Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Informasjonsteknologi, automatisering og signalbehandling	Vårsemesteret, 2015 Åpen
Forfatter: Kristoffer Solheim	…… (signatur forfatter)
Fagansvarlig: Karl Skretting Veileder(e): Morten Mossige Karl Skretting	
Tittel på masteroppgaven: Komprimeringsalgoritme for generering av en robotbane til bruk i Simplified Robot Programming Engelsk tittel: A compression algorithm for generating a robot path to be used in Simplified Robot Programming	
Studiepoeng: 30	
Emneord: Simplified Robot Programming Algoritme Robotbane Vektet sum av kvadrert feil Minste kvadraters løsning Linjetilpassing	Sidetall: 104 + vedlegg/annet: Zip-fil med figurer og kildekode Stavanger, 15.06.2015

Masteroppgave - Algoritme for generering av en robotbane til bruk i Simplified robot programming

Kristoffer Solheim

15. juni 2015

Sammendrag

Oppgaven er et resultat av et samarbeid mellom Universitetet i Stavanger og ABB Robotics. Hensikten med oppgaven er å se om det er mulig å lage en bedre algoritme enn den som i dag ligger til grunn for ABBs teknologi, Simplified Robot Programming.
I denne oppgaven presenteres en algoritme for å tilpasse en bane til et et sett med rådata, samlet inn ved hjelp av en sensor som sporer posisjon og orientering i rommet. Målet med algoritmen er å tilpasse flere rette linjestykker til rådataen, slik at det returneres en komplett bane som en robot kan følge, kun ved bruk av lineære bevegelser. Banen skal kun bestå av slike bevegelser fordi en slik bevegelse er den enkleste av de to grunnleggende elementene som en robotbane bygges opp av. Sirkelbuer er det andre elementet, og er vanskeligere å implementere. Banen ønskes å få så god nøyaktighet som mulig, samtidig som den skal inneholde få punkter. Det introduseres ulike mål for hvor god en bane er, der punktreduksjon og avviket i posisjon, orientering og hastighet evalueres. I rapporten blir resultatet fra algoritmen sammenlignet med ABBs genererte baner, referansebanene. Resultatene viser at algoritmen presentert i denne rapporten kan gi et bedre resultat enn hva resultatet fra ABBs algoritme gir. Den muliggjør også å optimalisere banen for en bestemt egenskap, ved å tillate et større avvik i de andre egenskapene i banen. Ved å bruke algoritmen presentert i denne rapporten kan en bane representeres med færre punkter enn antall punkter i referansebanen og fremdeles ha mindre avvik enn referansebanen.

Forkortelser

- ABB - Asea Brown Boveri
- UiS - Universitetet i Stavanger
- SRP - Simplified Robot Programming
- TCP - Tool center point, midtpunktet i robotens verktøy
- SSE - Sum of squared error, sum av kvadrert feil
- MKM - Minste kvadraters metode
- LS - Least squares

Forord

Denne oppgaven er skrevet som en masteroppgave ved Universitetet i Stavanger, i samarbeid med ABB Robotics. Oppgaven ser på muligheten for å utvikle en ny og bedre algoritme som skal kunne brukes i deres teknologi, Simplified Robot Programming.

Jeg vil takke mine veiledere Karl Skretting, Morten Mossige og Ståle Freyer for hjelp, råd og motivasjon gjennom hele utførelsen av masteroppgaven min.
Jeg vil også takke alle de ansatte på ABB som har hjulpet meg med faglig støtte og nyttig informasjon om roboter generelt og SRP spesielt.
Til slutt vil jeg takke familie, venner og min samboer for tålmodigheten de har vist meg, både under utførelsen av masteroppgaven min og i studietiden generelt.

Innhold

1 Innledning

Målet med Simplified Robot Programming(SRP) er å forenkle programmeringen av roboter. Mennesker uten kjennskap til programmering skal kunne lage robotbaner raskt og enkelt, på en intuitiv måte. Dette sparer bedrifter for mye tid og penger i forbindelse med opplæring av personell, i tillegg til at tiden det tar å lage en bane blir vesentlig redusert. Ved å bruke en håndholdt sensor som sporer posisjon og orientering i rommet, er det mulig for et menneske å lære en robot hvordan den skal bevege seg ved å "vise" den bevegelsen. Med denne måten å programmere en robot på, er det også mulig å gjenskape en bevegelse med en flyt lik den som er i de menneskelige leddene. Dette kan være vanskelig og tidkrevende å gjøre dersom robotbanen programmeres manuelt i et dataprogram. Ulempen med en slik innspilling er at det returneres en bane som består av mange punkt, og flere av punktene er overflødige. Ved å programmere hvert punkt i banen for hånd, kan en enkel og god bane lages, men det tar lang tid. Hver punkt må defineres hver for seg, der både posisjon og orientering spesifiseres og hastighet på bevegelsen må bestemmes.
Denne rapporten tar for seg muligheten til å spille inn en bane ved hjelp av en sensor som sporer både posisjon og orientering i rommet med en høy samplerate. Det skal så tilpasses en kortere bane, kun bestående av lineære bevegelser til dataen som er spilt inn. Det vil si en bane som er så lik som mulig den originale banen, representert med kun en brøkdel av punktene, og med kun lineære bevegelser. Grunnen til å uttrykke banen med så få punkter som mulig er at det da enkel kan utføres endringer i banen gjennom et grafisk grensesnitt, etter at banen er generert. Dette vil resultere i en rask innspilling av en bane, samtidig som det returneres en enkel bane som består av få punkter, som om den skulle vært programmert for hånd.
ABB er samarbeidende bedrift for denne oppgaven og er firmaet som står bak denne teknologien. Firmaet har allerede utviklet en slik algotitme, men ønsker å se på muligheten for å gjøre det annerledes og bedre. Simplified Robot Programming ble introdusert av ABB som en revolusjon innen robotprogrammering.
Det kan også nevnes at det gjennom en samarbeidende oppgave blir sett på muligheten til å samle inn data ved hjelp av en *Microsoft kinect-sensor*, til bruk i Simplified Robot Programming. Det er en annen person som ser på denne muligheten, og dette kommer ikke til å drøftes i denne rapporten.
Posisjon og orientering samples typisk hvert 50-100ms, og returnerer da veldig mange punkter. Flere av disse punktene er overflødige, og kan fjernes. For eksempel kan flere punkter som danner en lineær bevegelse med konstant hastighet erstattes med to punkter som definerer startpunkt og endepunkt for bevegelsen. Oppgaven går ut på å lage en komprimeringsalgoritme i Matlab hvor rådatabanen skal representeres med så få punkter som mulig, samtidig som avviket i banen skal minimeres. Den resulterende banen skal gjenskape bevegelsene i rådatabanen med så lite avvik som mulig for alle egenskapene i banen. Det er også ønskelig å kunne velge å optimalisere banen for en gitt egenskap. For å bestemme banens- og algoritmens godhet er det også nødvendig å utvikle noen mål som skal brukes for å sammenligne resultatene fra algoritmen med resultatet fra andre komprimeringsalgoritmer.

Denne oppgaven har bydd på mange utfordringer. Rådatafilene måtte tolkes, prosesseres og leses inn i Matlab før de kunne arbeides med. For å løse oppgaven måtte brukerma-

nualene til RobotStudio og RAPID leses godt, for å forstå hvordan et RAPID-program og en robotbane ble laget. Det var nødvendig å lage flere figurer for å illustrere utviklingen i hver rådatabane og hver referansebane for å vurdere godheten i referansebanene og forstå hvor problemene lå. Referansebanene måtte skaleres i tid for å kunne sammenligne de med rådatabanene, da referansebanene går over kortere tid enn rådatabanene. Algoritmen ble skrevet i Matlab, som krevde at dokumentasjonen for Matlab ble lest grundig. Det krevde mye tid og teori om brukte metoder for å forstå Matlab-funksjonene som ble gitt av veilederen for oppgaven. Det ble undersøkt hvordan hver av funksjonene kunne tilpasses for å bli brukt i løsningen av oppgaven, og det ble gjort flere endringer i disse. Funksjonene ble også utvidet for å gi mulighet til å optimalisere hver bane for en gitt egenskap. Dette ble gjort ved å implementere en vektet sum av kvadrert feil i funksjonene. Det måtte også defineres noen mål på godheten i hver bane. Dette var er å kunne sammenligne resultatene fra hvert utkast til en algoritme, i tillegg til å sammenligne resultatet fra algoritmen med referansebanene fra ABB. Det ble undersøkt hvilken effekt ulike valg av parametre i algoritmen ga for utviklingen av avviket i en bane når antall punkter i banen ble redusert. I tillegg er det laget figurer og tabeller som illustrerer avviket i banen fra algoritmen, sammenlignet med avviket i referansebanene.

I delene som følger kommer

- Litt generell teori om industriroboter med matematiske metoder for å planlegge en robotbane
- Eksempler på ulike metoder for å programmere en robotbane
- Teori og muligheter ved programmering i RAPID og RobotStudio
- Teori om linjetilpassing og kurvetilpassing
- Ulike mål for avviket i banen og mål på banens prestasjoner
- Matematikk
- Verktøy og utstyr som ligger til grunn for løsningen av oppgaven
- Implementering av algoritme og flytdiagram
- Resultater fra algoritmen og sammenligninger med referansebane
- Diskusjon og forslag til videre arbeid
- Konklusjon
- Forklaring av funksjonene som er brukt.

2 Teori

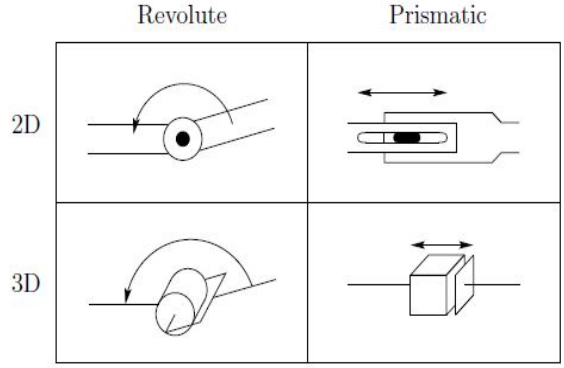
2.1 Roboter

Ordet robot stammer fra det tsjekkiske ordet robota, som betyr arbeid, og blitt brukt til å beskrive en rekke mekaniske innretninger. Den offisielle definisjonen på en robot fra Robot Institute of America er som følger:
"En robot er en omprogrammerbar, multifunksjonell manipulator, designet for å bevege materialer, verktøy, eller spesielle enheter gjennom en serie variable, programmerte bevegelser med hensikt å utføre et utvalg oppgaver."
Gjennom denne rapporten vil ordet robot, med mindre noe annet er spesifisert, referere til programmerbare robotarmer brukt i industrien.[2]
I endepunktet av roboten er det ofte montert en griper, sugekopp, en lakkeringspistol eller andre verktøy, som det også må tas hensyn til når roboten skal programmeres. Endepunktet av robotarmen vil herved bli referert til som robotens verktøy, selv om det ikke er montert et verktøy på roboten.

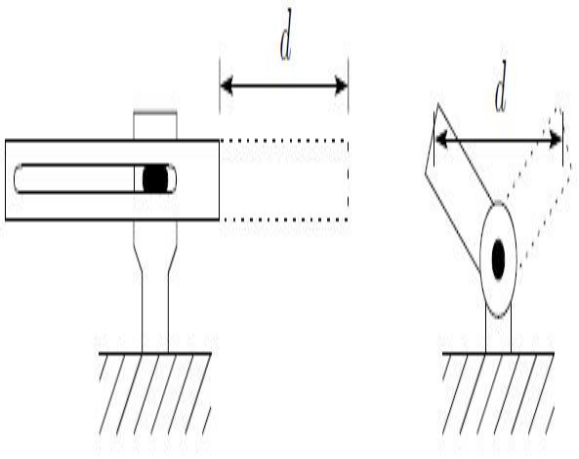
2.1.1 Matematisk modellering av roboter

Ved å lage en matematisk modell av roboten er det mulig å utvikle metoder for å planlegge og kontrollere robotbevegelser slik at roboten kan utføre spesifikke oppgaver. Den matematiske beskrivelsen av roboten varierer blant annet etter hvor mange, og hvilke typer ledd roboten har, hvor lange armene til roboten er, hvor mange frihetsgrader den har og hvor leddene er plassert i forhold til hverandre.

Robotledd: Industrielle roboter består av flere armer som er koblet sammen med flere ledd. Disse leddene er enten rotasjonsledd eller prismatiske ledd. Et rotasjonsledd virker som et hengsel som gir rotasjon mellom to av robotens armer, mens et prismatisk ledd gir en lineær bevegelse mellom to av robotens armer.



Figur 1.a: Venstre: rotasjonsledd,
Høyre: prismatisk ledd [2]

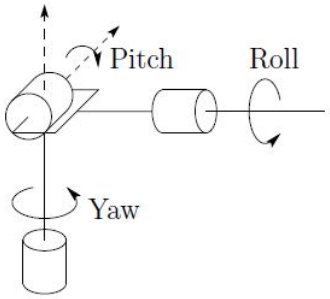


Figur 1.b: Venstre: prismatisk ledd
Høyre: Rotasjonsledd

Figur 1: Leddtyper for en industriell robot [2]

Frihetsgrader: Et objekt sies å ha n frihetsgrader, hvis dets *konfigurasjon*[1] kan beskrives med minimum n parametere. En industriell robot konstrueres ofte med 6 frihetsgrader, 3 for posisjon og 3 for orientering. Dersom robotarmen har mindre enn 6 frihetsgrader er det ikke mulig å nå alle posisjonene i rommet med full kontroll over orienteringen til verktøyet. Det er fullt mulig å lage en robot med mindre enn 6 frihetsgrader, men denne vil ikke være like allsidig. [2]

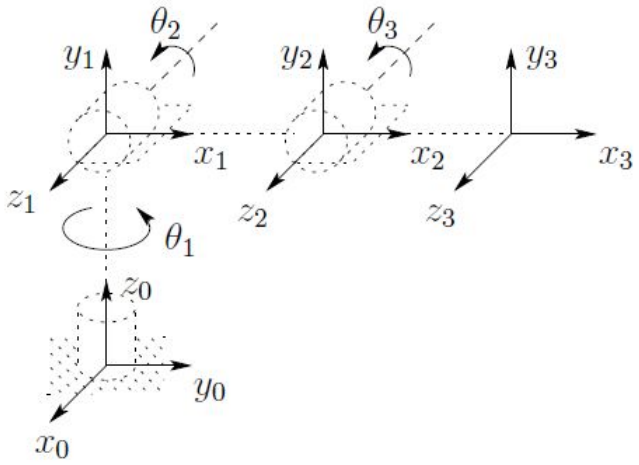
Kuleformet håndledd Leddene mellom armen til roboten og verktøyet blir kalt robotens håndledd. Leddene i håndleddet er som regel roterende ledd, og det blir mer og mer vanlig å designe disse som et kuleformet håndledd som har tre felles akser som skjærer samme punkt, se fig.2. Et slikt håndledd forenkler i stor grad den matematiske modelleringen av roboten da det gjør det mulig å skille fra hverandre posisjonering og orientering av verktøyet. [2]



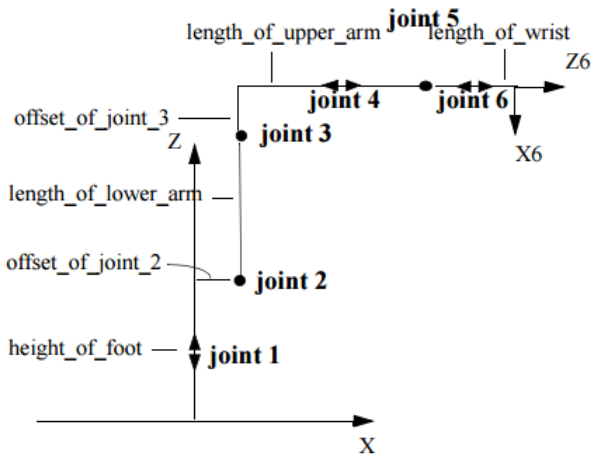
Figur 2: Kuleformet håndledd [2]

[1]Et objekts konfigurasjon er en fullstendig spesifikasjon av lokasjonen til alle punkt i objektet

Foroverkinematikk: Foroverkinematikk er en metode for å finne verktøyets posisjon og orientering i rommet, ved å se på posisjonen til alle robotens ledd.
I hvert ledd plasseres et koordinatsystem som er gjeldende kun for det leddet hvor aksesystemet er plassert. Det tas utgangspunkt i første koordinatsystem, som er plassert i basen til roboten, og rotasjon og translasjon finnes for hvert ledd frem til robotens verktøy. Rotasjonen og translasjonen blir ofte kombinert og representert i en matrise, som kalles transformasjonsmatrisen. [2]



Figur 3: En modell av en robot med et koordinatsystem i hvert ledd [2]



Figur 4: En modell av roboten IRB1400 med alle ledd, armer og posisjonen til disse, som er nødvendig å vite for å lage en kinematisk modell av roboten [3]

For hvert koordinatsystem i, må posisjon og rotasjon spesifiseres i forhold til koordinatsystem *i-1*. Dette gjøres med en homogen transformasjonsmatrise, A_i, hvor $\tilde{R}_i^{i-1}$ er rotasjonsmatrisen fra koordinatsystem *i-1* til koordinatsystem i og $\tilde{O}_i^{i-1}$ er translasjonsmatrisen fra koordinatsystem *i-1* til koordinatsystem i. Det betyr at rotasjonen og translasjonen til koordinatsystem i er beskrevet i forhold til hvor aksene i koordinatsystem

i-1 peker, og hvor origo for koordinatsystem i er plassert i rommet i forhold til origo for koordinatsystem *i-1*.

$$A_i = \begin{bmatrix} \tilde{R}_i^{i-1} & \tilde{O}_i^{i-1} \\ \bar{0} & 1 \end{bmatrix}$$

$$\tilde{R}_i^{i-1} = R_x(\alpha)R_y(\beta)R_z(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix} \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\tilde{O}_i^{i-1} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix}$$

hvor d_x er forflytning av origo for koordinatsystem i i forhold til *i-1*, i i x-retning gitt av koordinatsystem *i-1*, d_y og d_z er tilsvarende for y- og z-retning. α, β og γ er rotajonen rundt hhv. x-, y- og z-aksen i koordinatsystem *i-1*. Det er viktig å forstå at dersom det utføres en rotasjon rundt f.eks. x-aksen, så kan y- og z-aksen peke i en annen retning enn hva de originalt gjorde. Dette medfører at påfølgende rotasjon rundt y- og z-aksen vil gi en rotasjon rundt en rotert versjon av de originale aksene. Dette er også grunnen til at det er viktig å følge spesifisert rekkefølge for rotasjonen, da en annen rekkefølge kan gi en annen rotasjon. $\bar{0}$ er en 0-vektor, $\bar{0} = [0, 0, 0]^T$

Inverskinematikk: Inverskinematikk er en metode for å finne orienteringen og posisjonen til alle robotens ledd ut fra hvilken posisjon og orientering verktøyet har i rommet. [2] . Homogen transformasjonsmatrise benyttes som for foroverkinematikk, men det tas utgangspunkt i siste koordinatsystem og arbeides bakover til første koordinatsystem. Dette gir typisk flere mulige løsninger da robotarmene kan settes i ulike posisjoner, og fremdeles plassere verktøyet i samme posisjon i rommet.

