppdateringen ikke skulle være vellykket (verste tilfelle ødelegge robotkontrollern) og dermed ødelegge semesteret for lagt flere en forfatter.

```
% Rotasion om z.
R = @(theta)    [cos(theta) -sin(theta) 0;
                 sin(theta) cos(theta) 0;
                     0           0       1];

t = [0 0 0]; % Avstand fra avstand mellom tool0 og flyttet  kamera kord
RT = [R(pi/2) t'; 0 0 0 1]
pNy = nan(size(nywP)+1);

% Nytt punkt blir naa pNy = RT * P
for i = 1:length(nywP)
    pNy(i,:) = RT*[nywP(i,:) 1]';
end

% Rydd opp, gjort om fra homogene kord.
pNy(:,4) = [];
pNy(end,:) = [];

pNy = round(pNy*1000); % Gjor om til mm, rund av til nermeste.
% Skriver alle datapunktene til fileName
fileName  = 'XYZ.txt'; %Navn paa filen
fileID = fopen(fileName,'w'); %Gi matlab rettigheter til aa skrive
fprintf(fileID,'%6.4f:%6.4f:%6.4f:%6.4f\r\n',[pNy,theta]');  %Skriv
fclose(fileID); %Lukk skriving

% % Last opp til robot v/FTP
ip = '152.94.0.38'; %IPen til robot
usrName = 'nordert'; %Brukernavn
pw = ''; % Passordet,  ikke idelt og lagre "husj husj".

ftpCli = ftp(ip,usrName,pw); %Opprett FTP objekt og koble til

try
    mput(ftpCli,fileName); %Prov laste opp filen
catch
    disp('Noe gikk galt'); %Noe gill kaldt melding
end
close(ftpCli) %Avslutt FTP client

disp(['Opplastning ferdig, start rapid-kode'])
```

3.20 ABB - Rapid

Rapid er et programmeringsspråk utviklet av ABB til programmering av Deres roboter. Koordinatene er lagret i en tekstfil på robotkontrollen som en kan få tilgang til gjennom Rapid. Det har blitt skrevet meget enkelt brukergrensesnitt som gir bruken to muligheter; Ta bilde eller Kjør bane.



Figur 3.30: Gjennom FlexPendanten får brukeren to valg; Ta bilde eller Kjør bane.

3.21 Utføre bevegelse

På dette tidspunktet i programmet har en funnet et sett med koordinater som beskriver banen. Neste problemet blir å lese inn koordinater fra en tekstfil som er lagret på robotkontrollern sin harddisk. Rapid-variabelen *fileName* er oppretter som en string og gir Rapid informasjon om hvor datafilen ligger, samt navnet. En oppgir også strukturen, hvor en har valgt *:* som separeringstegn.

3.22 Forfatter prøver

Jeg følge meg litt gavmild og ga meg selv ti forsøk på å løse testbane 2. Hvor jeg holder gaffelen i hånden og starter en stoppeklokke med andre hånden. Hvor jeg teller antall ganger jeg nær banen. Dette hørtes ganske lett ut helt til det skulle gjennomføres. Til ettertanke burde en kanskje hatt mer enn en person som prøver seg. Men resultantene var meget lite å skryte av. Hvor det er ganske interessant avbalansering en må gjøre, kjapt å komme nær eller sakte uten feil? Utfallet er oppsummert i tabellen under.

Beste tid	10[s] / Ant. nær 5
Dårligste tid	30[s] / Ant. nær 0
Gjennomsnittstid	23[s] Gj.snit ant. nær 3.2

4

Resultater

4.1 Planlagt bane

Oppgavens mål er å gi roboten muligheten til å spille "Dont touch the wire", ved hjelp av et kamera og bildebehandling. For å estimere dybden(avstanden fra kamera til banen) ble oppgaven utvidet til et stereokameraoppsett. Oppsettet bygger på robotens suverene presisjon og evne til å repetere, som gjør at en kan kalibrere oppsettet. Løsningen gir sluttbrukeren muligheten til å endre translasjon og orienteringen mellom kameraene. Dette gir også store muligheter til å eksperimentere med forskjellig translasjon og orientering mellom de to posisjonene som bildene blir tatt i. Et lite problem med løsningen er at en antar at scenen er uforandret fra tiden det tar å ta første bildet, flytte kamera for så å ta det andre bildet(normalt ca. 10sekunder).

4.2 Feilmargin

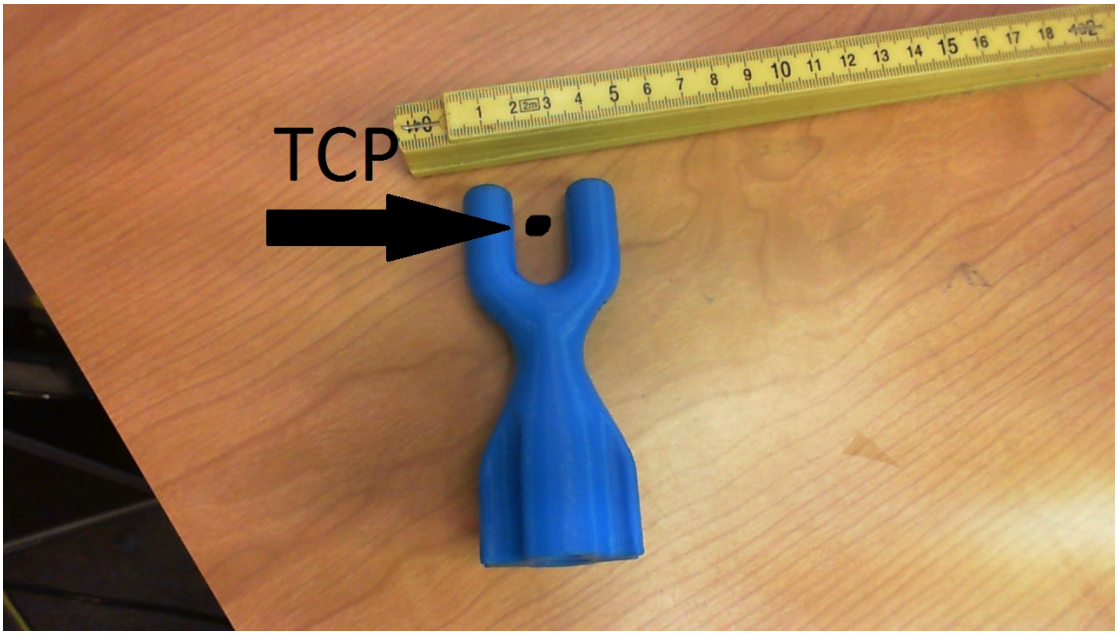
I 2.5.4 ble rammeverket for beregning av unøyaktigheter i estimeringen utredet. Gjennomsnittsavstanden til punktbanen er beregnet til $\hat{Z} \approx 0.8m$ for et tilfelle. Disparitet har tydeligere blitt regnet ut og verifisert til ca. $50pixel$. Man må derfor kunne klare å si noe om σ_d for å bergene teoretisk nøyaktighet. Sistnevnte parameter kan være litt vanskelig.

Ettersom σ_d i stor grad er avhengig av nøyaktigheten til canny-hjørnedeteksjonen tar jeg utgangspunkt i at gjennomsnittsverdien til hvor godt canny treffer. Dette har blitt estimert til 0.2. Fra 2.5.4 finner en dermed at $\sigma_d = \sqrt{0.2^2 + 0.2^2} = 0.63$. Legger en inn denne dataen i ligningen 2.5.4 får man

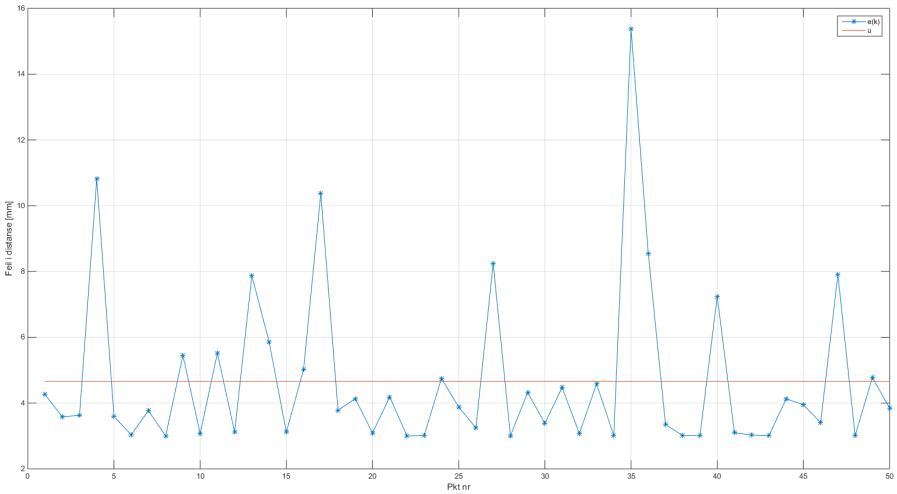
$$\sigma_Z = \hat{Z}\frac{\sigma_d}{d} \quad \Rightarrow \quad \sigma_Z = 0.8m\frac{0.63pix}{45pix} = 11[mm] \tag{4.1}$$

4.2.1 Målt feilmaring

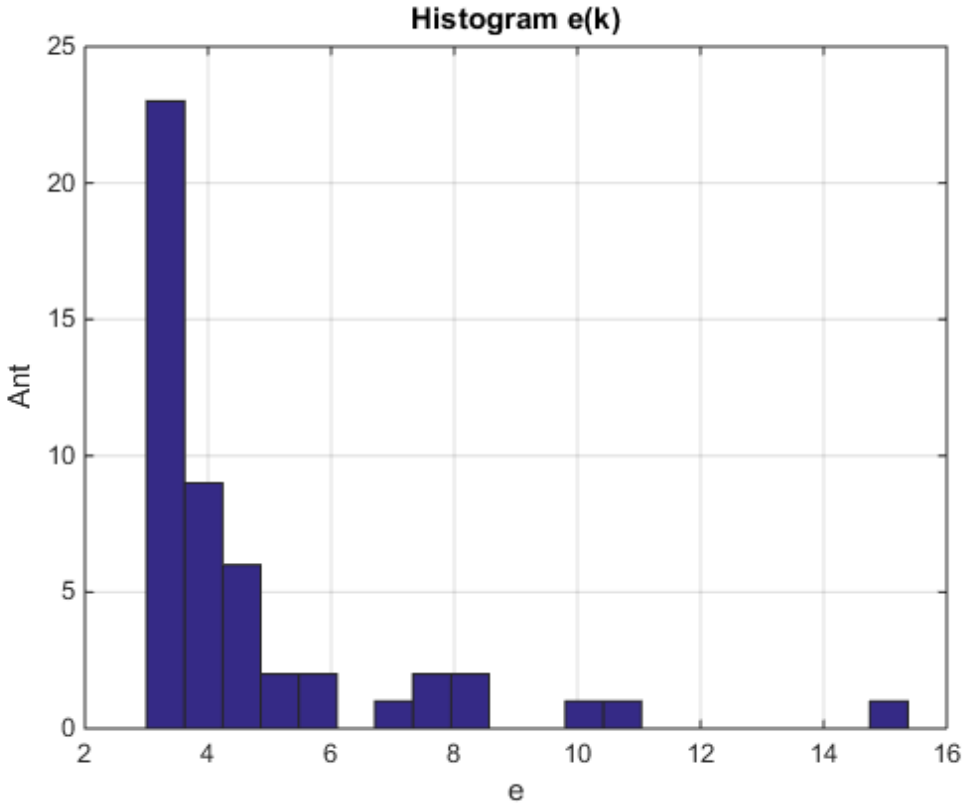
Målingene blir gjort ved å kjøre roboten til første punkt i settet for så å måle avstanden (målt i mm) ved hjelp av skyvelære. Avstanden som ble målt er distansen fra hvor roboten tror banen er, til hvor den egentlig er. Målingene er ikke hundre prosent pålitelige ettersom de er gjort i løse luften, hvor det er vanskelig å få vinkler korrekt i forhold til Tool Centure Point (TCP). Målingene som har blitt gjort ser ut til å stemme bra med estimeringer i 4.2.



Figur 4.1: TCP punktet er plassert 17.5 cm. langs tool0 sin Z-akse.



Figur 4.2: Distansen mellom reelt og estimert punkt



Figur 4.3: Histogram til målt avvik, søylene er i 1$[mm]$ inkrement.

$\hat{\mu}$	$4.6[mm]$	Estimert gjennomsnittlig avvik
$max\ avvik$	$15.4[mm]$	Max avvik

4.3 Kommentarer til implementering

Måten som implementeringen er gjort på legger til grunn at banen som ønskes løst må stå innenfor synsfeltet til kameraene. Ved å benytte seg av en datafil som blir overført til robotkontrollen kan en kjøre programmet separat uten bistand fra datamaskinen. Som nevnt i 3.3 har studentlisensen til MATLAB ikke "image acquisition toolbox". Dette har som følge at implementeringen ikke er så fullkommen som en skulle ønske.

4.4 Punktreduksjon

I oppgaven er det blitt sett på tre metoder som kan brukes til punktreduksjon. Hvor "No free lunch therom" kan trekkes frem. Hver metode har sine fordeler, men og sine ulemper.

Lineærpunktreduksjon er desidert den raskeste metoden, men har ulempen at en trenger et relativt høyt antall punkter for å kunne få en presis representasjon av banen. Dette gir robotkontrollen liten innsikt i fremtidige punkter og gir lav hastighet på utførelsen.

K-means har vist seg å være ganske tidskrevende når en ønsker mer enn 50 punkter. Punktene har blitt plukket ut noe smartere i den forstand at områder med en overvekt av punkter blir høyere prioritert. Punkttettheten er som regel høy i krumning enn langs en linje. Gjennom om en del eksperimenter ser det ut som k-means gjør en overraske god jobb, men blir fort tidkrevende når en ønsker et høyt antall punkter som resultat.