Hastighetskinematikk: Den tidsderiverte av kinematikkligningene gir robotens Jacobian. Denne relaterer leddenes rotasjonshastighet til den lineære hastigheten og rotasjonshastigheten i robotens verktøy. [2] . Robotens Jacobian er nødvendig for å planlegge en robotbane og bevegelser med konstant hastighet.

Denavit-Hartenberg konvensjonen For å spesifisere en hvilken som helst transformasjonsmatrise trengs normalt 6 parametre. Det er likevel mulig å forenkle kinematikkligningene noe, til kun å trenge 4 parametre for å spesifisere en hvilken som helst transformasjonsmatrise. Dette gjøres ved å velge origo for hvert aksesystem i hvert av robotens ledd på en smart måte. Dette kalles Denavit-Hartenberg konvensjonen. Det er to regler som må oppfylles for å få dette til.

1. Aksen x_i må være vinkelrett på aksen z_{i-1}, hvor i refererer til aksesystem nr.i.
2. Aksen x_i må krysse aksen z_{i-1}, hvor i refererer til aksesystem nr.i.

2.1.2 Programmering av industielle roboter

Å programmere en robot til å utføre de oppgavene den skal er ikke en enkel oppgave da det krever både tid og kunnskaper. Her følger en rekke måter å programmere en robot på.

Programmering av en robot, direkte fra kinematikkligningene: For å programmere en robotbevegelse direkte med kinematikkligninger, trengs en god matematisk beskrivelse av roboten. Det må så bestemmes hvilke posisjoner i rommet som ønskes nådd, før det kan regnes ut hvilke posisjoner hvert robotledd må bevege seg til for å få robotens verktøy til å nå ønsket posisjon. Robotleddene kan så settes i disse posisjonene og punktet nås. Derretter må samme operasjon brukes for å nå neste punkt. Dette er både tidkrevende og tungvint, og det er ofte utviklet programvare som tar seg av disse beregningene.

Programmering av en robot ved å fysisk flytte leddene: Det er mulig å programmere en robot ved å fysisk vri og flytte på leddene til roboten, slik at robotens verktøy når ønsket posisjon. Dette gjøres ved å spore hvilken posisjon aktuatorene i leddene har til en hver tid. Da er det mulig å flytte en robotarm i ønsket posisjon, for så å lagre posisjonen til alle ledd. De lagrede posisjonene til leddene vil, så lenge roboten ikke flyttes og repeterbarheten i bevegelsen er god, alltid gi samme posisjon og orientering i rommet for robotens verktøy.
Dette er en metode å programmere roboter på som var vanligere før, men som fremdeles er mulig i dag. Blant annet finner man denne muligheten hos Aldebaran Robotics, som produserer en liten humanoidrobot kalt *NAO*, som har opp til 25 frihetsgrader. Med så mange frihetsgrader kan det være en mer effektiv måte å programmere en bevegelse på enn det vil være for en industrirobot.
For en industrirobot er det fare for at det vil bli i overkant tungt og vanskelig å flytte robotleddene, da robotene ofte er store og tunge. En ulempe med denne metoden er også at innspillingen av banen er nødt til å skje i samme rom som roboten, og det er da en begrensning på at roboten ikke kan plasseres i en omgivelse der det er farlig for mennesker å oppholde seg. Dette er en lite effektiv måte å programmere robotbevegelser på, sett i forhold til andre metoder.

Programmering av robot med en "teachpendant" En "teachpendant" er en bærbar kontrollenhet som ofte er utstyrt med en joystick. Denne kan benyttes til å bevege roboten om alle dens akser, slik at posisjonering av robotens verktøy i ønsket punkt er mulig, uten å måtte flytte roboten ved bruk av muskelkraft. Dette gir en mulighet til å programmere robotposisjoner og robotbaner på samme måte som ved å fysisk flytte leddene med muskelkraft, men er mindre krevende. Det vil fremdeles ta lang tid å flytte

roboten fra punkt til punkt og programmeringen av en hel robotbane kan ta veldig lang tid.

Programmering av en robot gjennom programvare fra leverandør: Det finnes flere produsenter av industriroboter i dag, og alle disse har utviklet egen programvare for programmering av roboter. ABBs *RobotWare* og *RobotStudio*, med programmeringsspråket *RAPID* er et eksempel på en slik programvare. Her trenger brukeren kun å lære seg instruksjonene som brukes i programvaren for å programmere en robot. Gjennom ferdig definerte funksjoner kan brukeren få roboten til å bevege seg til et spesifisert punkt, eller følge en spesifisert bane. Brukeren får for eksempel også mulighet til å bestemme toleranse i nøyaktighet, hastighet eller tidsbruk og begrensninger i akselerasjon når bevegelsene programmeres. En annen fordel er at det ikke er nødvendig å være i samme rom som roboten for å programmere en bane. Dette kan gjøres hvor som helst, og programvaren har ofte en mulighet til å simulere bevegelsen før den implementeres på roboten. Brukeren tar seg kun av den høyere ordens programmeringen, mens programvaren regner ut hvilke posisjoner hvert ledd må bevege seg til, og med hvilken hastighet de må bevege seg for å nå programmert posisjon. Dette gjøres gjerne gjennom kinematikkligningene. Selv om dette er en forholdsvis enkel måte for brukeren å programmere en robot på, sett i forhold til å utføre alle beregningene selv, så kan det ta ganske lang tid å programmere en kompleks bane.

Programmering av robot ved hjelp av Simplified robot programming:

> "A revolution in paint programming, ABB Simplified Robot Programming cuts programming time from hours to minutes, allowing even individuals without programming experience to create professional robotic paint programs easily"[4]

Ved hjelp av en håndholdt sensor som sporer posisjon og orientering i rommet, kan brukeren spille inn en bevegelse som ønskes gjentatt av roboten. Alle posisjonene blir logget og roboten kan bevege seg gjennom alle posisjonene i banen for å gjenta bevegelsen. Ved å tilpasse en bane med vesentlig færre punkt til den loggede dataen, legges det også til rette for at det raskt og enkelt kan være mulig å redigere banen i ettertid. Dette muliggjør å utføre endringer og optimalisere banen etter ønske. Innspillingen av en bane kan utføres over alt, uten å være i nærheten av roboten. Det er i tillegg ingen krav til programmeringskunnskaper fra den som spiller inn en bane. Innspillingen av en kompleks bane som tidligere tok flere timer eller dager å programmere, kan nå være ferdig i løpet av minutter. Det er også enklere å spille inn en bane som har en flyt lik den i de menneskelige leddene, noe som kan være vanskelig å programmere for hånd. [4]

2.2 Robotstudio og ABB-roboten

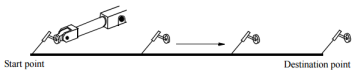
Robotstudio er et program fra ABB som brukes til å programmere og simulere robotbevegelser. Programmeringsspråket som brukes for å programmere robotbevegelser heter

RAPID. Gjennom *Robotstudio* og *RAPID* får brukeren mulighet til å lage punkter, relatert til et hvilket som helst spesifisert koordinatsystem, og lage en robotbane av disse. Brukeren får mulighet til å spesifisere flere egenskaper for bevegelsen, men de viktigste er bevegelsestype, hastighet for bevegelsen, nøyaktighet rundt punkt og orientering av verktøyet i hvert punkt.

- Type bevegelse bestemmes av valgt Move-funksjon 2.2.1
- Hastighet bestemmes gjennom en parameter kalt hastighetsdata 2.2.3
- Nøyaktighet rundt punkt bestemmes av sonedataen 2.2.2
- Verktøyets orientering bestemmes av orienteringen i hvert definerte punkt 2.2.4

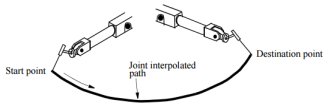
2.2.1 Move-funksjoner

MoveL flytter robotverktøyet lineært fra en posisjon til en annen. Se figur 5
(MoveL p1, v1000, z30, tool2;).



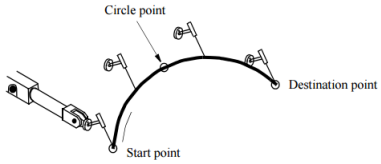
Figur 5: Bevegelse fra et punkt til et annet med MoveL [3]

MoveJ flytter robotverktøyet raskt, med en bevegelse som ikke er lineær. Den brukes kun når bevegelsen ikke er nødt til å være lineær. MoveJ-funksjonen er mindre utsatt for feilmeldinger grunnet orientering og posisjon i leddene, enn hva MoveL-funksjonen er. Dette er på grunn av at den ikke er begrenset til å bevege seg i en rett linje. Alle akser når programmert posisjon samtidig. Se figur 6
(MoveJ p1, vmax, z30, tool2;).



Figur 6: Bevegelse fra et punkt til et annet med MoveJ [3]

MoveC flytter robotverktøyet i en sirkulær bane, til en gitt posisjon. Funksjonen krever at to punkt spesifiseres, nemlig et endepunkt som skal nås, og et mellompunkt som er med på å definere sirkelen. Sammen definerer startpunktet, midtpunktet og endepunktet den sirkulære banen som skal utføres. Normalt sett holdes orienteringen uendret, relativt til sirkelen, gjennom hele bevegelsen. Se figur 7
(MoveC p1, p2, v500, z30, tool2;).



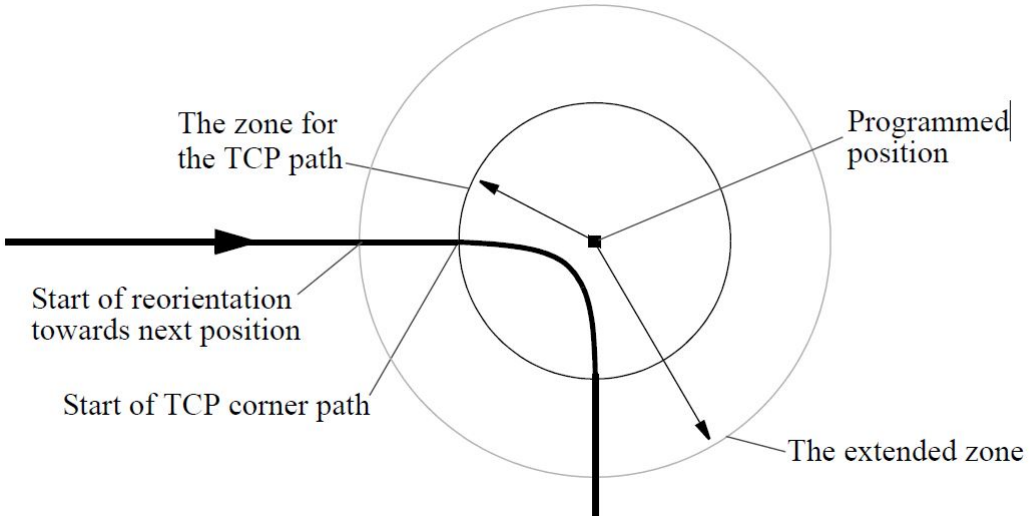
Figur 7: Bevegelse fra et punkt til et annet med MoveC [3]

2.2.2 Sonedata

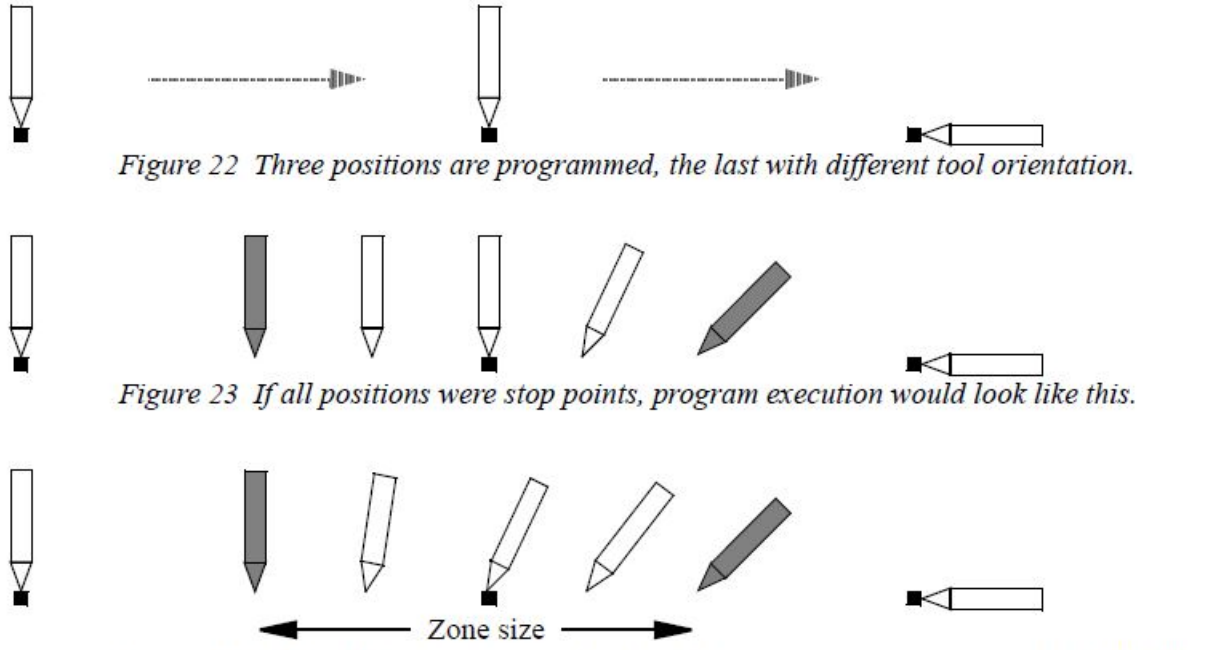
Sonedata er en parameter som spesifiseres når det utføres en bevegelse i RAPID. Denne parameteren bestemmer hvor nært et punkt roboten må bevege seg før den kan bevege seg videre til neste punkt og påvirker dermed nøyaktigheten til roboten når den skal følge en bane.[5] Sonedataen kan ses på som en sone, formet som en sirkel rundt punktet det er definert for. Når roboten når sirkelen, kan den begynne bevegelsen mot neste punkt. Størrelsen på sirkelen blir bestemt ved valg av sonedata.
Et punkt kan klassifiseres som et stoppepunkt eller et "fly-by" punkt. Et fly-by punkt har en bestemt sone med radius $r > 0$, mens et stoppepunkt har en sone med radius r=0. Dersom punktet er et stoppepunkt må roboten bevege verktøyet helt bort til punktet før den kan gå videre til neste punkt, noe som krever at bevegelsen stopper opp når roboten når punktet. Et fly-by punkt er et punkt som aldri nås, da roboten istedenfor å gå helt bort til punktet, endrer retningen på bevegelsen når ytterkanten av sonen nås. Roboten følger da en parabel, $x = y^2$ fra den treffer ytterkanten i sonen til den går ut av sonen igjen og mot neste punkt. Se fig 8. Dette medfører at roboten ikke behøver å stoppe bevegelsen, og kan starte akselerasjonen mot neste punkt så snart ytterkanten i sonen nås. Ved å bruke fly-by punkt i banen oppnås en glattere bane som enklere klarer å følge spesifisert hastighet i banen.

Sonedataen består egentlig av to soner, hvor den ene normalt er større enn den andre. Den minste sonen blir kalt en *TCP path-sone* og bestemmer hvor bevegelsen i x-, y- og z-retning kan begynne å endres. Den største sonen kalles den utvidede sonen, og bestemmer hvor roboten kan starte endringen i orientering og endringen i de eksterne aksene[5]. Se fig 8

Valg av verdi for sonedataen definerer hvor stort område rundt et fly-by punkt som kan klassifiseres som et treff i punktet. Bruker man *fine* som parameter for sonedataen, blir punktet definert som et stoppepunkt. For *referansebanene*, del.3.1 til ABB blir den forhåndsdefinerte sonedataen *z50* brukt for alle punkter i hele banen. Dette betyr at den minste sonen, *TCP path-sonen* har en radius på 50mm og den utvidede sonen for reorientering av verktøyet har en radius på 75mm.



Figur 8: Fly-by point", illustrasjon av sonedata [5]



Figur 9: Reorientering av verktøy i en bane [5]

2.2.3 Hastighetsdata

Hastighetsdata blir brukt til å spesifisere hvilken hastighet roboten og de eksterne aksene beveger seg med. Den bestemmer: [5]

- hvor raskt midtpunktet i robotens verktøy beveger seg

- hvor raskt reorienteringen på verktøyet er
- hvor raskt de lineære eller roterende eksterne aksene beveger seg

Hastigheten på bevegelsen vil være konstant fra startpunktet i bevegelsen, frem til sluttpunktet i bevegelsen.
Verdien på hastighetsdataen settes når det skal utføres en bevegelse i robotstudio. Det finnes mange forhåndsdefinerte hastighetsdata i robotstudio, men det er også mulig å definere egne hastighetsdata. I robotstudio vil en bevegelse skrives slik:
MoveL mittPunkt, v500, z10, tool1;, hvor *v500* er en forhåndsdefinert hastighetsdata.
Dersom man skal definere en egen hastighetsdata må det defineres 4 verdier: [5]

- v_tcp: Hastigheten til verktøyets midtpunkt, i mm/s
- v_ori: Hastigheten til verktøyets reorientering, i grader/s
- v_leax: Hastighet til den lineære eksterne aksen, i mm/s
- v_reax: Hastighet til den roterende eksterne aksen, i grader/s

I RAPID defineres en hastighetsdata slik:
VAR minSpeeddata speeddata := [v_tcp, v_ori, v_leax, v_reax] [5]

Når hastighetsdataen til en bevegelse skal spesifiseres, har Robotstudio også muligheten til å spesifisere hvor lang tid som skal brukes fra startpunktet i bevegelsen til sluttpunktet i bevegelsen. Denne funksjonen overstyrer hastighetsdataen, og setter hastigheten til den hastigheten som behøves for å ende opp med riktig konfigurasjon i programmert punkt i løpet av den spesifiserte tiden. I linjen under er det spesifisert at roboten skal flyttes fra posisjonen den står i, til punktet *mittPunkt* i løpet av 0.05 sekunder.

MoveL mittPunkt, v500/T:=0.05, z10, tool1;

2.2.4 Orientering

Orienteringen til robotens verktøy bestemmes for hvert punkt som programmeres i RAPID. Dette gjøres når punktet defineres. Orienteringen uttrykkes i kvaternioner, men det finnes funksjoner i RAPID som kan konvertere mellom kvaternioner og eulervinkler. Dersom to punkt p_1 og p_2 i en bane er definert med forskjellige orienteringer, og roboten skal forflytte seg fra p_1 til p_2, vil endringen i orientering skje lineært, med konstant hastighet som spesifisert i hastighetsdataen 2.2.3.