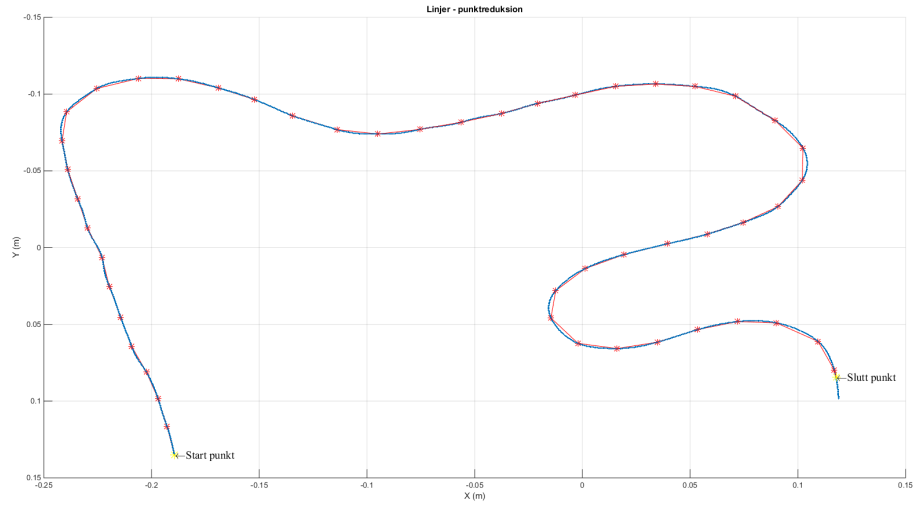
Punktreduksjon med volum-transformen er den mest tidskrevende, mye grunnet at den regner ut avstanden som en funksjon av roten[1]. Tester som har blitt gjort viser at en kan benytte seg av en ca. en 1/3 av punktene i forhold til lineær reduksjon. I sammenligningen av lineær- og volumpunktreduksjon vil en se at områder på banen hvor endringen er stor blir det liggende flere punkter. Der hvor endringene er mindre, blir det liggende mindre punkter. Isteden for den lineære som ikke tar hensyn til banens utforming.

På neste side kommer en tabell som viser tidsforbruket til de forskjellige punktreduserings teknikkene. Samt grafer som viser de forskjellige punktreduseringsteknikkene anvendt på data fra prosjektet.

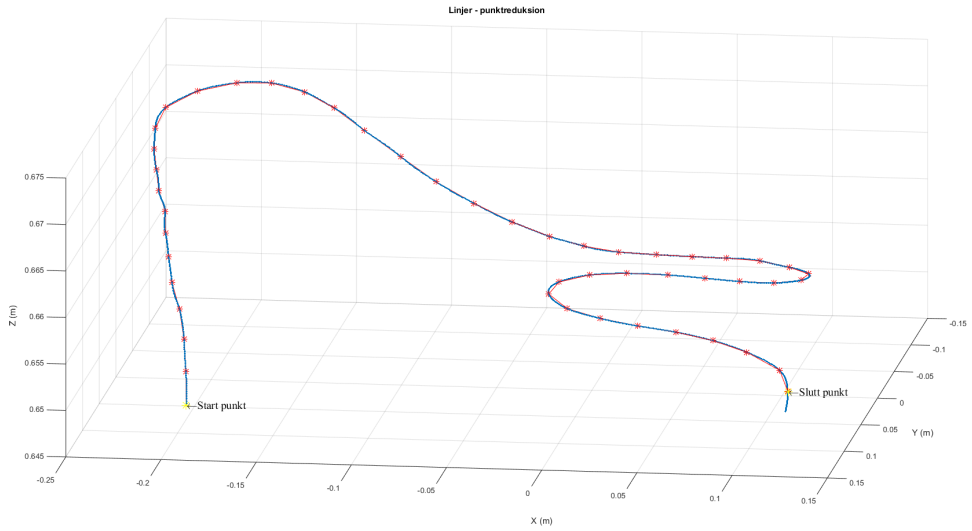
[1]Funksjoner som inneholder å ta roten er ganske tunge regneprosesser.

Punktreduksjons metode	Tidsbruk[s]	Antall punkter
Lineær	0.02	50*punkter*
Lineær	0.02	270*punkter*
Kmean	4.20	50*punkter*
Kmean	20.43	270*punkter*
Volum	29.89	50*punkter*
Volum	28.0	270*punkter*

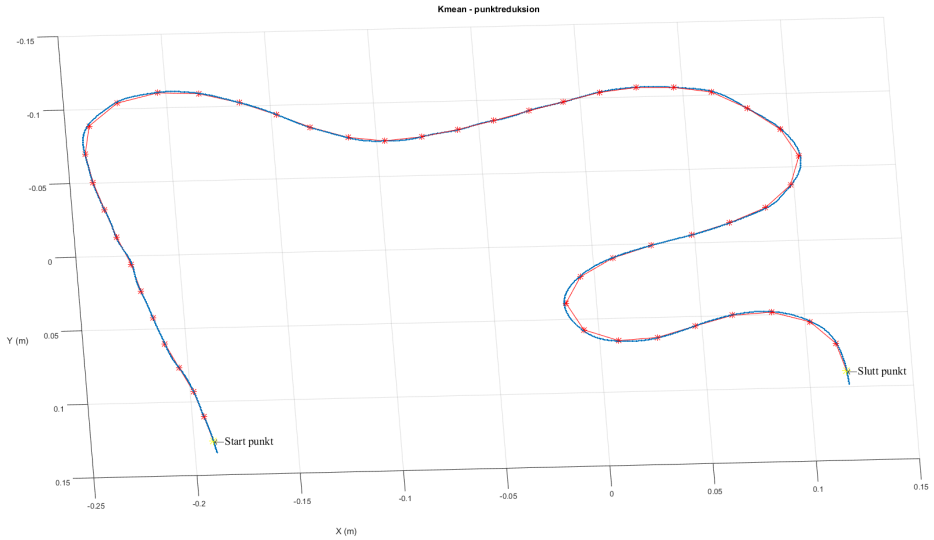
Figur 4.4: Tiden er målt som gjennomsnittet av ti tidsmålinger, med relativ liten varians.



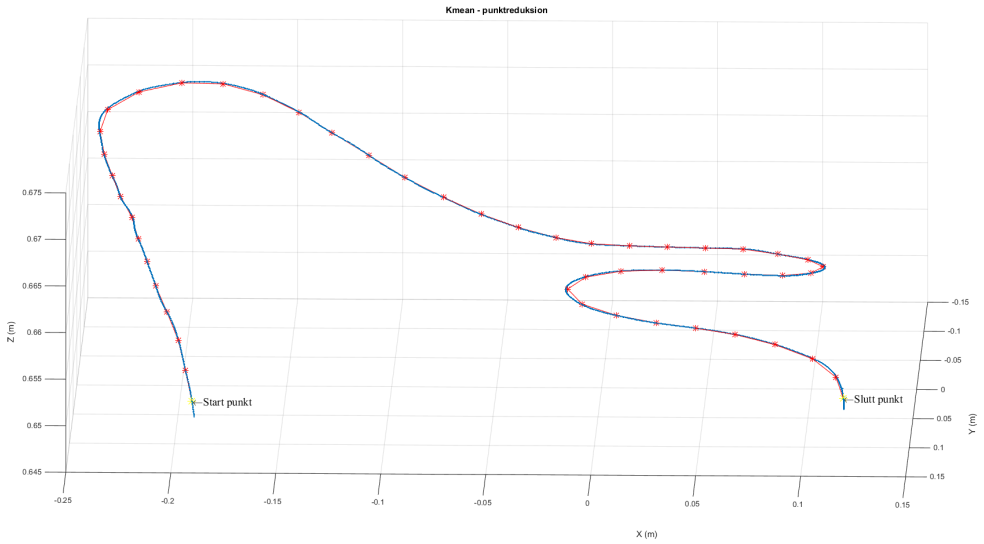
Figur 4.5: Lineær punktredusjon på virklig datasett. Original, Behandlet