2.2.5 Robot

Da algoritmen skal kunne brukes på en hvilken som helst robot, og til en hvilken som helst oppgave, enten det er lakkering, sveising eller andre oppgaver, skal det ikke tas hensyn til robotens dynamikk. Dynamikken til robotene vil også endres etter hvor stor last den bærer på, så banen må i alle tilfeller tilpasses hver enkel situasjon.
Det antas i denne rapporten at robotene har en så kraftig akselerasjon at ønsket hastighet i en bevegelse oppnås øyeblikkelig og holdes konstant fra startpunktet i bevegeslsen til sluttpunktet i bevegelsen. Dette er et rimelig argument da robotene fra ABB er oppgitt å ha en kraftig akselerasjon. For eksempel har roboten *IRB 120* en oppgitt akselerasjon på $28m/s^2$

2.3 Linjetilpassing og kurvetilpassing

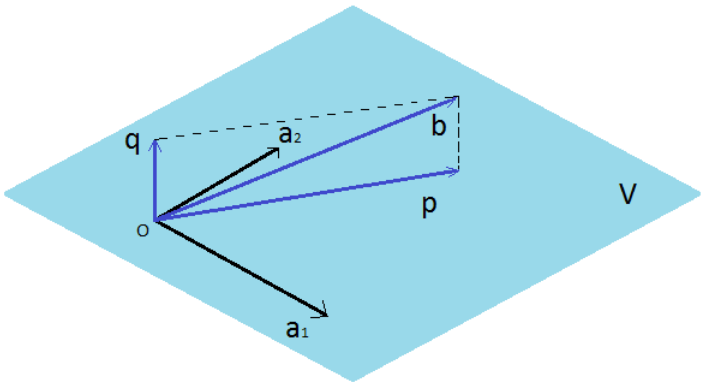
Kurvetilpassing er prosessen med å tilpasse en kurve eller linje til et sett med datapunkter. Det gir en matematisk beskrivelse av en kurve som går gjennom alle, eller nær alle punktene i datasettet. Ved å ha en matematisk beskrivelse av banen kan f.eks. en bane gjenskapes med et mye mindre antall punkter enn hva den startet med. I matematikken kalles dette ofte regresjon. [6]

Linjetilpassing er prosessen med å tilpasse en linje til et sett med datapunkter. Det må finnes parametre for linjen som gjør at avstanden til hvert eneste datapunkt minimeres. Da roboten kun skal bruke funksjonen *MoveL*, som er en lineær bevegelse, må banen kun bestå av en kombinasjon av rette linjer. [6]

2.3.1 Ortogonal projeksjon og minste kvadraters løsning, fra [1]

Minste kvadraters løsning brukes ofte når det skal finnes en løsning på et overbestemt ligningsystem hvor det ikke eksisterer noen eksakt løsning. Problemet er å finne en tilnærmet løsning som er så nær som mulig de observerte dataene.
Ved å ta utgangspunkt i to vektorer a_1 og a_2 som danner et plan V gjennom origo, og en utenforliggende vektor b, kan problemet illustreres, se figur.10. At ligningsystemet er inkonsistent betyr bare at b ikke ligger i planet V. b kan skrives som en sum av to vektorer,

$b = p + q$, der p er den ortogonale projeksjonen av b ned på planet V, og q er ortogonal til V.



Figur 10: Illustrasjon av den ortogonale projeksjonen p av b ned på planet V

Den ortogonale projeksjonen p av b, ned på planet V er det punktet i planet som er nærmest b. Det vil si at avstanden $||x - b||$ er mindre for $x = p$ enn noen andre punkter i planet V. Tilsvarende er også dens kvadrerte, $||p - v||^2$ den minste kvadrerte avstanden for noen punkt i V.
Dette kan bevises:
Gitt en vilkårlig vektor v i planet V. Fra pythagoras teorem følger:

$$||v - b||^2 = ||v - p||^2 + ||p - b||^2$$

som gir

$$||p - b||^2 = ||v - b||^2 - ||v - p||^2 \leq |v - b|^2$$

Dette gir videre, med mindre p=v.

$$||p - b|| < ||v - b||$$

Minste kvadraters løsning baserer seg på å finne den tilnærmede løsningen på et lignings-ystem som minimerer den kvadrerte feilen for hvert punkt. Da det ikke eksisterer en løsning på lignignsystemet direkte, brukes istedet det assosierte normalsystemet.
For alle lineære systemer $Ax = b$ er det assosierte normalsystemet $A^T Ax = A^T b$ konsistent, og alle løsninger av normalsystemet er det lineære systemets minste kvadraters løsning.
For et ligningssystem $Ax = b$, der A er en mxn matrise med n ukjente og m ligninger, er den minste kvadraters løsning den $\bar{x}$ gitt av normalsystemet $A^T A\bar{x} = A^T b$.

Dette gir $\bar{x} = (A^T A)^{-1} A^T b$ [1]

2.3.2 Linjetilpassing med minste kvadraters metode(MKM)

For å finne den rette linjen $y = a+bx$ som best passer til datapunktene $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, kan minste kvadraters løsning benyttes, se 2.3.1.
Den linjen som passer best til datapunktene finnes ved å løse ligningsettet

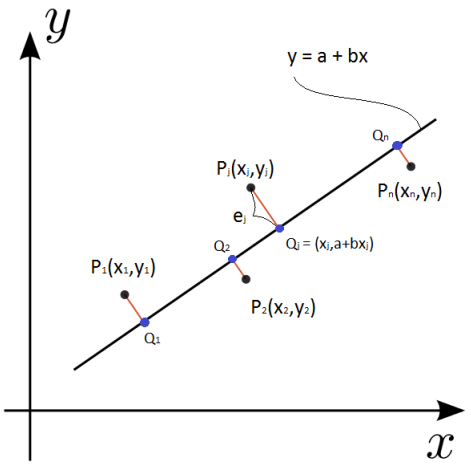
$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Dette kan skrives på matriseform som

$$Ax = b$$

Ligningssettets assosierte normalsystem finnes, og løses med hensyn på $\bar{x}$.

$$\begin{aligned} A^T A\bar{x} &= A^T b \\ (A^T A)^{-1} A^T A\bar{x} &= (A^T A)^{-1} A^T b \\ \bar{x} &= (A^T A)^{-1} A^T b \end{aligned}$$



Figur 11: Linjetilpassing med minste kvadraters metode. e_j symboliserer avviket fra et punkt til linjen. Q_j symboliserer punkt P_j projisert ned på linjen.

2.3.3 Kurvetilpassing med minste kvadraters metode

Minste kvadraters metode kan også brukes til å tilpasse en hvilken som helst polynomisk kurve til et sett med data [1] [6].
For eksempel kan det tenkes å være ønskelig å finne det polynomet $y = a_0 + a_1x + a_2x^2 +$

$... + a_k x^k$ som passer best til punktene $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$.

$$\begin{bmatrix} 1 & x_1 & x_1^2 & ... & x_1^k \\ 1 & x_2 & x_2^2 & ... & x_2^k \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & ... & x_n^k \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Dette skrives på matriseform

$$Ax = b$$

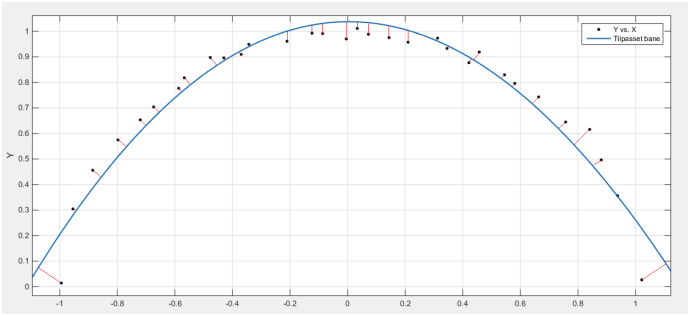
Det multipliseres med A^T på begge sider av likhetstegnet for å få det assosierte normalsystemet

$$A^T A\bar{x} = A^T b$$

Systemet løses så med hensyn på $\bar{x}$

$$\bar{x} = (A^T A)^{-1} A^T b$$

Resultatet blir en $\bar{x} = [a_0, a_1, a_2, ..., a_k]^T$ som inneholder alle koeffisientene til polynomet $y = a_0 + a_1 x + a_2 x^2 + ... + a_k x^k$, som best beskriver det sett med punkter som var oppgitt [1] .

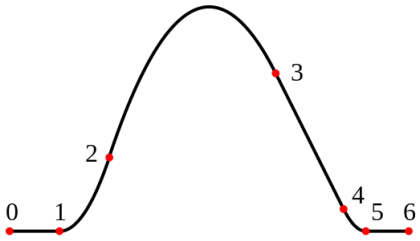


Figur 12: Et polynom av 3.orden tilpasset et sett med punkter. Røde linjer viser avviket fra kurve til punkt. Sum av kvadrert feil: 0.1444

2.3.4 Spline og bézierkurve

Å tilpasse en linje, eller et polynom, eller et sett med rette linjer til en samling med datapunkt, er ikke nødvendigvis den beste tilnærmingen til en bane som går gjennom alle punktene. Det finnes flere, bedre egnede metoder som kunne blitt brukt dersom de var kompatible med *Robotstudio og RAPID*, ved at det f.eks. fantes en *MoveBezier*-funksjon i *RAPID*.

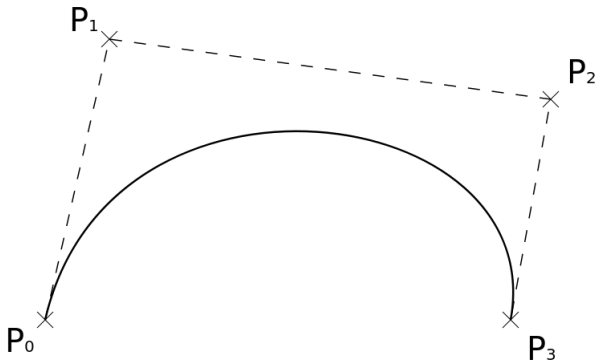
Spline Et lavere ordens polynom er greit å bruke for å tilnærme en funksjon til en glatt kurve i en begrenset region, men over en større region virker det ikke like bra. Det er mulig å bruke et høyere ordens polynom, men dette kan vise seg å være veldig oscillerende, og kan gi dårlig nøyaktighet.
Løsningen kan være å sette sammen flere lavere ordens polynom til en stykkevis polynomfunksjon for å beskrive kurven. Dette kalles en *spline*-funksjon. En spline-funksjon av orden p med knutepunkt $k_1, k_2, ..., k_K$ er en stykkevis polynom-funksjon, s, som endrer form på polynomet i hvert knutepunkt og hvis høyeste orden polynom er av orden p. Dette gjøres på en slik måte at den deriverte nr.j av s er kontinuerlig for $j \leq p - 1$. I praksis er p generelt 1,2 eller 3. [7],[8]



Figur 13: En spline-funksjon bestående av 6 polynomiske funksjoner. Knutepunktene er markert med rødt

Bèzierkurve En bèzierkurve er en parametrisk kurve som ofte blir brukt i datagrafikk. Gitt (n+1) punkter i P_i: i = 0,1,2,...,n, en bèzierkurve er da definert som følger:
$P(t) = \sum_{i=0}^{n} P_i B_i(t), 0 \leq t \leq 1,$
hvor $B_i(t) = \binom{n}{i}(1-t)^{n-i}t^i$ er Bernsteins polynom.

En bèzierkurve går alltid gjennom første og andre punkt. P_i er kurvens kontrollpunkter og dens grad er alltid èn mindre enn antall kontrollpunkter. Polygonet som dannes av kontrollpunktene og rommet under denne bestemmer hvordan kurven beveger seg. [8]



Figur 14: En bèzierkurve med 4 kontrollpunkter [9]

Det kunne tenkes at det kunne blitt brukt en slik kurve, eller en sammensatt funksjon av flere slike kurver for å beskrive en ulineær bevegelse i rommet på en god måte, dersom det fantes en moveBèzier-funksjon i RAPID.

2.3.5 K-means clustering

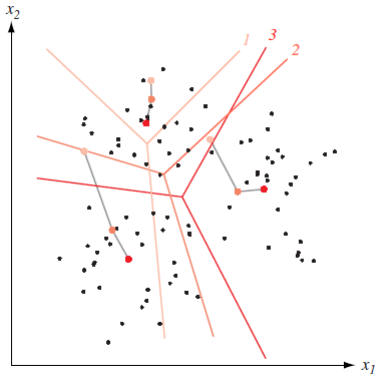
K-means clustering er en metode som brukes for å dele opp en samling med datapunkter x_k inn i flere grupper, $\omega_i, i = 1, 2, ..., n$. Hensikten er at sum av kvadrert distanse fra alle punkt til gjennomsnittet av gruppen hvor punktene hører hjemme minimeres. Gjennomsnittet kan også kalles gruppesenteret.
Punktene deles først inn i noen grupper. Den initielle inndelingen av punkter trenger ikke å være optimal. Gruppesenteret $\hat{\mu}_i$ for hver gruppe finnes for den initielle inndelingen av punkter. Den kvadrerte euklidske avstanden $||x_k - \hat{u}_i||^2$ for alle punkt x_k til hvert gruppesenter, $\hat{\mu}_i$ beregnes, og dersom det finnes at et av punktene ligger nærmere en annen gruppes senter enn sin egen gruppes senter, da flyttes dette punktet. Gruppenes senter beregnes på ny, for så å gjenta prosessen. Dette foregår frem til det ikke lenger finnes noen endring i gruppenes senter, da har algoritmen konvergert og inndelingen av punktene i grupper er ferdig. Se figur 15 for K-means algoritmen og figur 16 for utvikling av punktenes inndeling i grupper ved beregning av nytt gjennomsnitt for gruppene [10]. Avhengig av den initielle inndelingen av punkter og beregning av initielle senter, kan K-means algoritmen konvergere mot ulike løsninger.

$$\hat{P}(\omega_i|x_k, \hat{\theta}) \approx \begin{cases} 1, & hvis \quad i = m \\ 0, & 0 \quad ellers \end{cases}$$

Algorithm 1 (K-means clustering)

```
1 begin initialize n, c, μ_1, μ_2, ..., μ_c
2         do classify n samples according to nearest μ_i
3             recompute μ_i
4         until no change in μ_i
5   return μ_1, μ_2, ..., μ_c
6 end
```

Figur 15: K-means algoritmen, brukt til å dele inn et datasett i c grupper [10]



Figur 16: Utvikling av grensene for hver gruppe når punkter flyttes og nytt gjennomsnitt finnes [10]

2.4 Feilmål

For å kunne sammenligne flere baner må feilen i banene uttrykkes med samme mål. Feilen brukes som et optimaliseringskriterie for å tilpasse banen så godt som mulig til rådataen, samtidig som den brukes for å sammenligne resultatet fra algoritmen med allerede eksisterende baner fra ABB. Banene fra ABB vil videre i rapporten kalles *referansebanene*.
For å få et uttrykk for feilen i banen, sett i forhold til hvert punkt i rådataen, som jo inneholder mange flere punkt enn banen, må hvert punkt i rådataen projiseres ned på de rette linjene som dannes mellom to og to punkt i banen. Dette legges som grunnlag i beregning av feilen for alle banene. Mer om projeksjon i del 2.6.3

2.4.1 Sum av absolutt feil

Avviket, e_i kan finnes for hvert rådatapunkt x_i, ved å se på avstanden mellom rådatapunktet og tilhørende punkt projisert ned på linjen i banen, $\hat{x}$. Summen av alle disse avvikene kan så bergenes.

$$S = \sum_{i=1}^{L} e_i = \sum_{i=1}^{L} (x_i - \hat{x}(t_i))$$

Summen av avvik er ikke aktuell å bruke da feil med omvendt fortegn opphever hverandre. En slik situasjon oppstår gjerne dersom det eksisterer punkt både over og under linjen som er tilpasset rådatapunktene. En løsning kan være å bruke sum av absolutt feil.
Sum av absolutt feil ser på den absolutte avstanden fra hvert punkt i rådataen, ned til linjen som er tilpasset rådataen. Å bruke absolutt feil fører til at feil med motsatt fortegn ikke kansellerer hverandre.

$$S = \sum_{i=1}^{L} |e_i| = \sum_{i=1}^{L} \sqrt{(p_i - \hat{p}(t_i))^2} = \sum_{i=1}^{L} \sqrt{(x_i - \hat{x}(t_i))^2 + y_i - \hat{y}(t_i))^2 + z_i - \hat{z}(t_i))^2}$$

Sum av absolutt feil som optimaliseringskriterie tar ikke så mye hensyn til utliggerene, punkt som ligger lenger borte enn resten av punktene. Dette vil resultere i en bane som

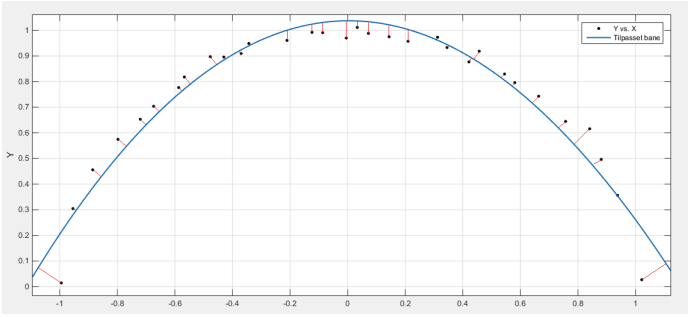
kan ha vesentlige feil for enkelte punkter. Da et punkt i rådatabanen inneholder både tidsstempel, posisjon og orientering vil dette kunne gi store feil i posisjon, orientering og hastighet som ikke er ønskelig.

2.4.2 Sum av kvadrert feil

Ved å kvadrere feilen i hvert punkt unngås problemet med at positive og negative feil opphever hverandre. Summen av alle de kvadrerte feilene vil gi et mål på total feil gjennom hele banen [6] . Sum av kvadrert feil brukes for å gjøre endringer i banen og se om det får en forbedring i nøyaktigheten på banen ved at feilen minimeres. Da feilen kvadreres vil et punkt i rådatabanen, x_1 som ligger langt fra banen gi en langt høyere påvirkning på sum av kvadrert feil enn et punkt, x_2 som ligger nær banen. Det vil da være lønnsomt å tillate en endring i banen som fører til at banen ligger lenger fra x_2, men nærmere x_1. Resultatet av å bruke sum av kvadrert feil som optimaliseringskriterie i banen vil være en bane som tilpasses alle punktene på best mulig måte. Selv for punkter som ligger med en lengre avstand fra resten av punktene.

$$S = \sum_{i=1}^{L} e_i^2 = \sum_{i=1}^{L} ||(p_i - \hat{p}(t_i))||^2 = \sum_{i=1}^{L} (x_i - \hat{x}(t_i))^2 + (y_i - \hat{y}(t_i))^2 + (z_i - \hat{z}(t_i))^2$$

I figur 17 vises en kurve tilpasset et sett med punkter. Kurven treffer ikke alle punktene, og feilen e_i for hvert punkt p_i er markert med en rød linje fra punkt p_i ned til tilsvarende punkt $\hat{p}_i$ på kurven. Summen av den euklidske lengden for alle disse punktene kvadrert gir sum av kvadrert feil.



Figur 17: En kurve tilpasset er sett med punkter, hvor feilen for hvert punkt er markert med rødt. Sum av kvadrert feil: 0.1444

2.4.3 Vektet sum av kvadrert feil

Vektet sum av kvadrert feil er en metode som ofte benyttes når noen av målingene i et datasett ikke er like pålitelige som resten av målingene, grunnet usikkerhet i målingen. Til forskjell fra vanlig sum av kvadrert feil i del 2.4.2, som gir lik vekting av alle målte feil, vil vektet sum av kvadrert feil multiplisere feilen funnet i den usikre målingen med en konstant med verdi mindre enn 1. Eller alle andre målinger med en verdi større enn 1.

Dette fører til at feilen i den usikre målingen ikke bidrar like mye som resten av målingene når feilen summeres [6].
Vektet sum av kvadrert feil regnes ut slik:

$$S_{vektet} = \sum_{i=1}^{L} W_{ii} e_i^2 = \sum_{i=1}^{L} W_{ii} ||(p_i - \hat{p}(t_i))||^2 = \sum_{i=1}^{L} W_{ii}((x_i - \hat{x}(t_i))^2 + (y_i - \hat{y}(t_i))^2 + (z_i - \hat{z}(t_i))^2),$$

hvor W_{ii} er vektmatrisen og e_i er feilen for hvert punkt i rådataen, til tilhørende punkt på linjen i banen, målt i euklidsk avstand. Tilhørende punkt på linjen i banen finnes ved å projisere punktet i rådataen ned på linjen som beskrevet i del 2.6.3

I algoritmen blir vektet sum av kvadrert feil brukt til å vektlegge feilen funnet i posisjon, orientering eller hastighet. Dette gjøres ved å multiplisere feilen med en konstant større enn 1 for å la feilen for valgt egenskap bidra mer enn feilen for de andre egenskapene ved banen. Algoritmen vil da kompansere ved å flytte punktene i banen slik at avviket for valgt egenskap minimeres på bekostning av avviket for de andre egenskapene. Ved å benytte en slik funksjon for beregning av avviket vil det være mulig å optimalisere banen i henhold posisjon, hastighet eller orientering.

2.5 Prestasjonsmål

Når det lages en algoritme for generering av en bane, må det bestemmes hva banen skal optimaliseres for. Det er bestemt at banen skal kunne optimaliserers for disse egenskapene:

- Minimalt avvik i posisjon
- Minimalt avvik i hastighet
- Minimalt avvik i orientering
- Minimalt antall punkt i banen

Det vil da være aktuelt å se på sum av kvadrert feil, gjennomsnittlig avvik og maksimalt avvik for hver egenskap. Det vil i tillegg undersøkes hvor stor den summerte endringen i posisjon og orientering er i banen, i forhold til i rådatabanen. Dette vil si noe om hvor mye godt banen klarer å gjenskape dynamikken i rådatabanen.

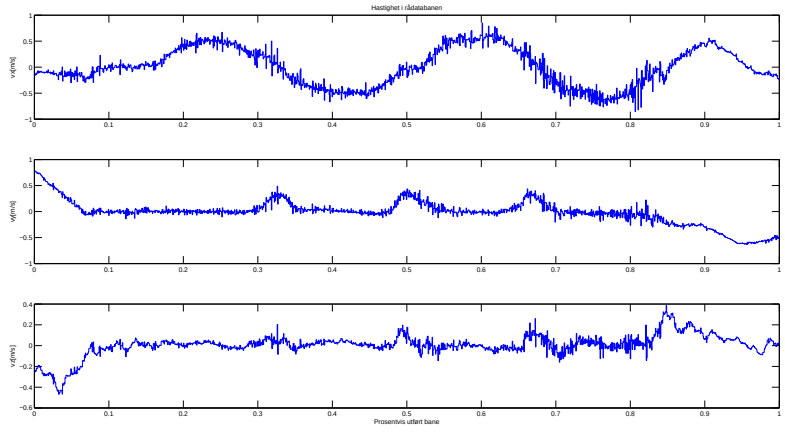
2.5.1 Avvik i posisjon

Det er viktig å minimere avviket i posisjon for en bane for å kunne stole på at roboten beveger seg der den skal. Et minimalt avvik i posisjon muliggjør også å bruke banen til oppgaver som krever presisjon, gitt at sensoren som brukes til å spille inn banen ikke er støybefengt.
Ved å se på den euklidske avstanden, $S = \sqrt{(\hat{x_i} - x_i)^2}$ fra hvert punkt, x_i i rådatabanen, til det tilsvarende punktet, $\hat{x}_i$ i banen og summere disse, vil det være mulig å finne total feil i posisjon i banen. Rådatabanen inneholder mange flere punkter enn banen, men da det kun benyttes rette linjer i banen er det mulig å projisere rådatapunktene ned på

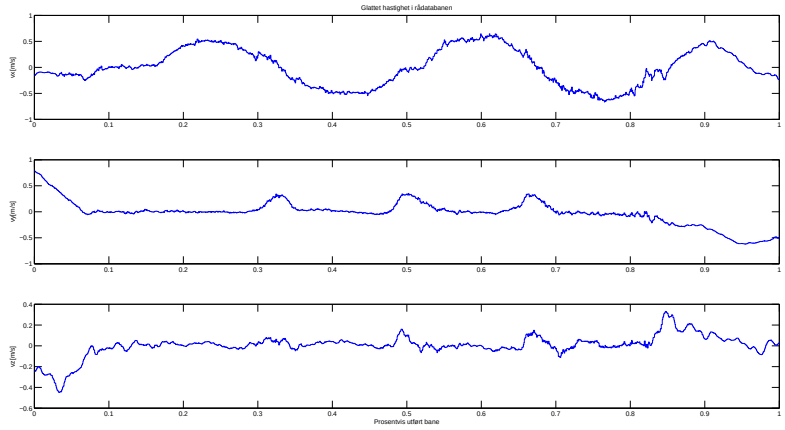
linjene som dannes av punktene i banen. Det er da mulig å regne ut euklidsk avstand fra banen til alle punktene i rådatabanen. Se del 2.6.3 for informasjon om hvordan punktene projiseres ned på linjene i banen. Avviket i posisjon vil kunne uttrykkes i mm dersom det måles i euklidsk avstand. Det er også mulig å finne gjennomsnittlig avvik i posisjon ved å summere avviket for hvert punkt i rådataen, for så å dele på antall punkt i banen. $S_{gjennomsnitt} = \frac{1}{n} \sum_{i=1}^{n} \sqrt{(\hat{x}_i - x_i)^2}$

2.5.2 Avvik i hastighet

Hastighet på bevegelsene er en viktig faktor i utførelsen av mange oppgaver. For eksempel vil en lakkeringsrobot som ikke beveger seg like raskt som den burde, eller raskere enn den burde, ikke gi en optimal lakkpåførsel på objektet den lakkerer. Det er derfor ønskelig at banen som genereres fra rådataen skal ha lik hastighet som i rådatabanen. Maksimal hastighet og akselerasjon er avhengig av robotens dynamikk. En robot kan ha veldig forskjellig dynamikk i forhold til en annen robot, og dynamikken blir også påvirket av hvor stor last roboten bærer. Da oppgaven ikke er spesifisert mot en bestemt robot eller en bestemt oppgave, vil det ikke tas hensyn til robotens dynamikk.
Da banen kun består av lineære bevegelser, vil hastigheten fra et punkt til et annet være konstant. Det er da nødvendig å finne ut hvor punktene i rådatabanen befinner seg, i forhold til punktene i banen, sett i forhold til tid. Hastigheten fra punkt til punkt i rådatabanen blir sammenlignet med hastigheten fra punkt til punkt i banen, og avviket kan da beregnes. Det totale summerte avviket, maksimalt avvik og gjennomsnittlig avvik i banen kan da finnes, og brukes til å evaluere banens godhet.
Da hastigheten i banen er endring i posisjon over tid, vil små bevegelser som følger av for eksempel at sensoren blir ført gjennom rommet av en ustø hånd, eller støy i målingene fra sensoren føre til at det kan bli store endringer i hastighet underveis i rådatabanen. Det kan da være aktuelt å sammenligne hastigheten i banen med en glattet versjon av hastigheten i rådatabanen. Se figur 18 og 19 for en sammenligning av hastighet og glattet hastighet i en rådatabane. Om det er realistisk mulig å følge hastigheten i banen nøyaktig er avhengig av spesifisert sonedata i banen, se del 2.2.2 og robotens dynamikk.



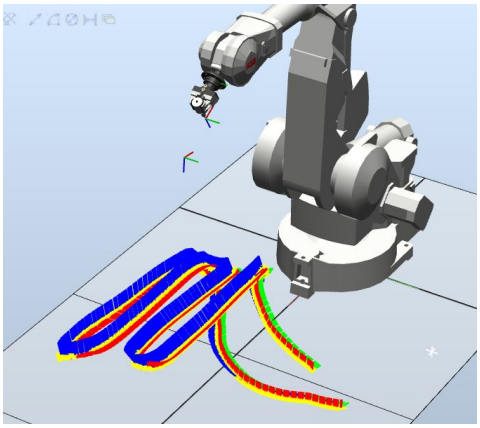
Figur 18: Hastighet i rådatabanen uten glatting, i x-, y- og z-retning



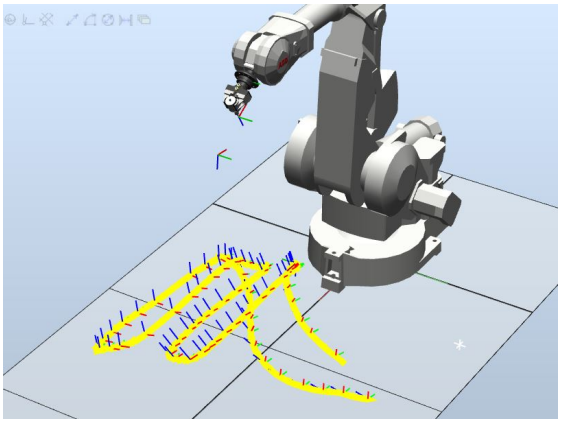
Figur 19: Glattet hastighet i rådatabanen, i x-, y- og z-retning

2.5.3 Størrelse på program

Hovedargumentet for å utvikle en algoritme slik som den som presenteres i denne rapporten er å beskrive en robotbane så godt som mulig med så få punkter som mulig. Ved å tilnærme en bane til rådataen på en slik måte at banen får så få punkt som mulig vil det være mye enklere å gjøre endringer i banen i ettertid. Eenten dette er gjennom et grafisk grensesnitt eller ved manuelt å manipulere punktene i banen gjennom en editor. Dette gjør det mulig å endre en bane som ikke er helt tilfredstillende, uten å måtte spille inn en ny bane. I stedet for å endre hundrevis av punkter for å gjøre en forandring i banen, er det nå mulig å gjøre samme forandring i banen ved å endre et punkt eller to. Figur 20.b og 20.a illustrerer dette.



Figur 20.a: Original bane med mange punkter



Figur 20.b: Ny bane med færre punkter

2.5.4 Avvik i orientering

Det er viktig å ha riktig orientering gjennom hele robotbanen, uansett om roboten skal brukes til lakkering, sveising, eller andre oppgaver. Robotbanen skal kunne brukes til å arbeide med tredimensjonale objekter der roboten skal kunne bevege seg rundt, over og under objektet, og det ønskes da ikke noen begrensninger på grunn av store feil i orienteringen.

For å presentere avviket i orienteringen på en forståelig måte, blir orienteringen konvertert fra kvaternioner til eulervinkler som i del 2.6.4

Som et mål på avviket i orientering måles disse parametrene:

- Avvik i orientering i hvert punkt i banen mot hvert punkt i rådataen. Orienteringen endres lineært, og det er dermed mulig å måle feilen i orientering, selv for punkter i rådataen som ligger mellom to punkter i banen. Dette gjøres ved å se på orienteringen som rette linjer, og projisere punktene fra rådataen ned på linjene i banen på samme måte som for posisjon. Se del 2.6.3 og 2.5.1 . Avviket kan til slutt uttrykkes som en sum av totalt avvik gjennom banen, et gjennomsnittlig avvik gjennom hele banen og med et mål på maksimalt avvik. Til sammen uttrykker dette hvor god banen er til å følge orienteringen i rådataen.
- Total endring i orientering i banen blir sammenlignet med total endring i orientering i rådataen. Dette målet uttrykker hvor godt vi klarer å gjenskape rådatabanen og dynamikken i denne. Dette er et uttrykk på hvor stor del av orienteringen som blir "glemt".

2.5.5 Visuell bedømmelse av banen

Det er ikke ønskelig å ha en typisk 'robotbevegelse' hvor roboten beveger seg raskt opp til et punkt, stopper og går videre i en 'rykk-og-napp' bevegelse. Det er derfor ønskelig å ha en glatt bane som virker naturlig og flytende. Dette må evalueres visuelt og kan oppfattes

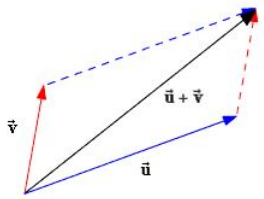
noe forskjellig fra person til person. Det er vanskelig å sette et mål på en slik evaluering, men da det vil være ønskelig å unngå slike bevegelser er det fremdeles aktuelt å se på. Visuell bedømmelse av banen kan gjøres gjennom simulering av banen i robotstudio. Hvor glatt og flytende bevegelsen fra roboten er, avhenger i stor grad av spesifisert sonedata. Se del 2.2.2.

2.6 Matematikk

2.6.1 Vektorregning

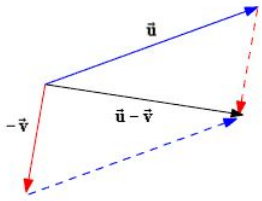
Her kommer generell teori om vektorregning. En vektor er en størrelse som er bestemt av både måltall og retning. Eksempler på dette kan være hastighet, forflytning eller kraft. Geometrisk representeres en vektor ofte ved et rett linjestykke med en bestemt lengde og retning, symbolisert med en pil. [1]
En vektor i $\mathbb{R}^3$-rommet består av en x-, y- og z-komponent, $\bar{v} = [x, y, z]^T$. Lengden av en tredimensjonal vektor er definert som: $|v| = \sqrt{x^2 + y^2 + z^2}$.

Addisjon av vektorer To vektorer, $u = [x_1, y_1, z_1]^T$ og $v = [x_2, y_2, z_2]^T$ kan adderes ved å addere vektorene komponentvis, $w = u + v = [x_1 + x_2, y_1 + y_2, z_1 + z_2]^T$. Resultatet kan ses på som om v forflyttes slik at startpunktet på vektoren starter i sluttpunktet på u, og vektoren w peker fra startpunktet i u til sluttpunktet i den forflyttede v. [1]



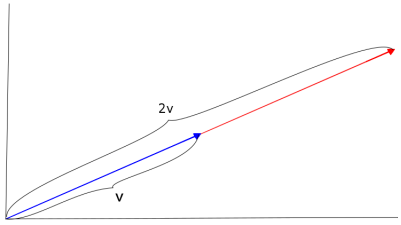
Figur 21: Addisjon av to vektorer [11]

Subtraksjon av vektorer Subtraksjon av en vektor fra en annen virker på samme måte som addisjon, men ved at en av vektorene har motsatt fortegn. Dvs. $w = u - v = u + (-v)$. [1]



Figur 22: Subtraksjon av en vektor fra en annen [11]

Multiplikasjon med en konstant Det er mulig å multiplisere en vektor med en konstant. Dersom konstanten er positiv og ulik 1, beholder vektoren sin retning, men lengden endres. Er konstanten negativ og ulik -1, blir retningen snudd og lengden endres. Hver komponent i vektoren multipliseres da med konstanten.
Eks.: $w = 2v = 2[x_1, y_1, z_1] = [2x_1, 2y_1, 2z_1]$



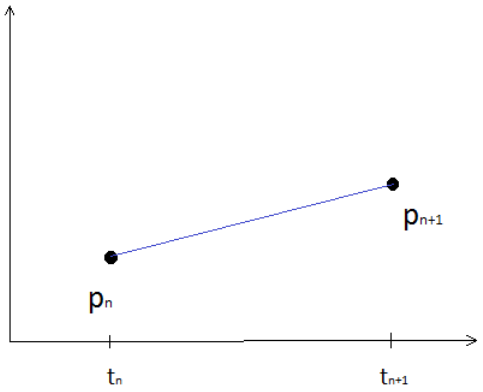
Figur 23: En vektor multiplisert med en positiv konstant. Retning beholdes, lengde endres

2.6.2 Hastighet i 3 dimensjoner

For å beregne hastigheten fra et punkt p_n til et annet punkt p_{n+1}, må forflytningen i rommet og tiden som brukes på forflytningen finnes. Dette gjøres ved å beregne den euklidske avstanden for vektoren som spennes ut fra p_n til p_{n+1}, for så å dele på tiden forflytningen bruker. Da finnes gjennomsnittlig hastighet for forflytningen.
Hastigheten kan også beregnes for hver dimensjon, som medfører at avvik i hastighet kan undersøkse for hver retning. Hastigheten i hver retning kan så kvadreres, summeres og roten av disse kan finnes for å finne gjennomsnittlig hastighet for bevegelsen. Se ligningene under.