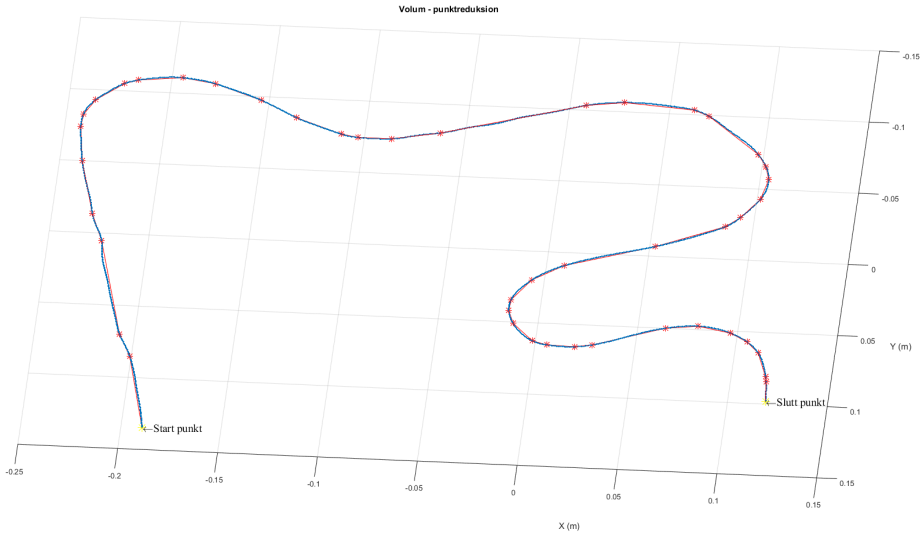
Figur 4.6: Lineær punktredusjon på virklig datasett, annen synsvinkel. Original, Behandlet



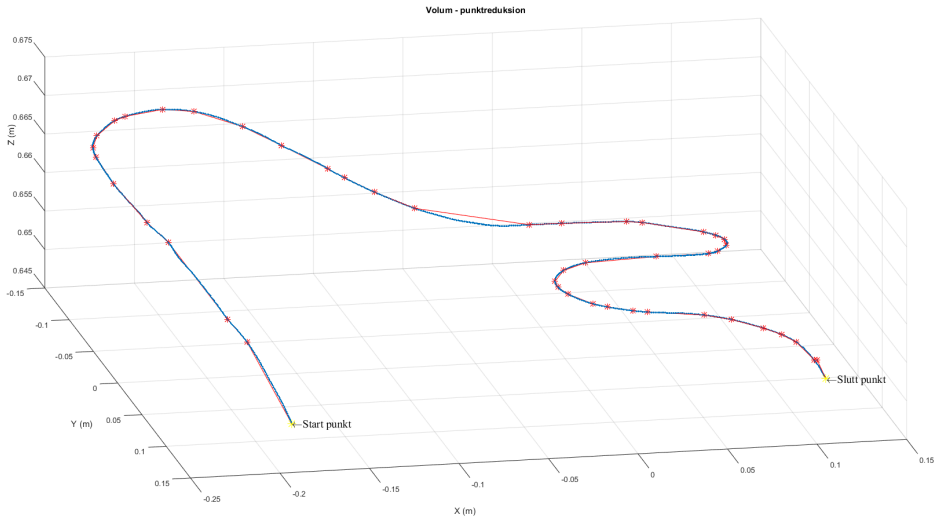
Figur 4.7: K-mean punktredusjon på virklig datasett. Original, Behandlet



Figur 4.8: K-mean punktredusjon på virklig datasett, annen synsvinkel. Original, Behandlet



Figur 4.9: Volum-punktredusjon på virklig datasett. Orginal, Behandlet



Figur 4.10: Volum-punktredusjon på virklig datasett, annen synsvinkel. Original, Behandlet

4.5 Filmklipp

Det er laget en filmsnutt som viser resultatet fra de to testbanene; `https://youtube.com/watch?v=V3f_y7ZAo2w`.[2] Klippet viser første testoppsett nr. 1 sammen med testverktøyet "gaffelen" etterfulgt av roboten som går til de to posisjonene for å ta bildesettet. Biten hvor bildene blir behandlet i MATLAB er ikke med i klippet. Jeg velger "Kjør bane" på FlexPandenten og den tilhørende rapid-koden starter.

[2] For en eventuell "papir utgave" kan en bruke linken : `http://goo.gl/IH9UA5`.

5

Konklusjon og videre arbeid

I denne oppgaven har jeg sett på muligheten til å benytte meg av en stereokamerarigg til forenklet robotprogramming. Hovedkonklusjonen er at oppgaven er løst og fungerer som ønsket. Med en gjennomsnittlig nøyaktighet på ca. 1cm syns jeg løsningen gir gode resultater som tangerer usikkerheten som ble beregnet i teoridelen.

MATLAB-implementasjonen er robust og finner et godt estimat av banen på første forsøk i testene som har blitt gjort. Tidsforbruket er noe i overkant stort dersom man velger en annen punktreduksjon enn lineær.

Tre metoder har blitt utforsket for å redusere antall punkter. Hvor alle har fordeler og ulemper. Min oppfatning er at punktreduksjon ved hjelp av volumtransformen gir best resultat til tross for sin lange kalkuleringstid.

Testverktøy og testbane som har blitt utviklet til testing av metoder gjennom prosjektet har vært til stor hjelp. Samt det gir stor mestringsfølelse når en har kommet i mål med teori og implementering. Banen ettersom banen er laget i et 5mm kobberrør har en muligheten til å teste flere baner ved å bøye det til fra hånd. Som gir store muligheter for å teste forskjellige senarioer.

Det 3D-printede testverktøyet "gaffelen" har noen riller (bilde 3.4f) i seg hvor tanken er at en skal kunne implantere en kollisjonsdetektor ved at en får en lukket krets dersom man er borti. Implementering av detektoren gir muligheten til la roboten lære banen i en form for tilfeldig gange (eng.:Random walk). Hvor en kan klassifisere et steg som bra hvis det ikke gir en lukket krets. Det motsatt blir også sant ved å lukke kretsen blir steget klassifisert som dårlig.

Det kunne også vært interessant å se på hva som skjer med nøyaktigheten hvis en in-

troduserer enda et kamera. Hvor forhåpentligvis nøyaktigheten blir enda bedre. Men for oppgavens formål ser det ut som to kamera fungere.

En tidligere student har lagt mye arbeid i å prøve å finne noen gode parametere som beskriver kameraets orientering i forhold til tool0. Det hadde vært greit og sett på hvordan en kan automatisere denne prosessen i håp om å få bedre presisjon og fleksibilitet. Samtidig er jeg er sikker på at flere vil støte på denne problemstillingen i fremtidige oppgaver.

Bibliografi

[1] Rgb color model @ONLINE.
URL http://en.wikipedia.org/wiki/RGB_color_model

[2] D. A. Forsyth, J. Ponce, Computer Vision - A modern approach - Secong Edition, PEARSON, 2012.