Dersom p_n og p_{n+1} er to punkter i rommet og tiden som brukes for å bevege seg fra p_n til p_{n+1} er $\triangle t$ og i, j og k enhetsvektorene $i = [1,0,0]^T$, $j = [0,1,0]^T$, $k = [0,0,1]^T$, kan hastigheten for bevegelsen finnes ved:

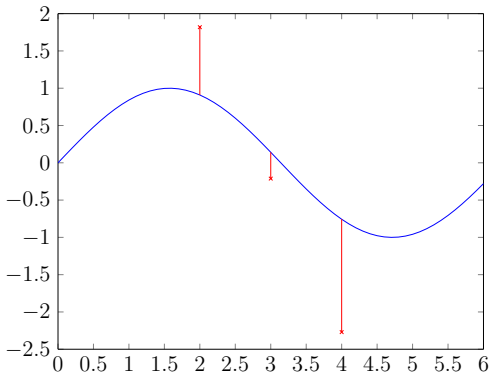
$$\bar{r}_n(t) = x_n(t)i + y_n(t)j + z_n(t)k$$
$$\bar{r}_{n+1}(t) = x_{n+1}(t)i + y_{n+1}(t)j + z_{n+1}(t)k$$
$$\triangle\bar{r}(t) = \bar{r}_{n+1}(t) - \bar{r}_n(t) = (x_{n+1}(t) - x_n(t))i + (y_{n+1}(t) - y_n(t))j + (z_{n+1}(t) - z_n(t))k$$
$$V_{avg}(t) = \frac{\triangle\bar{r}(t)}{\triangle t} = \frac{(x_{n+1}(t) - x_n(t))}{\triangle t}i + \frac{(y_{n+1}(t) - y_n(t))}{\triangle t}j + \frac{(z_{n+1}(t) - z_n(t))}{\triangle t}k$$
$$\bar{V}(t) = V_x(t) + V_y(t) + V_z(t) = \frac{d_x(t)}{dt}i + \frac{d_y(t)}{dt}j + \frac{d_z(t)}{dt}k$$
$$|\bar{V}(t)| = V(t) = \sqrt{V_x(t)^2 + V_y(t)^2 + V_z(t)^2},$$



Figur 24: To punkter på i rommet p_{n+1} og p_n danner en vektor p_{n+1}-p_n

2.6.3 Projeksjon av et punkt ned på en linje

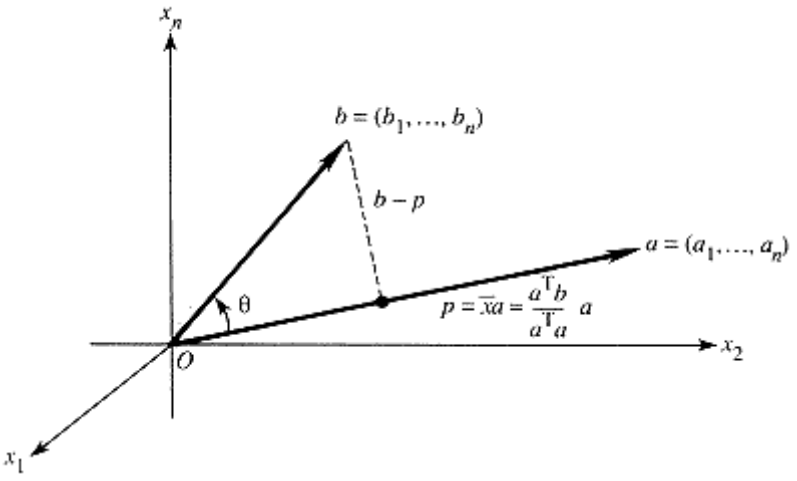
Projeksjon er et annet ord for avbildning og er en entydig og spesifisert geometrisk overføring av punkter fra en referanseflate til en projeksjonsflate. Projeksjon foregår ved at punktene tenkes overført langs rette linjer i rommet til et objekt, eller en linje i rommet. Punktet avbildes da i det punkt der projeksjonslinjen treffer objektet eller linjen. [12] [13]
Når en bane skal tilpasses et sett med rådatapunkter på best mulig måte, må feilen for hvert punkt undersøkes for å se om banen har oppnådd ønsket godhet. Feilen er her definert som den euklidske avstanden fra alle punkt i rådataen til den tilpassede banen.
Ved å projisere et hvert punkt i rådataen ned på banen, er det mulig å regne ut avstanden fra kurven for hvert eneste punkt i rådataen, selv for punkter i rådataen som ligger mellom punkter i banen.



Figur 25: Projeksjon av 3 omkringliggende punkter ned på en sinuskurve

Den ortogonale projeksjonen av en vektor $\bar{u}$ ned på linjen gitt av vektoren $\bar{v}$ finnes ved ligningen[1]:

$$\bar{w} = \frac{\bar{u}\cdot\bar{v}}{\bar{u}\cdot\bar{u}}\bar{u} = \frac{\bar{u}^T\bar{v}}{\bar{u}^T\bar{u}}\bar{u} = \frac{[u_x u_y u_z]\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}}{[u_x u_y u_z]\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}}\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \frac{u_x v_x + u_y v_y + u_z v_z}{u_x u_x + u_y u_y + u_z u_z}\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \text{ [1]}$$

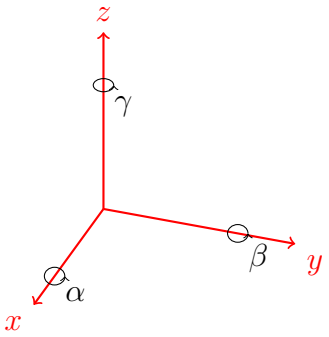


Figur 26: Projeksjonen p av punktet b ned på linjen a [14]

2.6.4 Rotasjon med kvaternioner

Kvaternioner er den mest kjente formen for hyperkomplekse tall [15],en ikke-kommutativ utvidelse av de komplekse tallene, først introdusert i 1843 av William Rowan Hamilton. De har formen $q = \alpha_1 1 + \alpha_2 i + \alpha_3 j + \alpha_4 k$, der koeffisientene $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ er reelle tall, og basiselementene i, j og k multipliseres slik: $i^2 = j^2 = k^2 = -1$, $ij = -ji = k, jk = -kj = i, ki = -ik = j$. Kvaternioner har flere anvendelser, bl.a. innen tallteori og i fysikk, særlig kvantemekanikk. [15] [16]
Quaternioner er også mye brukt i 3D-spillutvikling og i industrielle roboter for å beskrive rotasjoner og orienteringer i rommet.

For å beskrive total rotasjon kan man finne et uttrykk for rotasjonsmatrisene for hver av rotasjonene om x-, y- og z-aksen, for så å multiplisere rotasjonsmatrisene sammen. Det er viktig å multiplisere rotasjonsmatrisene sammen i riktig rekkefølge, da total rotasjon er avhengig av hvilken rekkefølge matrisene multipliseres etter.



Figur 27: Rotasjoner rundt x-,y- og z-aksen

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix}, R_y(\beta) = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Matrise 2.1: Rotasjonsmatriser for rotasjon rundt x-,y- og z-akse

Total rotasjon blir da:
$R_{zyx} = R_z(\gamma)R_y(\beta)R_x(\alpha)$ hvor α, β, γ er rotasjonen rundt henholdsvis z-, y- og x -aksene

Det er viktig å holde kontroll på hvor aksene peker til en hver tid, i tillegg til å utføre rotasjonen i riktig rekkefølge for å oppnå den endelige rotasjonen som ønskes.

Kvaternioner er en mer kompakt og entydig måte å uttrykke samme rotasjon på, da kvaternioner trenger 4 tall og en rotasjonsmatrise trenger 9 tall.[5]
Rotasjon uttrykt som en rotasjonsmatrise:

$$R_i^{i-1} = R_x(\alpha)R_y(\beta)R_z(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix} \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ved å rotere et punkt $p_1 = [1, 0, 0]^T$ med $90^{\cdot}$ rundt y-aksen med rotasjonsmatrisen, blir resulterende punkt [2] :

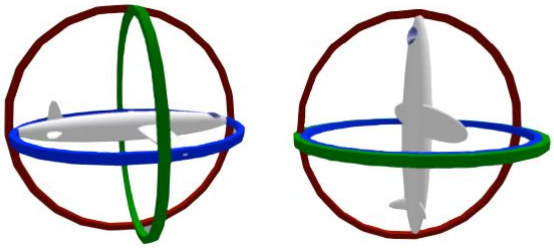
$$p_2 = R_i^{i-1}p_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} [1, 0, 0]^T = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} [1, 0, 0]^T = [0, 0, 1]^T$$

For å rotere et punkt p_1 med en rotasjon uttrykt i kvaternioner slik at vi får et nytt punkt, p_2, med ønsket orientering i rommet, brukes følgende formel [17] :

$$p_2 = q * p_1 * conj(q)$$

der p_1 er punktet som roteres og p_2 er resulterende punkt etter rotasjon, begge uttrykt som kvaternioner. q er en kvaternion som representerer rotasjonen, og conj(q) er den komplekskonjugerte til q. Det er også mulig å kombinere rotasjoner på en enkel måte når de uttrykkes som kvaternioner. En kombinasjon av rotasjonen q_1 og q_2 kan skrives som $q_3 = q_2 * q_1$.

En annen stor fordel ved å bruke kvaternioner er å unngår at det oppstår en såkalt "gimbal lock", som er en overlapping av to rotasjonsakser. Dette fører til at muligheten til å rotere over en av aksene forsvinner, og det er absolutt ikke ønskelig. Se figur 28. [18]



Figur 28: Gimbal lock [18]

Konvertering mellom kvaternioner og eulervinkler: For å konvertere eulervinkler til kvaternioner brukes disse formlene: [17] [19] [5]

$$
\begin{aligned}
q &= [q_w, q_x, q_y, q_z] \\
q_w &= c_1c_2c_3 - s_1s_2s_3 \\
q_x &= s_1s_2c_3 + c_1c_2s_3 \\
q_y &= s_1c_2c_3 + c_1s_2s_3 \\
q_z &= c_1s_2c_3 - s_1c_2s_3 \\
hvor& \\
c_1 &= cos(\frac{\beta}{2}), \quad c_2 = cos(\frac{\gamma}{2}) \\
c_3 &= cos(\frac{\alpha}{2}), \quad s_1 = sin(\frac{\beta}{2}) \\
s_2 &= sin(\frac{\gamma}{2}), \quad s_3 = sin(\frac{\alpha}{2})
\end{aligned}
$$

For å konvertere kvaternioner til eulervinkler brukes disse formlene: [17] [19] [5]

$$
\begin{aligned}
\gamma &= \tan^{-1}\left(\frac{2q_yq_w - 2q_xq_z}{1 - 2q_y^2 - 2q_z^2}\right)\frac{180}{\pi} \\
\beta &= \sin^{-1}\left(2q_xq_y + 2q_zq_w\right)\frac{180}{\pi} \\
\alpha &= \tan^{-1}\left(\frac{2q_xq_w - 2q_yq_z}{1 - 2q_x^2 - 2q_z^2}\right)\frac{180}{\pi}
\end{aligned}
$$

Unntak ved singulariteter:
Dersom $q_xq_y + q_zq_w = 0.5$:

$$
\begin{aligned}
\gamma &= 2\tan^{-1}\left(\frac{q_x}{q_w}\right)\frac{180}{\pi} \\
\alpha &= 0
\end{aligned}
$$

Dersom $q_xq_y + q_zq_w = -0.5$:

$$
\begin{aligned}
\gamma &= -2\tan^{-1}\left(\frac{q_x}{q_w}\right)\frac{180}{\pi} \\
\alpha &= 0
\end{aligned}
$$

Ved tilnærming av en bane til rådataen ved hjelp av algoritmen som er utviklet, blir orienteringen konvertert fra kvaternioner til eulervinkler. Dette er fordi eulervinkler er

et mer intuitivt mål på orienteringen, som er enklere å forstå og som samtidig gir en lineær utvikling i orientering fra punkt til punkt. Dette fører igjen til at det er enklere å finne ut hvor mye feil det er for orienteringen. Selv mellom to punkter i banen, der det ikke finnes noe mål på orienteringen direkte. Orienteringen finnes da ved hjelp av interpolering. Dette gjøres ved at orienteringen fra punkt til punkt i banen blir sett på som en rett linje, og orienteringen i punktene i rådatabanen projiseres ned på linjene i banen. Se del 2.6.3.

3 Verktøy og utstyr

3.1 Rådatabaner og ABB-baner

Ettersom ABB allerede har utviklet en algoritme for SRP-produktet er det tilgjengelig flere testinnspillinger av rådata som ABB har tilpasset baner til, med sin egen algoritme. Disse banene, i kombinasjon med rådataene brukes til å sammenligne resultatene generert med algoritmen i denne rapporten. Videre i rapporten vil disse banene bli kalt *referanse-banene*.

3.2 Matlabkode fra veileder

Karl Skretting ved Universitetet i Stavanger står som faglig ansvarlig og veileder for denne oppgaven. Han har laget noen matlabfunksjoner som ble gjort tilgjengelig for undertegnede ved begynnelsen av semesteret. Disse funksjonene er *initP.m*, *fitP.m*, *fitMiddle.m*, *fitMiddle2.m*, *dist2lines.m*, *fitOneEnd.m*, *fitTwoLines.m*. Se del Vedlegg A . Store deler av koden i algoritmen presentert i denne rapporten og tilhørende funksjoner stammer fra disse matlabfilene.

3.3 Robot

Banen fra algoritmen skal kunne brukes på en hvilken som helst robot, og roboten skal kunne utføre en hvilken som helst oppgave. Det er derfor ikke spesifisert hvilken robot som skal brukes. Robotene kan ha svært forskjellig dynamikk, og dynamikken er også avhengig av hvilken last roboten bærer på. Det skal av den grunn ikke fokuseres på dynamikken til roboten. Det tenkes at roboten følger den genererte banen optimalt, uten avvik.

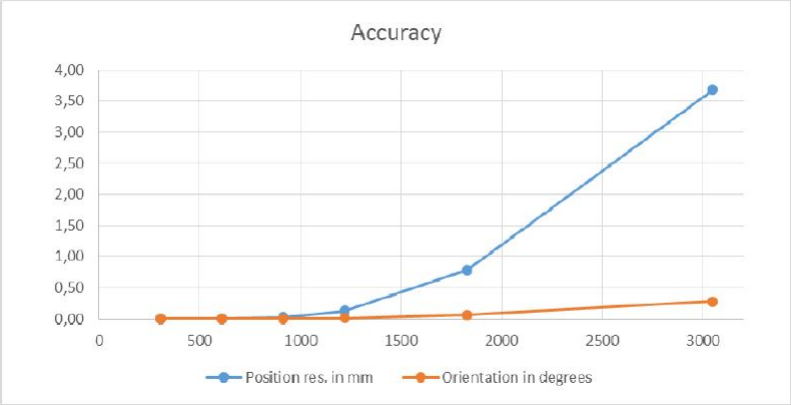
3.4 Sensor

Sensoren som er brukt for å spille inn rådatabanene er en Polhemus 6 punkts elektromagnetisk bevegelsessensor. Rådataen fra Polhemus-sensoren består av et tidsstempel, koordinater i x-, y- og z-retning og orientering uttrykt i kvaternioner. [20]

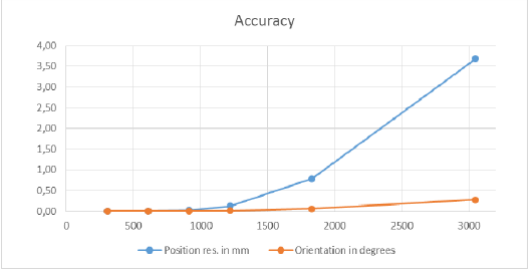
Sensoren er montert på et verktøy som kalles 'SRP Teach Handle' og etterligner en tradisjonell spraypistol. Det er dette verktøyet brukeren lager robotbanen med. [21]

Sensoren i seg selv er ikke så viktig i denne rapporten, da algoritmen skal kunne tilpasse en bane til et sett med rådata, uavhengig av hvilken sensor som brukes. Det eneste kravet er at sensoren registrerer de samme dataene som Polhemus-sensoren, dvs. tid, posisjon og orientering i rommet.

Figur 29: SRP Teach Handle [21]



Figur 30: Nøyaktighet på SRP Motion Tracker [22]

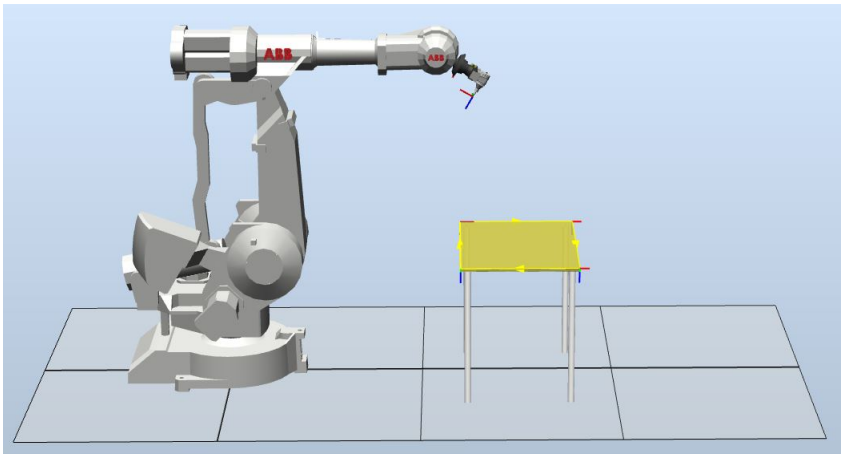


Figur 31: Nøyaktighet på SRP Motion Tracker [22]

3.5 Rapid-stasjon

Det er laget en stasjon med en robotcelle i *Robotstudio*. Denne er brukt til å simulere rådatabanen og banen fra algoritmen i denne rapporten. Det er også laget en funksjon *LagRapidBane.m* som lager en *RAPID*-kode av banen fra algoritmen, slik at banen enkelt kan simuleres i *Robotstudio*. Det er gjennom simuleringen bekreftet at punktene og orienteringen i banen er riktig konfigurert og kjørbar. Dette avhenger dog av hvilket arbeidsobjekt som er laget og brukes i *Robotstudio*, samt hvilken robot som brukes under

simuleringen. Dersom banen ikke er kjørbar direkte fra *RAPID*-koden, er dette på grunn av begrensning i rekkevidden til roboten, og banen må da flyttes noe i forhold til arbeidsobjektets koordinatsystem. Gjennom simuleringen er det bekreftet at både posisjon og orientering i banen blir som forventet. Hastighetsfølgingen i banen er selvfølgelig avhengig av spesifisert sonedata for punktene i banen.



Figur 32: Stasjon med en IRB4400-robot og et bord som arbeidsobjekt

4 Implementering

4.1 Start av algoritmen

Algoritmen benytter seg av ulike funksjoner for å tilpasse en bane, kun bestående av rette linjer ved å minimere en vektet sum av kvadrert feil. Algoritmen kalles med [banen,raadata,sjekk] = Algoritme5(vekt, margin, lengde, RAPID, raadata) og lar brukeren spesifisere et sett parametere som bestemmer utfallet av algoritmen. Endelig bane, rådatabane og en sjekk-variabel returneres når algoritmen er ferdig. I tillegg lagres en *RAPID*-kode av banen i samme mappe som .m-filen til algoritmen er plassert i, dersom RAPID-parameteren er satt til 1. Sjekk er en struct-variabel som returneres fra algoritmen og inneholder mål på banens godhet underveis i utviklingen av banen. Sjekk-variabelen får en ny verdi for hver gang antall punkt i banen reduseres.

- vekt - En 1x3 rekkevektor som bestemmer hvilken egenskap ved banen som skal optimaliseres
- margin - Et tall som bestemmer hvor stor økning i avvik som tillates i banen ved reduksjon av antall punkter
- lengde - Ønsket antall punkt i banen. Overstyrer parameteren margin
- raadata - En matrise som inneholder alle punktene i rådatabanen
- RAPID - En parameter som bestemmer om det skal genereres en *RAPID*-kode av banen

Vekt-parameterens høyeste tall spesifiserer hvilken egenskap ved banen som skal optimaliseres. Dersom tallet plasseres i første kolonne, vil posisjon favoriseres. Dersom tallet plasseres i andre kolonne vil orientering favoriseres, og tredje kolonne vil favorisere hastighet. Vekt = [posisjon, orientering, hastighet]. Vekten blir multiplisert med den kvadrerte feilen som blir funnet i banen, men kun for den egenskapen som det velges å optimalisere for. Metoden kalles vektet sum av kvadrert feil, se del 2.4.3. Vekt = [1 1 1] vil gi lik vekting for alle egenskaper ved banen. Vekt = [1 5 1] vil multiplisere kvadrert feil for orienteringen i banen med faktoren 5. Da algoritmen ikke kan optimalisere for hastighet direkte, vil algoritmen, dersom den skal optimalisere banen for hastighet, vektlegge både tid og posisjon. Hastighet er som kjent avhengig av endring i posisjon over tid, så et minimalt avvik i tid og posisjon vil gi et minimalt avvik i hastighet.

Margin må spesifiseres som et tall større enn 1 dersom denne parameteren vil benyttes. Dette tallet bestemmer hvor stor økning i total sum av kvadrert feil som tillates i banen, sett i forhold til feilen fra første utkast av en bane. Det første utkastet av en bane fra algoritmen inneholder mange flere punkter enn nødvendig, og vil følgelig også ha en veldig god nøyaktighet. Se del 4.6 for en beskrivelse av hvordan dette første utkastet lages. Velges f.eks. margin = 1.2, vil det tillates en 20% økning i feilen før algoritmen avsluttes og endelig bane returneres.
Egenskapen som måles for å avgjøre om økningen i feil er under ønsket margin er gitt

av vekt-parameteren. Dersom det velges å ikke vektlegge noen egenskaper i banen, vil økningen i avvik for posisjon avgjøre om avviket er under ønsket margin.

Lengde er et tall som bestemmer hvor mange punkter som ønskes i banen. Antall punkter i banen vil reduseres med 5% frem til ønsket antall punkter oppnås. Det tas ikke hensyn til margin-parameteren dersom lengde spesifiseres. Dersom det ikke ønskes å spesifisere en ønsket lengde på banen, settes lengde-parameteren lik 0.

Raadata er en matrise som inneholder alle punktene i rådatabanen. Dersom denne parameteren ikke spesifiseres, vil brukeren få opp en filvelger som brukes til å finne en rådatafil av typen .Mod eller .Raw.

RAPID-parameteren bestemmer om algoritmen skal generere en *RAPID*-kode av banen som blir returnert. Parameteren settes lik 1 dersom det ønskes å generere en *RAPID*-kode, og 0 ellers.

Se del Vedlegg A for informasjon om funksjonene som brukes i algoritmen.

4.2 Hente data fra rådatafilene

Det er gitt tre forskjellige filtyper for hver av de N banene som er spilt inn av ABB. $raw.N$.log, MN.Mod og $RawN$.Mod. Disse må leses inn før en bane kan genereres i algoritmen og bli sammenlignet med referansebanen. .log-filene er formatert som en tekstfil med kommaseparerte verdier. .Mod filene er formatert som tekstfiler med RAPID-format.

Lese data i rådatafil fra Polhemus-sensoren: I figur 33 ses et utdrag av rådataen fra Polhemus-sensoren. Hver rekke er èn måling og rådataen består ofte av veldig mange målinger. Disse filene er av filtype .log
I første kolonne finnes tidsstempelet, kolonne 2-4 er henholdsvis x-, y-, og z-koordinater og kolonne 5-8 er orienteringen uttrykt i kvaternioner. Det er skrevet en funksjon, *lesLog.m* for å lese inn dataene fra en slik rådatafil og organisere dataen slik som dataen ønskes organisert i Matlab. Disse rådataene kan det tilpasses en bane til med algoritmen.

```
1540012, 719.734558105469, -97.2053146362305, -480.808166503906, -0.610407650470734, 0.124314948916435, -0.749075055122375, 0.225465580821037, 0, 0
```

Figur 33: En linje fra en rådatafil

Lese data fra .Mod-fil med RAPID-formatering: Rådatafilene fra Polhemus-sensoren er brukt av ABB for å lage en bane med deres algoritme. Rådatafilene blir av ABB rotert og overført til et koordinatsystem tilhørende et arbeidsobjekt som blir definert når banen spilles inn. Dataen i rådatafilene med filtype .Mod er derfor noe annerledes enn i rådatafilene med filtype .log. Da dette er tilfellet må banene fra ABB sammenlignes med rådatabanene fra .Mod filene.

Rådatafilene og banene fra ABB inneholder posisjon i x,-y og z-retning og orientering i rommet, uttrykt i kvaternioner. Filene er av typen .Mod, som er ferdige programfiler som brukes i robotstudio. Formateringen i filene er da et *RAPID*-format, som fører til at det

trengs en egen funksjon for å lese inn dataen i disse.
Det er skrevet en funksjon, *lesMod.m* som leser inn data fra .Mod filene og organiserer dataen slik den ønskes organisert i Matlab. .Mod-filene er delt inn i to deler, hvor første del definerer punktene med posisjon og orientering, mens den andre delen inneholder *main*-funksjonen som utfører bevegelsene. Det er i *main*-funkjonen hastigheten, eller tidsbruken for bevegelsene spesifiseres. Se figur. 34 og 35 for et utsnitt av en rådatafil av type .mod.

```
module M1
  !
  ! Sensor WObj Frame : [ 1124.06 160.87 -199.9 ]
[ 0.515496674437117
0.472968370661527 -0.520042167536454 -0.490020655668759 ]
  ! Sensor Tool Frame : [ -322.4 0 0 ] [ 1 0 0 0 ]
  !
  Local Const robtarget p0 :=
[[349.5059,924.2835,425.4585],[0.195888182433874,0.69373051780106
,0.673120452632127,-0.165150051563343],[0,0,0,0],[0,0,0,0,0,0]];
  Local Const robtarget p200 :=
```

Figur 34: Utdrag fra starten av en datafil fra ABB

```
proc mainM1()
  MoveAbsJ jReadyPos, v500, z50, toolSRP \WObj:=wobjSRP;
  SetBrush 1;
  PaintL p0, v500, z50, toolSRP \WObj:=wobjSRP;
  PaintL p200, v500\T:=0.20, z50, toolSRP \WObj:=wobjSRP;
```

Figur 35: Utdrag fra en datafil fra ABB, main-metode

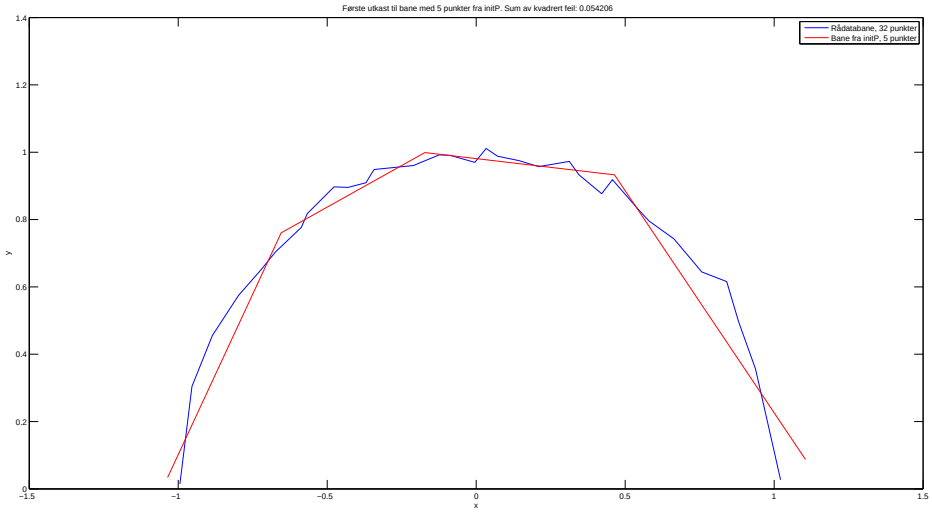
4.3 Behandling av rådata

Før algoritmen begynner å tilnærme en bane til rådataen, blir orienteringen i alle punkt konvertert fra kvaternioner til eulervinkler med funksjonen *quaternionMath.m*, se del A.14. Dette gir et mer intuitivt mål på rotasjonen fra punkt til punkt og gir i tillegg en lineær endring i orientering fra punkt til punkt i banen. Da konverteringen fra kvaternioner til eulervinkler gir et resultat i området $[-180°, 180°]$, er det nødvendig å foreta en korreksjon av vinkler som er $> 180°$ og $< -180°$. Dette gjøres i funksjonen *korrigerData.m*, se del A.29, hvor endringen i orientering fra punkt til punkt undersøkes. Dersom endringen i orientering er større enn $\pm 180°$, korrigeres orienteringen ved å legge til $\pm 360°$. Se også del A.29 for en illustrasjon av problemet som oppstår ved konvertering fra kvaternioner til eulervinkler og hvordan resultatet blir når feilen korrigeres.

4.4 Danne utgangspunkt for første utkast av en bane

Det første som blir gjort i algoritmen er å kalle funksjonen *initP2.m*, se del A.10 . Denne funksjonen deler rådatabanen inn i et passende antall grupper, hvor punktene i gruppen er de punktene som ligger nærmest hverandre i tid. Antall grupper blir bestemt av hvor mange punkter som ønskes i første utkast av banen. Hvor mange punkter som er optimalt for banen er vanskelig å bestemme da hver rådatabane kan være veldig forskjellig fra de andre. Ved forsøk viser det seg at det er mulig å lage en god bane med ca. 5% av punktene i rådataen. I algoritmen lages likevel første utkast til banen med 10% av antall punkter

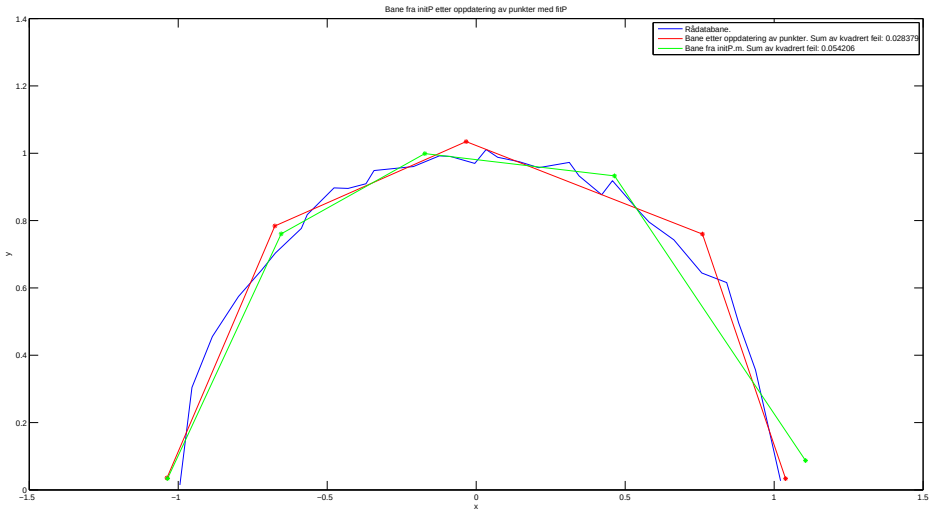
i rådatabanen for å få flere punkter å arbeide med, og for å få bedre plasserte punkter i banen. For hver gruppe dannes det en linje fra startpunktet til endepunktet i gruppen og endepunktene for linjene optimaliseres med funksjonen *fitOneEnd2.m*. Dette gjøres ved å ta utgangspunkt i startpunktet p_1 og punktene i rådataen som ligger mellom p_1 og endepunktet for linjen, p_2. Endepunktet p_2 flyttes så for å finne det punktet som, ved en linje fra p_1 til p_2 best passer til punktene i rådataen. Som avstandsmål brukes kvadrert 2-norm. Det samme gjøres så for startpunktet p_1.
Resultatet blir en grov tilnærming til en bane som passer til alle rådatapunktene. Denne har gjerne betydelige feil, men gir et godt utgangspunkt for videre arbeid. Se fig. 36 for et eksempel utført på en enkel bane.



Figur 36: En bane tilpasset noen rådatapunkter med funksjonen *initP.m*

4.5 Oppdatere punktene fra første utkast i banen

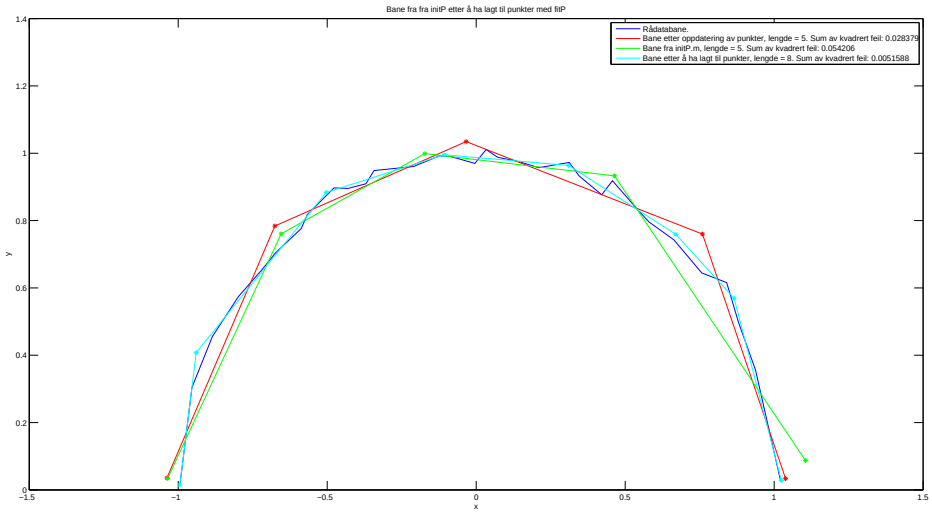
Det første utkastet til en bane fra funksjonen *initP2.m* er kun en grov tilnærming til en endelig bane. Punktene er ikke plassert optimalt, og det er betydelige feil i posisjon, hastighet og orientering. Neste steg i algoritmen er å se om noen av punktene kan flyttes for å forbedre vektet sum av kvadrert feil i banen, se del 2.4.3 . Dette gjøres med funksjonen *updateP.m*. Funksjonen kaller tre andre funksjoner, *fitOneEnd2.m*, *fitMiddle2_2.m* og *dist2lines2.m*. Se del Vedlegg A for informasjon om funksjonene.
Først brukes *dist2lines2.m* som beregner hvor stor den vektede summen av kvadrert feil er for banen, før noen endringer blir gjort. Den returnerer også en parameter t som forteller hvor punktene i rådataen er plassert i forhold til punktene i banen, sett i forhold til tid. Rådatapunktene som er plassert mellom første punkt i banen, p_1 og andre punkt i banen, p_2 får en t-verdi i området [1,2], hvor verdien bestemmer hvor nært rådatapunktet ligger punktet i banen. $t = 1.5$ tilsvarer et punkt som ligger midt mellom p_1 og p_2, $t = 1$ tilsvarer et punkt som ligger nøyaktig ved p_1 osv.
For første og siste punkt i banen, p_1 og p_n, brukes funksjonen *fitOneEnd2.m* for å bestemme om det vil gi en ønskelig endring i vektet sum av kvadrert feil ved å erstatte endepunktet i banen og endepunktets nærmeste punkt, p_2 og p_{n-1} med et nytt punkt. For de resterende punktene i banen plukkes det ut 3 og 3 punkter, p_{k-1}, p_k og p_{k+1}, som sammen danner to sammenhengende linjer i banen. Funksjonen *fitMiddle2_2.m* bestemmer om knutepunktet p_k kan erstattes av et nytt punkt for å forbedre vektet sum av kvadrert feil for rådatapunktene som ligger mellom p_{k-1} og p_{k+1}. Alle punktene i banen sjekkes med denne metoden, og den oppdaterte banen returneres til slutt. For hver gang alle punktene i banen flyttes kan en ny forflytning av punktene gi en ønsket innvirkning på vektet sum av kvadrert feil. Prosessen gjentas derfor helt til endringen i vektet sum av kvadrert feil er større eller tilnærmet lik 0, eller til prosessen har gjentatt seg selv maksimalt 40 ganger. Se fig. 37 for et eksempel på utviklingen i banen ved bruk av *updateP.m*.



Figur 37: Endring i bane fra initP.m etter oppdatering av punkter med updateP.m

4.6 Utvidelse av banen

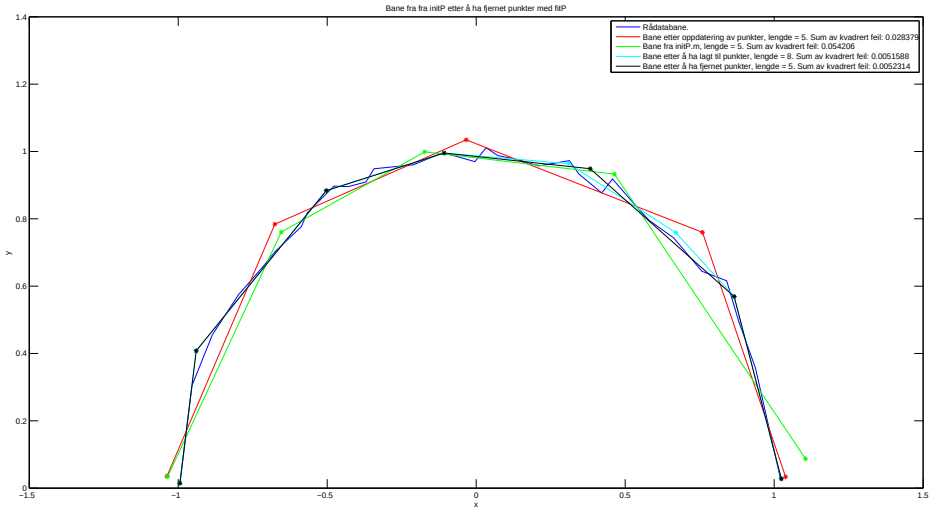
Banen blir så utvidet til å inneholde 50% flere punkter ved hjelp av funksjonen *fitP2.m*, se del A.11. Funksjonen ser på to og to nærliggende punkter i banen, og plasserer et nytt punkt et sted mellom disse punktene, dersom det fører til en positiv endring i vektet sum av kvadrert feil. De nye punktene blir funnet ved hjelp av funksjonen *fitMiddle2_2.m* som skalerer og roterer punktene i banen før den kaller funksjonen *fitMiddle_2.m*, se del A.12 og del A.13.
Ved å legge til flere punkter i banen før antall punkter reduseres igjen, håpes det at de punktene som blir lagt til passer bedre til rådatabanen enn de punktene som allerede eksisterer i banen. Dette legger til rette for at det skal blir enklere å fjerne punkter i banen, uten å gå for mye på bekostning av den vektede summen av kvadrert feil i banen. Dette henger sammen med at oppdateringsfunksjonen *updateP.m* A.18 har en begrenset innvirkning på plassering av punktene i banen. Punktene i banen oppdateres også på samme måte som i del 4.5, etter at banen har blitt utvidet.