[3] I. Austvoll, Robot vision (machine vision).
URL http://www.ux.uis.no/~lovland/abb/robotvision_2013.pdf

[4] M. W. Spong, S. Hutchinson, M. Vidyasagar, Robot modeling and control, John Wiley & Sons, Hoboken (N.J.), 2006.
URL http://opac.inria.fr/record=b1119287

[5] 3dgep@ONLINE.
URL http://3dgep.com/understanding-quaternions/

[6] Kvaternion @ONLINE.
URL http://nn.wikipedia.org/wiki/Kvaternion

[7] Conversion between quaternions and euler angles @ONLINE.
URL http://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles

[8] Z. Zhang, A flexible new technique for camera calibration, IEEE Trans. Pattern Anal. Mach. Intell. 22 (11) (2000) 1330–1334.
URL http://dx.doi.org/10.1109/34.888718

[9] J. Heikkila, O. Silven, A four-step camera calibration procedure with implicit image correction, in: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), CVPR '97, IEEE Computer Society, Washington, DC, USA, 1997, pp. 1106–.
URL http://dl.acm.org/citation.cfm?id=794189.794489

[10] Stereo calibration using the stereo camera calibrator app @ONLINE.
URL `http://www.mathworks.se/help/vision/ug/stereo-camera-calibrator-app.html`

[11] G. Bradski, A. Kaehler, Learning OpenCV : Computer Vision with the OpenCV Library, O'Reilly Media,INC, 2008.

[12] C. Loop, Z. Zhang, Computing rectifying homographies for stereo vision.
URL `http://research.microsoft.com/en-us/um/people/zhang/Papers/TR99-21.pdf`

[13] K. Skretting, Ele620 systemidentifikasjon, 2014, notat-4.
URL `http://www.ux.uis.no/~karlsk/ELE620/notat4.pdf`

[14] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification (2Nd Edition), Wiley-Interscience, 2000.

[15] Art of problem solving @ONLINE.
URL `http://www.artofproblemsolving.com/Wiki/index.php/Shoelace_Theorem`

[16] Wikipedia@ONLINE.
URL `http://en.wikipedia.org/wiki/Shoelace_formula`

[17] L. Lorentzen, A. Hole, T. Lindstrøm, Kalkulus - med én og flere variable, 3 opplag 2006, Universitetsforlaget, 2006.

[18] M. Tykhelle, Identifisere objekter på et bord ved hjelp av bildebehandling i matlab slik at de kan plukkes opp av robot.
URL `http://hdl.handle.net/11250/221479`

[19] D. A. Forsyth, J. Ponce, Computer Vision: A Modern Approach - Second Edition, PEARSON, 2012.

[20] J. B. Løvland, P. Leupi, 3d scanning med rudolf, implementert med matlab & rapid ,ele 520, Tech. rep., Uuniversitet i Stavanger v/IDE (2013).
URL `http://www.ux.uis.no/~lovland/abb/rapp.pdf`

[21] Morfologiske operasjoner på binære bilder @ONLINE.
URL `http://www.mathworks.se/help/images/ref/bwmorph.html`

[22] S. Huihuang, B. He, Stereo rectification of calibrated image pairs based on geometric transformation, mecs-press.
URL `http://www.mecs-press.org/ijmecs/ijmecs-v3-n4/IJMECS-V3-N4-3.pdf`

Forkortelser

ABB - **A**sea **B**rown **B**overi. Internasjonalt selskap med hovedkontor i Zürich.

AT - **A**real **t**ransform

CV - **C**omputer **v**ision.

EOF - **E**nd **O**f **F**ile

FOV - **F**ield **o**f **v**iew

FTP - **F**ile **T**ransfer **P**rotocol

IDE - **I**nstitutt for **D**ata- og **E**lektroteknikk

IDS - **I**maging **D**evelopment **S**ystems. Tysk skelskap som utvikler kamera.