Figur 38: Utvikling i bane ved å legge til punkter

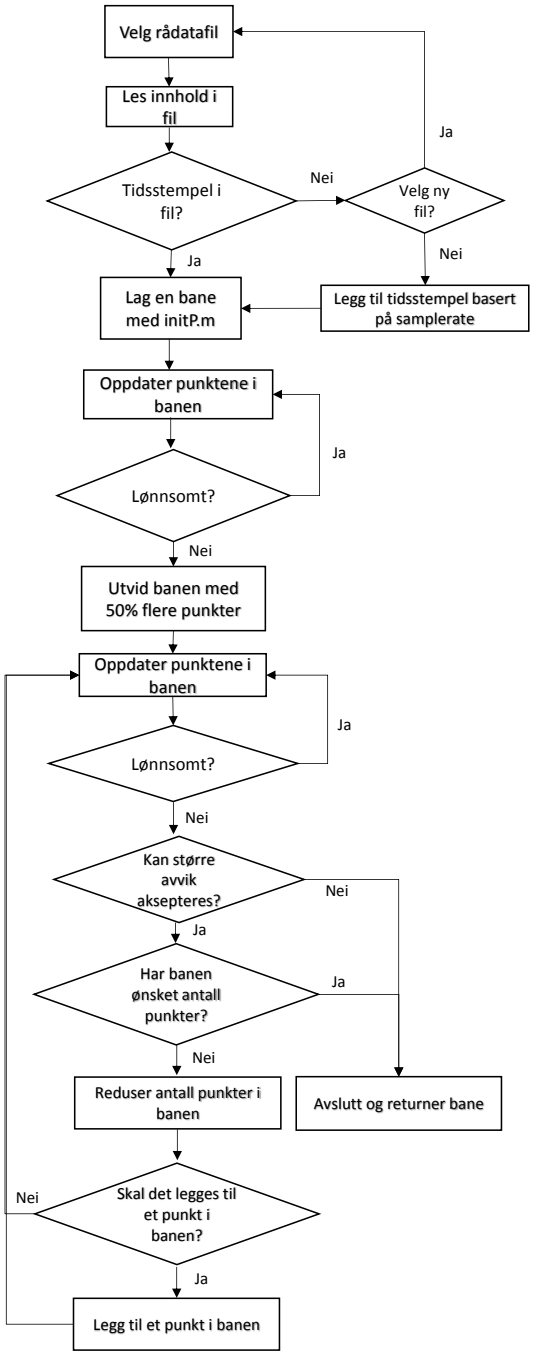
4.7 Reduksjon av punkter i banen

Målet med algoritmen er å returnere en bane med så få punkter som mulig, samtidig som nøyaktigheten i forhold til rådatabanen beholdes. Det må derfor fjernes punkter fra banen frem til vi har et ønsket antall punkter i banen, eller feilen i banen blir for stor. Dette blir gjort med funksjonen *decreaseP.m*, se del A.20 . Funksjonen er lagt inn i en løkke som fortsetter å kalle funksjonen og gir en reduksjon på 5% av punktene i banen for hver iterasjon. Dette pågår frem til en av to kriterier ikke lenger er oppfylt. Disse kriteriene bestemmes gjennom parametrene *margin* og *lengde*, som spesifiseres av brukeren når algoritmen startes. Margin sier hvor stor økning i feil i forhold til den utvidede banen i 4.6 som tolereres, og lengde sier hvor mange punkter som ønskes i banen. Dersom lengde er spesifisert, vil denne parameteren overstyre parameteren margin. I tillegg vil algoritmen legge til et ekstra punkt i banen hver gang antall punkter i banen reduseres med 5%. For det tilfellet når 5% av antall punkt i banen tilsvarer 1 punkt, blir det lagt til et ekstra punkt i banen for hvert tredje fjernede punkt. Det viser seg at dette gir en positiv innvirkning på vektet sum av kvadert feil når antall punkter i banen reduseres ytterligere. Grunnen til dette er at det dannes nye kombinasjoner av punkter som er mer optimalt plassert enn de gamle kombinasjonene. Dersom ønsket lende på banen er oppnådd, eller økningen i feilen blir større enn tillat margin, avsluttes løkken og den endelige banen returneres. For hver gang antall punkt i banen reduseres, blir alle punktene i banen oppdatert som i del 4.5.



Figur 39: Resulterende bane etter fjerning av punkter

4.8 Flytdiagram for algoritme



Figur 40: Flytdiagram for algoritme

5 Resultat

5.1 Begrensninger og antagelser

Det er valgt å ikke glatte dataen fra rådatafilene, selv om det er meget sannsynlig at dataene inneholder støy og unøyaktige målinger. Det ville sannsynligvis forbedret resultatene fra algoritmen dersom dataene ble glattet. Denne beslutningen er tatt etter en samtale med ABB, der det ble bestemt at algoritmen skal kunne brukes til å generere en bane, uavhengig av hva banens formål er. Dersom banen er ment for å implementeres på for eksempel en sveiserobot, vil det ikke være ønskelig å glatte dataen, da banen vil inneholde sagtann-lignende bevegelser som vil bevares. Dersom banen glattes vil sagtann-bevegelsene reduseres, eller fjernes helt. Det er i tillegg ikke bestemt hvilken sensor som skal brukes for å generere banen, noe som gjør det vanskelig å gjøre noen antagelser om støyen i sensoren.
Brukeren av algoritmen må selv bestemme om dataen skal glattes før den brukes i algoritmen. Dette må gjøres ut fra informasjon om hvilket formål banen skal brukes til og hvor god nøyaktigheten i sensoren er. Det legges også til grunn at roboten klarer å følge hastigheten i banen optimalt. Dette vil i realiteten avhenge av robotens dynamikk, hvilken last den bærer og valg av sonedata for bevegelsen.

5.2 Sammenligning av baner fra algoritmen og referansebanene fra ABB

I denne delen ønskes det å illustrere mulighetene ved algorimen og de forskjellige resultatene ved ulik bruk av algoritmen. Forsøkene er utført på bane nr.2, *Raw2.mod*, innspilt av ABB 05.11.2013. Dette er en bane som inneholder 1281 punkter og er relativ lang i forhold til de andre banene. Banen har også en god del endringer i orientering, hastighet og posisjon, som gjør dette til en god bane å utføre forsøk på. Det er også gjort forsøk på de andre banene, som illustrerer utviklingen av avviket i banen ved ulike valg av vekt-parameter i algoritmen. På grunn av at det produseres et betydelig antall figurer for hvert forsøk er det laget et eget vedlegg som inneholder de øvrige resultatene. Her finnes figurer for alle de 7 banene som ble innspilt 05.11.2013. Resultatet for hver bane er tilsvarende det samme som for bane 2, og det er derfor ikke nødvendig å vise resultatene for de resterende banene i denne rapporten. Se vedlegg Vedlegg B for resultat for bane 2 til bane 8. For hver bane som blir produsert med algoritmen i denne delen vil flere figurer og mål på godheten i banen presenteres:

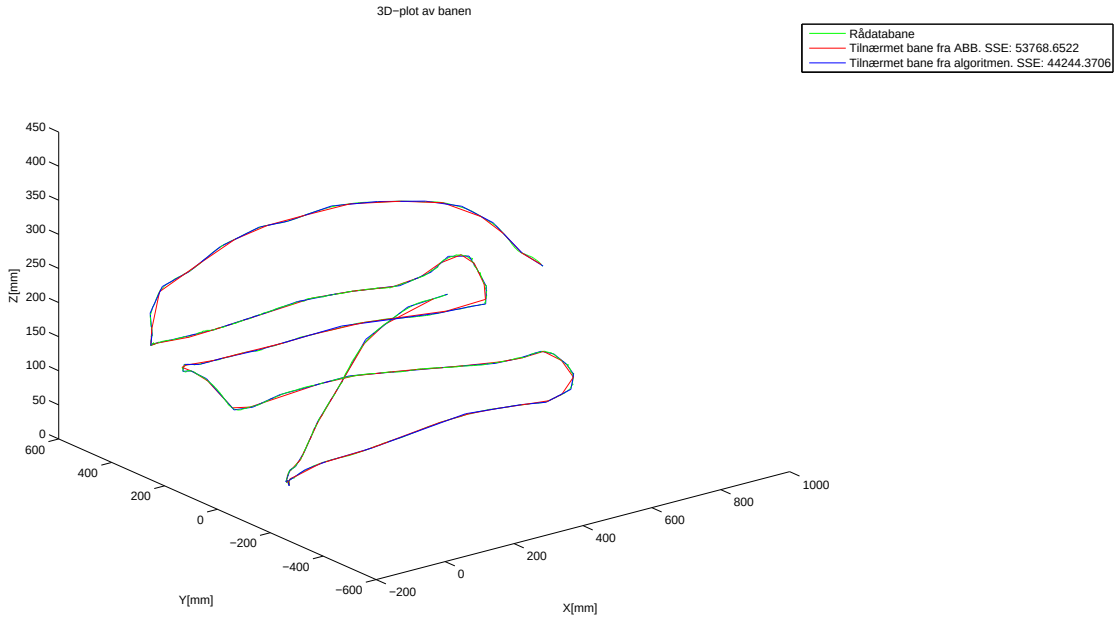
- 3D-plot av banen
- 2D-plot av posisjon
- 2D-plot av hastighet
- 2D-plot av orientering
- Plot av lengdeforhold
- Plot av orienteringsforhold
- Sum av kvadrert feil - for hele banen, med alle dimensjonene i dataen
- Sum av kvadrert feil - posisjon $[mm^2]$
- Sum av kvadrert feil - hastighet $[\frac{m^2}{s^2}]$
- Sum av kvadrert feil - orientering $[grader^2]$
- Gjennomsnittlig avvik posisjon $[mm]$
- Gjennomsnittlig avvik hastighet $[m/s]$
- Gjennomsnittlig avvik orientering $[grader]$
- Maksimalt avvik posisjon $[mm]$
- Maksimalt avvik hastighet $[m/s]$
- Maksimalt avvik orientering $[grader]$
- Lengdeforhold [%]
- Orienteringsforhold [%]

Alle målene på avvik er sett i forhold til rådatabanen. Lengdeforhold er forholdet mellom summen av lengden på alle vektorene i banen og summen av lengden for alle vektorene i

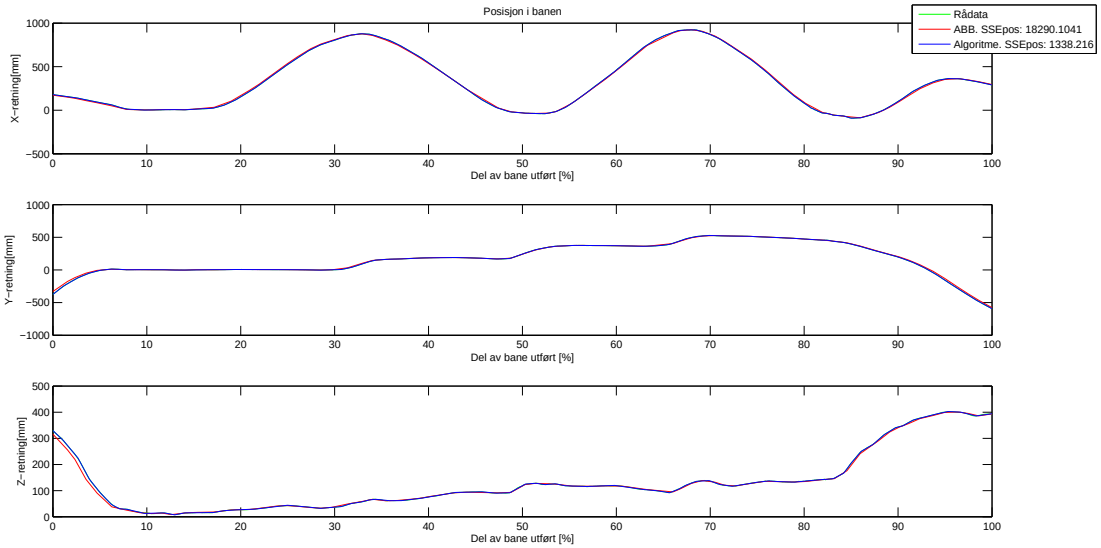
rådatabanen. Denne verdien forteller hvor stor del av posisjonsendringen i rådatabanen som blir representert med banen og blir målt i prosent av endringen i rådatabanen. Orienteringsforhold er forholdet mellom summen av absolutt orientering fra punkt til punkt i banen, og summen av absolutt orientering fra punkt til punkt i rådatabanen. Denne verdien forteller hvor stor del av endringen i orienteringen i rådatabanen som blir representert med banen og blir målt i prosent av endringen i rådatabanen.

5.2.1 Resultat fra algoritmen uten å spesifisere hvilken egenskap banen skal optimaliseres for. Banen inneholder like mange punkter som referansebanen

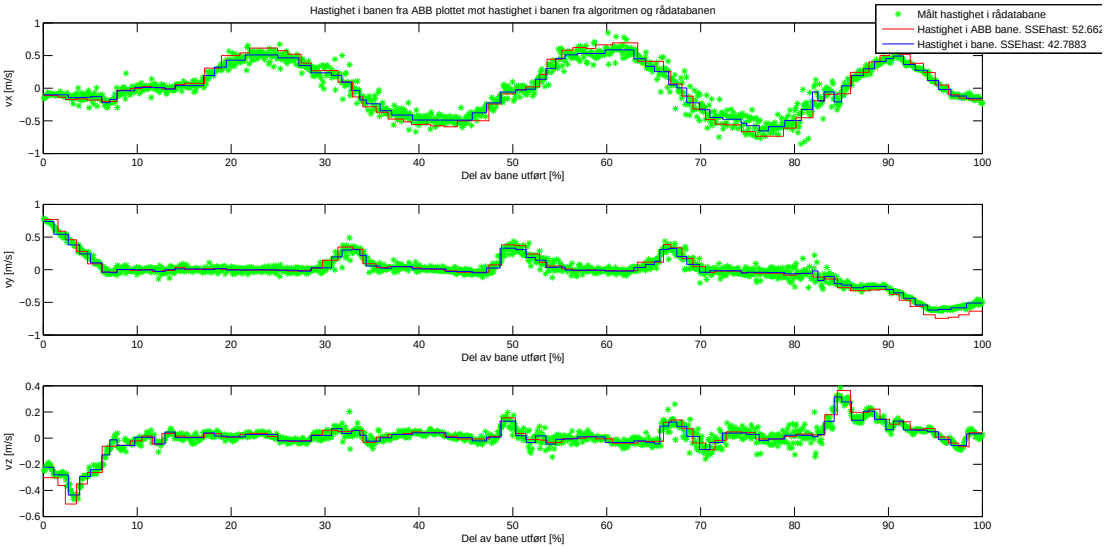
Her illustreres resultatet fra algoritmen ved å spesifisere vekt-parameteren som vekt = [1 1 1], som betyr at alle egenskapene i banen blir vektlagt like mye. Dette tilsvarer å bruke sum av kvadrert feil i algoritmen, i stedet for vektet sum av kvadrert feil. Se tabell 1 for ulike mål på banens godhet. Som tabellen viser er avviket i banen vesentlig mindre enn avviket i referansebanen, for alle egenskaper ved banen.



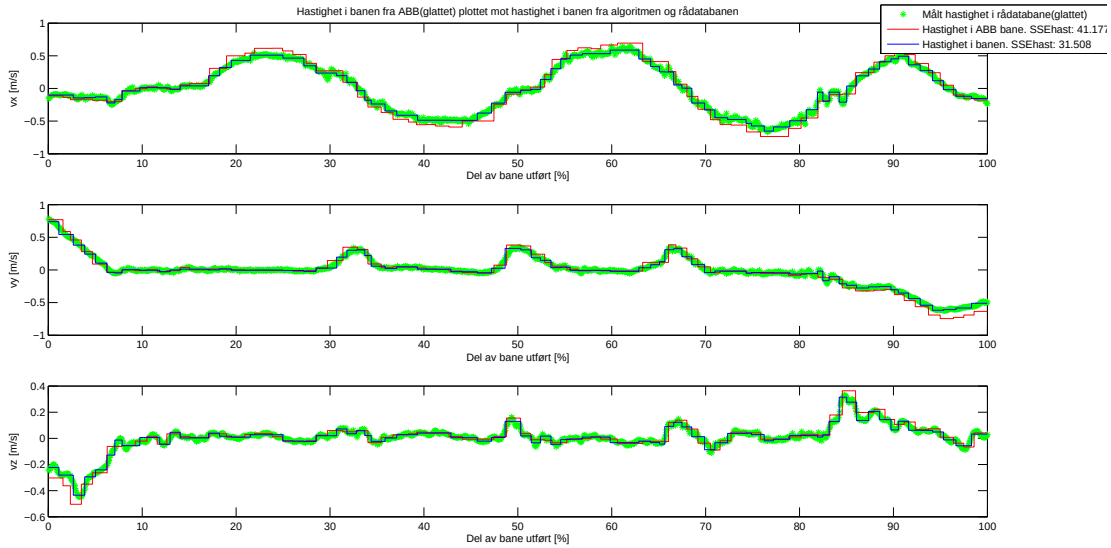
Figur 41: 3D-plot av bane



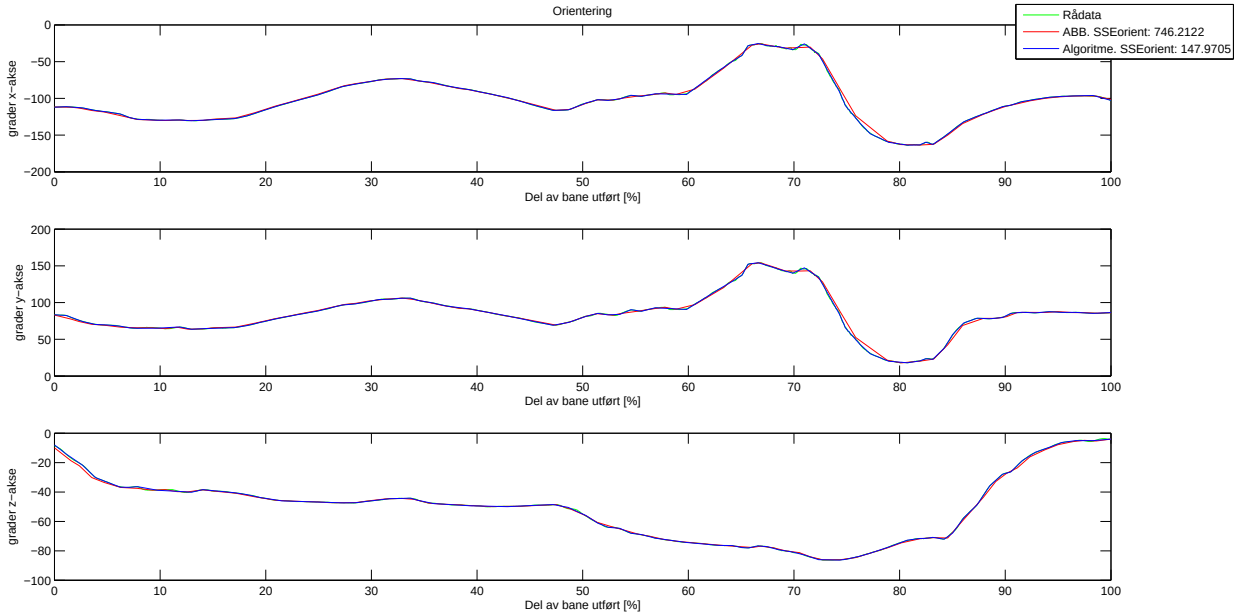
Figur 42: 2D-plot av posisjon i bane



Figur 43: 2D-plot av hastighet i bane



Figur 44: 2D-plot av hastighet i bane mot glattet hastighet i rådatabane



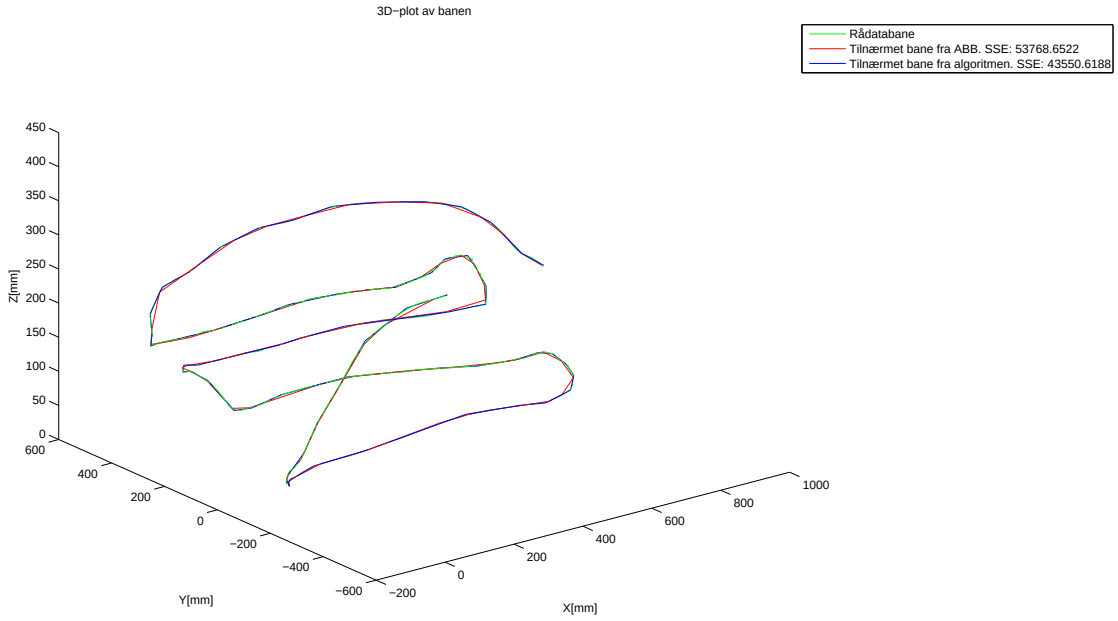
Figur 45: 2D-plot av orienteringen i bane