mhp - **m**ed **h**ensyn **p**å

NAT - **N**ormalisert **A**real **t**ransform

RGB - **R**ed,**G**reen,**Blue**

SAD - **S**um of **a**bsolute **d**ifferences

SE - **S**qurred **E**rror

SO - **S**pecial **O**rthogonal group

TCP - **Tool Center Point**

TR - **T**ranslation **R**otation matrix

UiS - **U**niversitet i **S**tavanger

μ**P** - Microprocessor

A

Appendix - MATLAB

StereoTestv1

```
%% clear all
clear all


%% Last inn camera parametere
load sP
sP = stereoParameters(sP.CameraParameters1,sP.CameraParameters1,...
    sP.RotationOfCamera2,sP.TranslationOfCamera2);

%% Last inn bilder
Img1 = imread('Basis_3.png');
Img2 = imread('20mmx_3.png');
% Vis bilder
imshowpair(Img1,Img2,'montage')


%% Rectify Stereo Images
[Img1Rect,Img2Rect]= rectifyStereoImages(Img1,Img2,sP);
imshowpair(Img1Rect,Img2Rect)
%% Disparity Map, bergnet at d burde bli ca 45, gir meg selv ca 1.5x ...
    margin.
disparityMap = disparity(rgb2gray(Img1Rect), rgb2gray(Img2Rect), ...
    'BlockSize',9,'DisparityRange', [0 80], 'Method','SemiGlobal');

marker_idx = (disparityMap == -realmax('single'));
disparityMap(marker_idx) = min(disparityMap(~marker_idx));

% Filtrer disparitymap
disparityMap = medfilt2(disparityMap,[5 5]);
```

```
29
30 imshow(disparityMap,[min(disparityMap(:)), max(disparityMap(:))])
31 colormap(jet(80)),colorbar
32
33 %% Point Cloud
34 pC = reconstructScene(disparityMap,sP)/1000;
35 z = pC(:,:,3);
36
37 z(z < 0 | z > 1) = NaN;
38 pC(:,:,3) = z;
39 showPointCloud(pC,Img1Rect),axis equal
40 xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)')
41
42 %% Kast punkter i bilde som er lenger borte en 1 meter
43 mask = repmat(z > 0 & z < 1, [1, 1, 3]);
44 Img1Rect(~mask) = mean(Img1Rect(:));
45 imshow(Img1Rect)
46
47 %% Plukk ut intresange punkter fra pC, ved brukt av bakgrunns supp
48 BW = ~im2bw(rgb2gray(Img1Rect),graythresh(rgb2gray(Img1Rect)));
49 BW = bwareaopen(BW, 2000); % Kast pixelobjekter mindre en 2000.
50 % Finner soorste objekt
51 stat = regionprops(BW,'Centroid','Area','PixelIdxList');
52 [~,idx] = max([stat.Area]);
53 BW = zeros(size(BW));
54 BW(stat(idx).PixelIdxList) = 1;
55 % Morph til skel
56 BW = bwmorph(BW,'skel',Inf);
57 % fill
58 BW = imfill(BW,'holes');
59 % Morph til skel
60 BW = bwmorph(BW,'skel',Inf);
61
62 % Morph spur, fjerning av "haar" som sitter igjen paa skel.
63 BW = bwmorph(BW,'spur',50);
64 disp(['Det er ' num2str(length(find(BW==1))) ' intresange punkter i BW'])
65 BW = bwmorph(BW,'clean',Inf);
66
67 imshow(BW)
68
69 %% Plukk ut skel fra punkt skyen
70 z = pC(:,:,3);
71 z(~BW) = NaN;
72 pC(:,:,3) = z;
73
74 showPointCloud(pC,Img1Rect),axis equal
75 xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)')
76
77 %% Plukk ut punker, og i XYZ kordiantsystem
78 x = pC(:,:,1);
79 y = pC(:,:,2);
80 z = pC(:,:,3);
81
```

```
82  wP = double([x(BW),y(BW),z(BW)]);
83
84  % Kast wP med NaN
85  wP(isnan(wP(:,3)),:) = [];
86
87  % Vis data og velg startpunkt paa banen.
88  fig = figure;
89  plot3(wP(:,1),wP(:,2),wP(:,3),'*','MarkerSize',1);,grid on,
90  camorbit(0, -125, 'camera', [0 1 0]),
91  xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)')
92  title('Velg start punkt m/ Data Curser, "enter" naar du har funnet ...
        punkte.')
93
94  % Hent info fra data curser.
95  dcm_obj = datacursormode(fig);
96  set(dcm_obj,'DisplayStyle','datatip',...
97      'SnapToDataVertex','off','Enable','on')
98  pause
99  c_info = getCursorInfo(dcm_obj);
100 close(fig)
101
102 % Start punkt XYZ
103 startP = c_info.Position;
104 disp(['Start punkt er : ' num2str(startP)])
105 %% Trenger og sotere datasett, staar i startpunkt wP(1,:) vil naa ...
        finne ...
106 % wP(x,:) som gjor at D = wP(1,:).^2 + wP(x,:).^2 er mist mulig.
107 % Nested loop har Big O(n^2),
108 P = startP;
109 wPsort = nan(size(wP));
110 Pr = repmat(P,[length(wP),1]);
111 g = waitbar(0,'Sortering paa gaar');
112 l = length(wP);
113 for i=1:size(wP,1)
114     D = nan(size(wP,1),1); %Dist vektor.
115     for k=1:size(wP,1)
116        D(k) = sum((P-wP(k,:)).^2); %Regn it sq eq dist mellom et punkt ...
               og alle andre.
117     end
118     [~,idx] = min(D); % Finn minste dist
119
120     P = wP(idx,:); %Plukk ut
121     wPsort(i,:) = P; %Legg inn
122     wP(idx,:) = NaN; %Sett gamle punkt til nan.
123     waitbar(i/l) % Oppdater statusbar
124 end
125 close(g)
126
127
128
129 %% Distanse testing, regen ut D mellom punktene.
130 D = zeros(1,length(wPsort));
131 for i=1:length(wPsort)-1
```

```
132     D(i) = (wPsort(i+1,:)-wPsort(i,:))*(wPsort(i+1,:)-wPsort(i,:))';
133 end
134 idx = find(std(D) < D);
135 idx = min(idx);
136 disp(['Antall kastet : ' num2str(length(wPsort) - idx)])
137 wPsort(idx:end,:) = [];
138 plot(1:length(D),D,idx,D(idx),'r*','MarkerSize',10),grid on
139 text(idx,D(idx),'\leftarrow Start kast.')
140
141 %% Visualise punktskyen
142 plot3(wPsort(:,1),wPsort(:,2),wPsort(:,3),'b-*','MarkerSize',1)
143 xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)')
144 grid on,set(gca,'YDir','reverse')
145
146 %% Glatting
147 plot3(wPsort(:,1),wPsort(:,2),wPsort(:,3),'b-*','MarkerSize',1),hold on
148 r =[10, 10,100];
149 for i = 1 : 3
150     wPsort(:,i) = poly1LMS(wPsort(:,i)',r(i),40)';
151 end
152
153 plot3(wPsort(:,1),wPsort(:,2),wPsort(:,3),'r-*','MarkerSize',1)
154 xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)')
155 grid on,set(gca,'YDir','reverse'),legend('Orginal','Glattet'),hold off
156 %% Punktreduksjon
157 nywP = punktRed(wPsort,'vol',60,startP);
158 figure(1),clf
159 plot3(nywP(:,1),nywP(:,2),nywP(:,3),'r-*','MarkerSize',8),hold on
160 plot3(wPsort(:,1),wPsort(:,2),wPsort(:,3),'*','MarkerSize',1)
161 plot3(nywP([1 end],1),nywP([1 end],2),nywP([1 end],3),...
162     'y*','MarkerSize',10),grid on
163
164 %Legg paa tekst
165 text(nywP(1,1),nywP(1,2),nywP(1,3),...
166 '\leftarrow\fontname{times}Start punkt','FontSize',14)
167 text(nywP(end,1),nywP(end,2),nywP(end,3),...
168 '\leftarrow\fontname{times}Slutt punkt','FontSize',14)
169
170 %Label og akser
171 xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)')
172 grid on,set(gca,'YDir','reverse')
173
174 %% Ut fra plottet over ser en at xy-planet kan brukes til og finne vinkel
175 % toolen maa ha for faa rett orientering iforhold til. Vinkel en ...
      finner er
176 % gitt som forholde mellom p(i,:) og p(i+1,:). Hvor en igjen kun er
177 % intresert i foorste og andre element i p.
178
179 deltax = diff(nywP(:,1)); %Endring i x
180 deltay = diff(nywP(:,2)); %Endring i y
181
182 theta = atan2d(deltay,deltax); % Finn vinkel
183 figure(1),clf
```

```
184 plot(theta,'*-'),hold on, %Orignal
185
186 theta = medfilt1(theta,5); %Median filtering.
187 % theta = smooth(theta,'lowess'); % 1-orden tilnerming mellem punkter,lms.
188 theta = poly1LMS(theta',5,50)';
189 theta = [theta; theta(end)]; % Diff fjener ett punkt, antar at to ...
        siste punkt er like.
190 % theta = round(theta-90); % Rotajon i forhold til kamera og tool0.
191 theta = round(theta+90);
192 plot(theta,'*-') % Behandlet theta
193 grid on,legend('Orignal','Behandlet','Location','northwest')
194
195 title('Vinkel tool - xy'),ylabel('Grader'),xlabel('pkt nr')
196
197 %% Her blir punktene fra wPny tilpasset det nye kordinatsystemet ...
        antatt i tool0.
198
199 % Rotasion om z.
200 R = @(theta)    [cos(theta) -sin(theta) 0;
201                  sin(theta) cos(theta) 0;
202                      0           0       1];
203
204 t = [0 0 0]; % Avstand fra avstand mellom tool0 og flyttet  kamera kord
205 RT = [R(pi/2) t'; 0 0 0 1]
206 pNy = nan(size(nywP)+1);
207
208 % Nytt punkt blir naa pNy = RT * P
209 for i = 1:length(nywP)
210     pNy(i,:) = RT*[nywP(i,:) 1]';
211 end
212
213 % Rydd opp, gjort om fra homogene kord, fra size + 1.
214 pNy(:,4) = [];
215 pNy(end,:) = [];
216
217 pNy = round(pNy*1000); % Gjor om til mm, rund av til nermeste.
218 % Skriver alle datapunktene til fileName
219 fileName  = 'XYZ.txt'; %Navn paa filen
220 fileID = fopen(fileName,'w'); %Gi matlab retihgeter til og skrive
221 fprintf(fileID,'%6.4f:%6.4f:%6.4f:%6.4f\r\n',[pNy,theta]');  %Skriv
222 fclose(fileID); %Lukk skrving
223
224 % % Last opp til robot v/FTP
225 ip = '152.94.0.38'; %IPen til robot
226 usrName = 'nordert'; %Brukernavn
227 pw = 'ide'; % Passord,  ikke idelt og lagre "husj husj".
228
229 ftpCli = ftp(ip,usrName,pw); %Opprett FTP objekt og koble til
230
231 try
232     mput(ftpCli,fileName); %Prov laste opp filen
233 catch
234     disp('Noe gikk galt'); %Noe gill kalt melding
```

```
235 end
236 close(ftpCli) %Avslutt FTP client
237
238 disp(['Opplastning ferdig, start rapid kode'])
```

punktRed

```
1  function nywP = punktRed(wP,metode,N,startP)
2  % Denne funksjonen gir brukeren muligheten til og velge mellom ...
       forskjellige
3  % maater og nedsample wP.
4  % Bruk wP(wP,'metode',N)
5  % wP -> Punktsky
6  % Mulige metoder:
7  %                   Linjer - "Lin"
8  %                   Kmens - "Kmean"
9  %                   Volemetrisk - "Vol"
10 %                   Vinkel - "ang"
11 % N -> Antall punkter en oonsker etter nedsampling. N < lengeth(wP)
12
13 if(N >= length(wP))
14     error('Foolgene maa vaare sant : N < lengeth(wP)')
15 end
16
17 switch lower(metode)
18     case 'lin'
19         nywP = linSample(wP,N);
20         nywP = cleanUpPoints(nywP);
21     case 'kmean'
22         nywP = kmeanSample(wP,N);
23         nywP = sorteqDist(nywP,startP);
24         nywP = cleanUpPoints(nywP);
25     case 'vol'
26         nywP = volSample(wP,N);
27         nywP = sorteqDist(nywP,startP);
28     otherwise
29         error('Ikke noe gyldig arguemt, se : help punktRed')
30 end
31
32 % Sjekk punkter
33
34
35 end
36
37 %% Pluker ut punkter linjert gitt av linspace.
38 function nywP = linSample(wP,N)
39 nywP = wP(round(linspace(1,length(wP),N)),:); % Plukk ut gitt av ...
40 % linspace
41 % nywP = cleanUpPoints(nywP); % Fjern daarlige punkter.
42 % nywP = sorteqDist(nywP,startP);
```

```
43 % nywP = [wP(1,:) ; nywP ; wP(end,:)];
44 end
45
46 %% Segmenterer wP til N clustere, er ganske regne krevende.
47 function nywP = kmeanSample(wP,N)
48
49 [~,nywP] = kmeans(wP,N,'Replicates',25);
50 % Punktene kan i noentiller ha ett senver av tilfeldinghet, trenger ...
51 % opprydning.
52 % nywP = sorteqDist(nywP,startP);
53 end
54
55 %% Proover og fjerne sampler som ligger paa en linje mellom to punkter,
56 % p1,p2,p3 vil gi volum = 0, om p2 ligger paa en linje mellom p1 og p3.
57 function wP = volSample(wP,N)
58 N = N - 2; % Legger til slutt og start punkt, trekker fra 2 punkter.
59 ssP = [wP(1,:) ; wP(end,:)];
60 i = 0;
61 kast = 0;
62 while (length(wP) >= N)
63     v = volCalc(wP); % Regen ut volum transformasjon
64     [~,idx] = min(v); % Finn minste vinkel
65     idx = idx + 1;
66
67     if(~isempty(idx))
68         wP(idx,:) = []; % Kast punkter.
69     end
70     i = i + 1;
71 end
72
73 disp(['Antall iter / kastet punkter : ' ...
74     num2str(i) ' / ' num2str(kast)])
75
76 if(wP(1,:) ~= ssP(1,:))
77     wP = [ssP(1,:) ; wP];
78 end
79
80 if(wP(end,:) ~= ssP(end,:))
81     wP = [wP ; ssP(end,:)];
82 end
83 end
84
85 %% Fjener punkter som ligger lengere borte fra vedrandre enn std.
86 function nywP = cleanUpPoints(nywP)
87 D = zeros(1,length(nywP));
88 for i=1:length(nywP)-1
89     D(i) = (nywP(i+1,:)-nywP(i,:))*(nywP(i+1,:)-nywP(i,:))';
90 end
91 if(std(D) > 1/10000)
92     idx = find(std(D) < D);
93     idx = min(idx);
94     disp(['Antall kastet : ' num2str(length(nywP) - idx)])
95     if(~isempty(idx))
```

```
96          nywP(idx:end,:) = [];
97      end
98  end
99  % plot(1:length(D),D,idx,D(idx),'r*','MarkerSize',10),grid on
100 % text(idx,D(idx),'\leftarrow Start kast.')
101 end
102
103 %% Sorter etter eqDist
104 function wPsort = sorteqDist(wP,startP)
105 P = startP;
106 wPsort = nan(size(wP));
107 for i=1:size(wP,1)
108     D = nan(size(wP,1),1); %Dist vektor.
109     for k=1:size(wP,1)
110         D(k) = sum((P-wP(k,:)).^2); %Regn it sq eq dist mellom et ...
                punkt og alle andre.
111     end
112     [~,idx] = min(D); % Finn minste dist
113     P = wP(idx,:); %Plukk ut
114     wPsort(i,:) = P; %Legg inn
115     wP(idx,:) = NaN; %Sett gamle punkt til nan.
116 end
117
118
119 end
120
121 %% Finner volume gitt av P(i-1,:),P(i,:) og P(i+1,:)
122 function v = volCalc(P)
123 % pDist = @(p1,p2) sqrt((p1(1)-p2(1))^2+(p1(2)-p2(2))^2+(p1(3)-p2(3))^2);
124 pDist = @(p1,p2) sqrt(sum((p1-p2).^2));
125 v = zeros(length(P)-2,1);
126
127 for i = 2 : length(P)-1
128     dist = 6*pDist(P(i-1,:),P(i+1,:));
129     d = det([P(i-1,:) ; P(i,:) ; P(i+1,:)]);
130     v(i-1) = abs(d)/dist;
131 end
132
133 end
```

poly1LMS

```
1  function y = poly1LMS(data,N,itr)
2  %% poly1LMS tar inn datavektoren data og tar N punkter hvor en bruker LMS
3  % til og estimere kurvden y = ax+b. LMS brukes til og finne kefistientene
4  % a og b. Etter at a og b er funnet blir kurven beregnet ut fra ...
       estimat og
5  % lagt tilbake i datasettet. Hvor en velger og ta gjennomsnittet av gammel
6  % og nytt estimat. Itr oppgir hvor mange ganger datasettet skal kjoores
7  % igjennom med tilnermingen.
```

```
% N maa vEre et positivlt oddetall.

N = (N-1)/2;
for k = 1:itr
    for i = N+1 : length(data) - (N+1)
        Y = data(i-N:i+N);
        x = linspace(data(i-N),data(i+N),2*N+1);
        phi = [x',ones(N*2+1,1)];
        theta = phi\Y';
        data(i-N:i+N) = ...
            (theta(1)*x+theta(2)*ones(1,2*N+1)+data(i-N:i+N))/2;
    end
end
y = data;
end
```

dispDemo

```
load sP
I = (imread('20mmx.png'));
I2 = (imread('Basis.png'));
[I,I2] = rectifyStereoImages(I,I2,sP);

%% Disp demo
for d = -5:5:128
    Idiff = abs(I(:,6:end-128,:) - I2(:,d+6:d+end-128,:));
    imshow(Idiff,[]),title(['d = ' num2str(d)])
    text(0,60,['d = ' num2str(d)],'FontSize',120,'Color','w')
    waitforbuttonpress
end
```

meshDemo

```
%% Last inn camera parametere
load sP
Img1 = imread('Basis.png'); %Last inn bilde
Img2 = imread('20mmx.png'); %Last inn bilde
[Img1Rect,Img2Rect]= rectifyStereoImages(Img1,Img2,sP); %rectify bildene
%% Disparity Map, bergnet at d burde bli ca 45, gir meg selv ca 1.5x ...
    margin.
% denne cellen er kopiert rett ut fra oppgaven.
disparityMap = disparity(rgb2gray(Img1Rect), rgb2gray(Img2Rect), ...
    'BlockSize',9,'DisparityRange', [0 80], 'Method','SemiGlobal');

marker_idx = (disparityMap == -realmax('single'));
disparityMap(marker_idx) = min(disparityMap(~marker_idx));
```

```
13
14 % Filtrer disparitymap
15 disparityMap = medfilt2(disparityMap,[5 5]);
16
17 imshow(disparityMap,[min(disparityMap(:)), max(disparityMap(:))])
18 colormap(jet(80)),colorbar
19
20 %Dybden estimers ut fra disparityMap. Z = f*t/d.
21 Z = norm(sP.TranslationOfCamera2)*...
22     mean(sP.CameraParameters1.FocalLength)./disparityMap;
23 Z = Z./1000; %Konv fra mm til m.
24
25 Z(Z>1) = NaN; %Sett Z>1 til NaN.
26
27 %% Finn yter parametere.
28 Kinv = inv(sP.CameraParameters1.IntrinsicMatrix'); %inverter intene ...
       matrisen
29 Z0 = mean(Z(isfinite(Z))); % Weak perspectiv
30 xy = [Kinv*[0,0,1]',Kinv*[1280,720,1]']*Z0; %Randverdier
31
32 % X og Y mesh bergenes
33 x = linspace(xy(1,1),xy(1,2),size(disparityMap,1));
34 y = linspace(xy(2,1),xy(2,2),size(disparityMap,2));
35 [X,Y] = meshgrid(x,y);
36 %% Plot og legg paa farger fra Img1Rect
37 mesh(X',Y',Z,Img1Rect)
38 xlabel('X (m)'),ylabel('Y (m)'),zlabel('Z (m)')
39 grid on,set(gca,'YDir','reverse')
```

B

Appendix - Rapid

```
1  MODULE Module1
2      PERS tooldata tGaffel := [TRUE, [[0,0,150] ,[1,0,0,0]],[1, [0, 0, ...
           0.001],[1, 0, 0, 0], 0, 0, 0]];
3      ! PERS tooldata tGaffel := [TRUE, [[0,0,190] ,[1,0,0,0]],[1, [0, ...
           0, 0.001],[1, 0, 0, 0], 0, 0, 0]];
4      PERS wobjdata tWork :=[FALSE,TRUE, "", [ [500,50,150], ...
           [0.582387,0.332931,-0.612491,0.418136] ],[[0,0,0],[1,0,0,0]]];
5      PERS tooldata tCamGripper := [TRUE, [[53.69,0,41.19] ,[0.997969, ...
           0.00541994, 0.0171534, -0.0611147]],
6          [1, [0, 0, 0.001],[1, 0, 0, 0], 0, 0, 0]];
7      CONST num delta := 20; ! Konstant som beskiver forsyving mellom ...
           kamera1 og kamera2.
8      VAR robtarget p1 := ...
           [[0,0,0],[1,0,0,0],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
9      ! Punkt p1 brukes som utgankspunkt.
10     VAR robtarget p2; !Punkt2 brukes som varbiel.
11     VAR robtarget pStart;
12     VAR bool firstTime := TRUE; ! Flag som brukes til
13     CONST string fileName := "XYZ.txt"; !Navn paa oonsket fil som skal ...
           aapnes
14     VAR string temp; ! String som brukes til og informere bruker mha ...
           pendant.
15     VAR iodev infile; ! Objekt som haandterer IO fra fil.
16     VAR num x := 0; !x ! x-kordinat
17     VAR num y := 0; !z ! y-kordinat
18     VAR num z := 0; !y ! z-kordinat
19     VAR num theta := 0; !Rotasjon i xy-planet
20
21
22
23 PROC main()
24     VAR num answer;
```

```
25      TPReadFK answer, "Hva skal roboten gjoore?", "Ta bilde", stEmpty ...
            ,"Kjoor banen", stEmpty, stEmpty;
26      IF (answer = 1)  then
27          taBilder; ! Ta bilder
28      ELSE
29          lesFraFil; ! Les fra fil.
30      ENDIF
31  ENDPROC
32
33  PROC lesFraFil()
34      Close infile; ! Prov og lukk filen.
35      Open fileName,infile\Read; ! Les filen
36
37      p2 := p1;
38      WHILE TRUE DO
39
40      x := ReadNum(infile\Delim:=":"); ! Les inn x-kordinat
41      y := ReadNum(infile\Delim:=":"); ! Les inn y-kordinat
42      z := ReadNum(infile\Delim:=":"); ! Les inn z-kordinat
43      theta := ReadNum(infile\Delim:=":"); ! Less in rotasion.
44
45          IF x > EOF_NUM OR y > EOF_NUM OR z > EOF_NUM THEN ! Sjekk om ...
                end of file, hvis ja, avslutt.
46              MoveL Offs(p2,0,0,-150),v50,z1,tGaffel,\WObj:=tWork;
47              WaitUntil \InPos,TRUE;
48              MoveJ pStart,v100,z1,tGaffel,\WObj:=tWork;
49              TPWrite "Ferdig";
50              RETURN ;
51          ELSE
52                  temp := "x = " + NumToStr(x,1) + " y = " + ...
                        NumToStr(y,1) + " z = " +NumToStr(z,1) +
53                  " theta = " + NumToStr(theta,1);!Info string til bruker.
54                  TPWrite temp; !Skriv til pendant.
55                  p2.trans.x := x; !Legg x kodriant inni p2
56                  p2.trans.y := y; !Legg y kordinat inni p2
57                  p2.trans.z := z; !Legg z kordinat inni p2;
58                  p2.rot := OrientZYX(theta,0,0); !Orientering i xy-planet
59
60              IF(firstTime) THEN ! Legger ett punkt paa utsiden for og ...
                    unga kolosion med vektooy og bord.
61                  !p2.trans.z := z*2/3;
62                  !p2.trans.x := x*2/3;
63                  pStart := Offs(p2,50,0,-150);
64                  MoveJ pStart,v100,z1,tGaffel,\WObj:=tWork;
65                  firstTime := FALSE;
66                  !p2.trans.z := z;
67              ENDIF
68                  ConfL \Off;
69                  MoveL p2,v50,z1,tGaffel,\WObj:=tWork;
70          ENDIF
71
72      ENDWHILE
73      MoveL Offs(p2,0,0,-150),v50,z1,tGaffel,\WObj:=tWork;
```

```
74      WaitUntil \InPos,TRUE;
75      MoveJ pStart,v100,z1,tGaffel,\WObj:=tWork;
76      TPWrite "Ferdig";
77  ENDPROC
78
79  PROC taBilder()
80          MoveJ p1,v100,fine,tCamGripper,\WObj:=tWork; ! Venstre kamera.
81          WaitTime 5; !Vent 10s, saa en rekker og ta bilde.
82          MoveJ Offs(p1,0,delta,0),v100,fine,tCamGripper,\WObj:=tWork; ! ...
                Hooyre kamera.
83  ENDPROC
84
85  ENDMODULE
```