Måling	Bane fra algoritme	Referansebane
Sum av kvadrert feil for alle dimensjonene i dataen	44244	53769
Sum av kvadrert feil posisjon	1338,2	18290
Sum av kvadrert feil hastighet	42,7883	52,6629
Sum av kvadrert feil orientering	147,9705	746,2122
Gjennomsnittlig avvik posisjon	0,8556	2,1286
Gjennomsnittlig avvik hastighet	0,1328	0,1540
Gjennomsnittlig avvik orientering	0,2570	0,4656
Maksimalt avvik posisjon	4,3774	48,1178
Maksimalt avvik hastighet	0,5945	0,5799
Maksimalt avvik orientering	1,6288	4,7551
Lengdeforhold	98,1356	96,1622
Orienteringsforhold	94,4283	89,7764
Antall punkter i banen	72	72

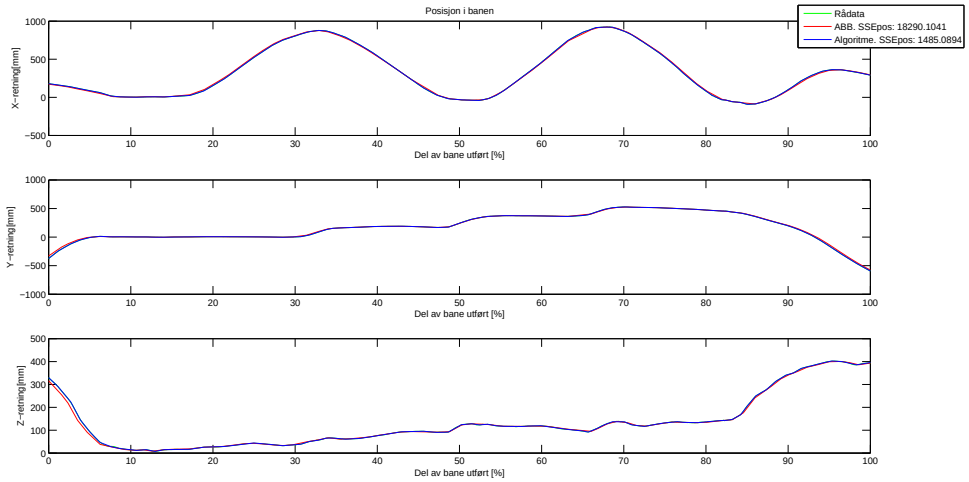
Tabell 1: Mål på godhet for referansebane og bane utviklet med algoritme, uten å spesifisere hvilken egenskap banen skal optimaliseres for

5.2.2 Resultat fra algoritmen ved å optimalisere for orientering i banen. Banen inneholder like mange punkter som referansebanen

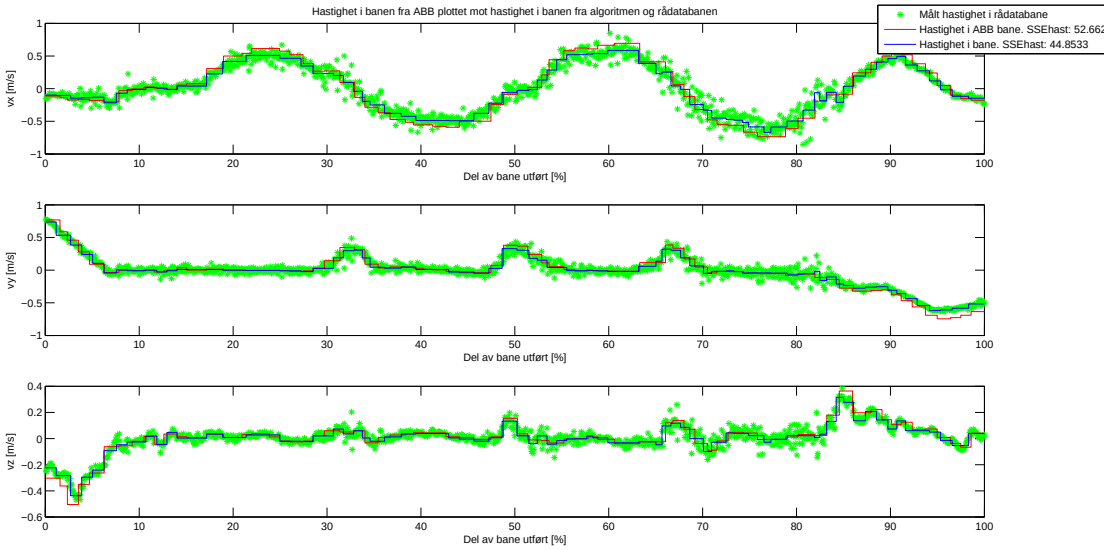
Her illustreres resultatet fra algoritmen ved å spesifisere vekt-parameteren som vekt = [1 5 1]. Dette gir en optimalisering av orienteringen i banen ved å tillate mer feil for de resterende egenskapene. Sum av kvadrert feil for orientering blir multiplisert med faktoren 5, som fører til at algoritmen vil minimere avviket for orienteringen og tillate mer avvik for de resterende egenskapene i banen. En kan sammenligne dette resultatet med resultatet i tabell 1, hvor det er valgt å ikke optimalisere banen for en gitt egenskap. Som resultatet i tabellen 2 viser, er det tydelig at avviket for orienteringen i banen er redusert, men da på bekostning av avviket i posisjon og hastighet. Vekt-parameteren er satt til [1 5 1] kun for å illustrere hva resultatet fra algoritmen blir dersom den optimaliseres for orientering. Det kan godt tenkes at andre valg av vekter som ikke favoriserer orientering i så stor grad er å foretrekke i en reell situasjon.



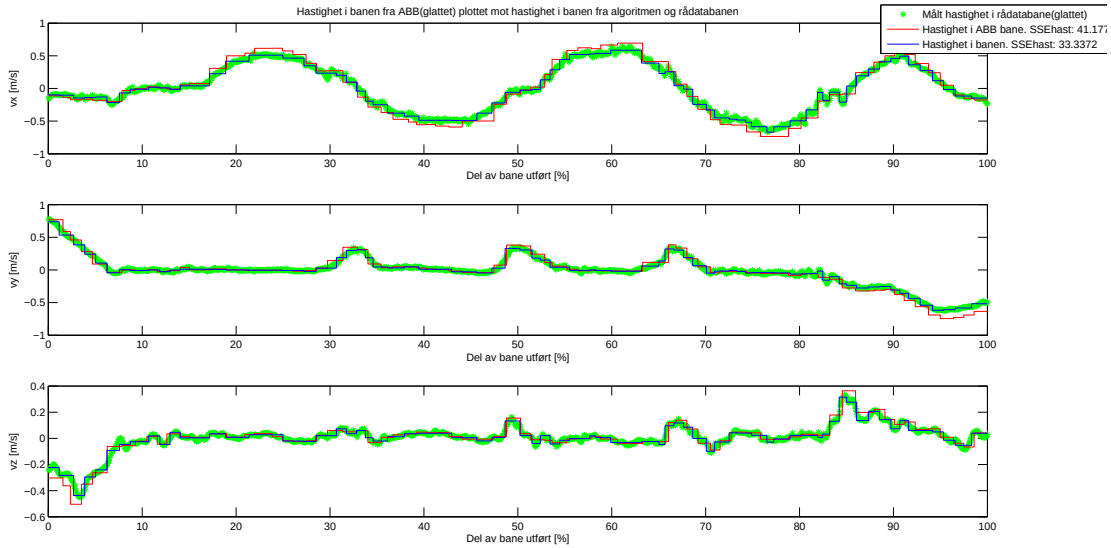
Figur 46: 3D-plot av bane



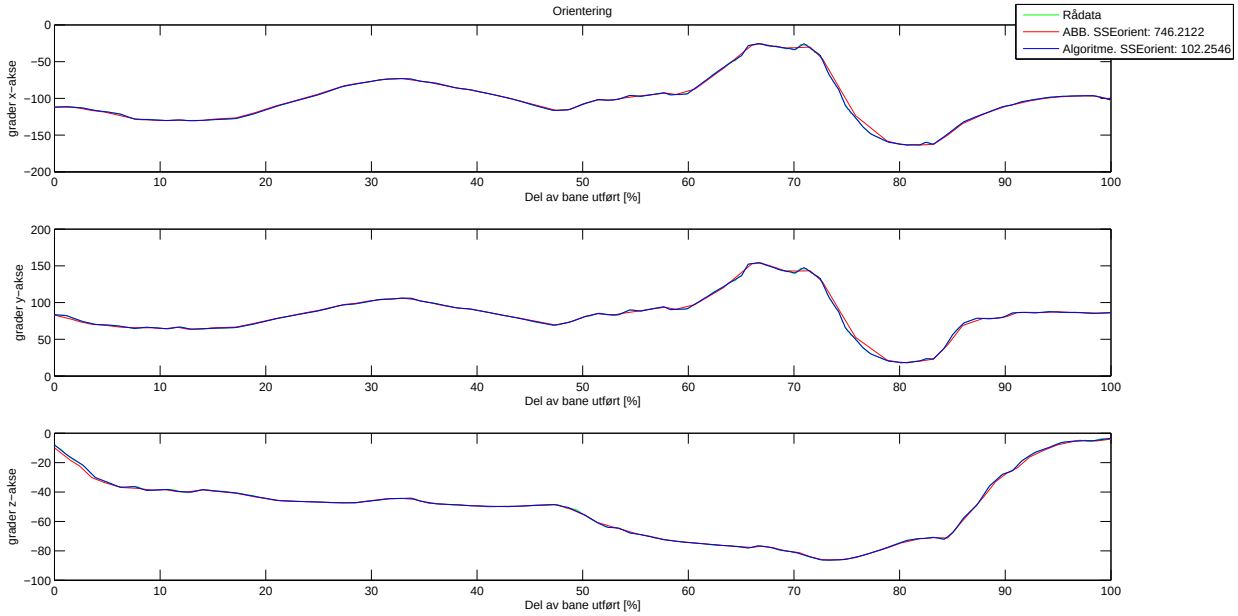
Figur 47: 2D-plot av posisjon i bane



Figur 48: 2D-plot av hastighet i bane



Figur 49: 2D-plot av hastighet i bane mot glattet hastighet i rådatabane



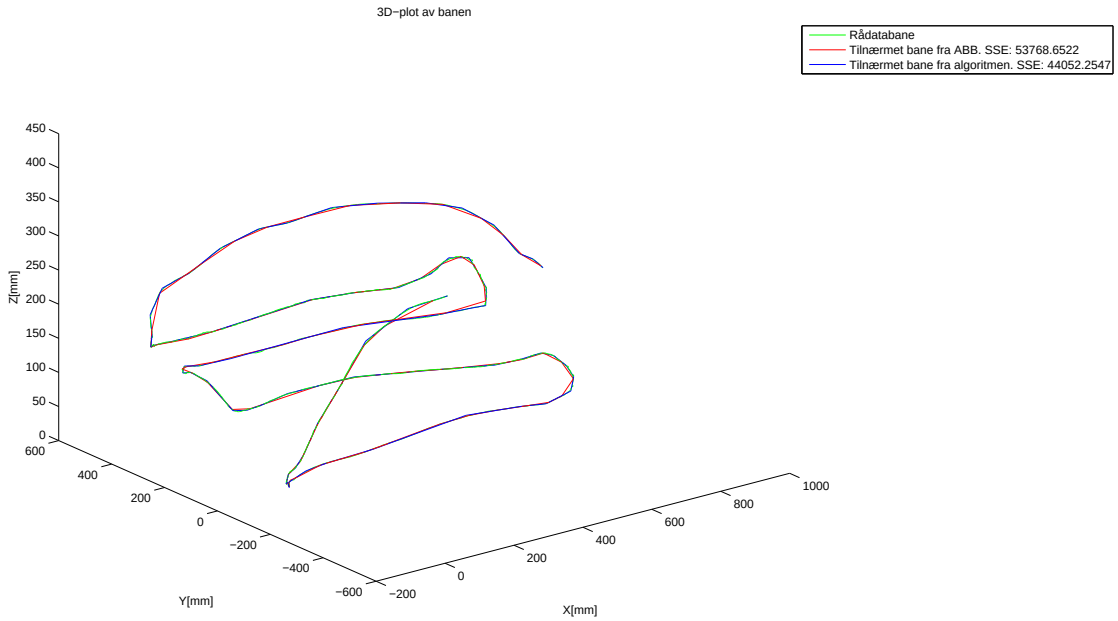
Figur 50: 2D-plot av orienteringen i bane

Måling	Bane fra algoritme	Referansebane
Sum av kvadrert feil for alle dimensjonene i dataen	43551	53769
Sum av kvadrert feil posisjon	1485,1	18290
Sum av kvadrert feil hastighet	44,8533	52,6629
Sum av kvadrert feil orientering	102,2546	746,2122
Gjennomsnittlig avvik posisjon	0,8855	2,1286
Gjennomsnittlig avvik hastighet	0,1362	0,1540
Gjennomsnittlig avvik orientering	0,2288	0,4656
Maksimalt avvik posisjon	5,0414	48,1178
Maksimalt avvik hastighet	0,5945	0,5799
Maksimalt avvik orientering	1,3508	4,7551
Lengdeforhold	98,1339	96,1622
Orienteringsforhold	95,5605	89,7764
Antall punkter i banen	72	72

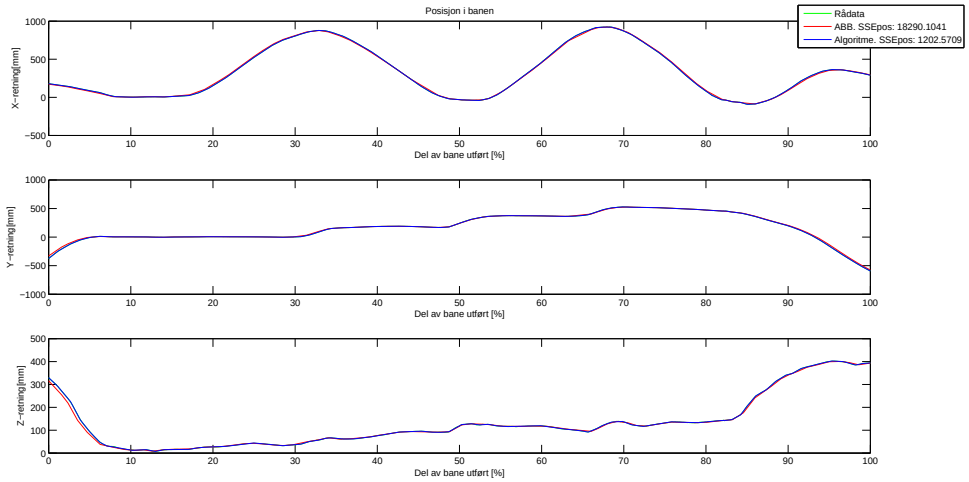
Tabell 2: Mål på godhet for referansebane og bane utviklet med algoritme, ved å spesifisere at banen skal optimaliseres for orientering

5.2.3 Resultat fra algoritmen ved å optimalisere for posisjon i banen. Banen inneholder like mange punkter som referansebanen

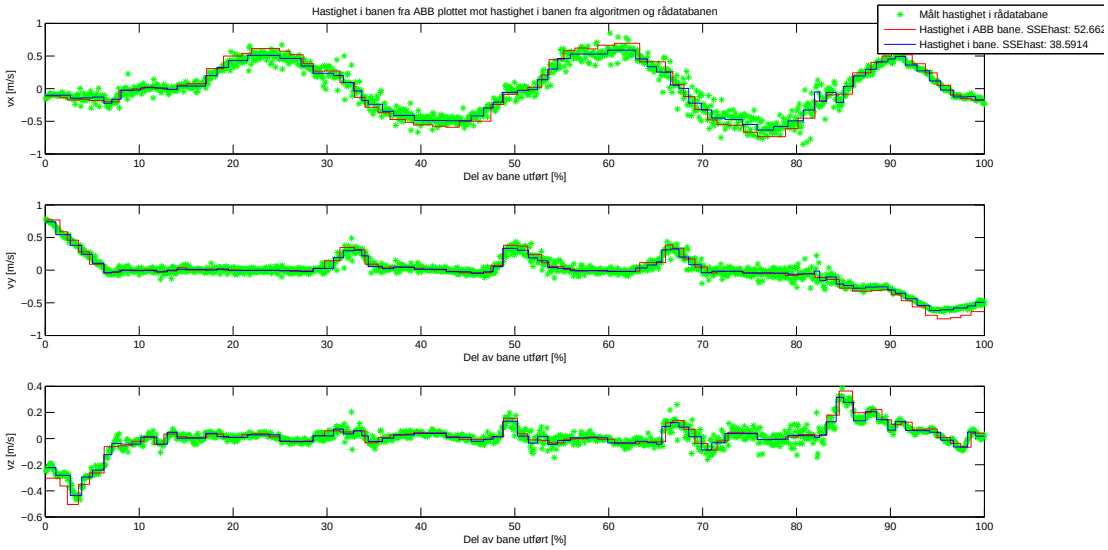
Her illustreres resultatet fra algoritmen ved å spesifisere vekt-parameteren som vekt = [5 1 1]. Dette gir en optimalisering av posisjon i banen ved å tillate mer feil for de resterende egenskapene. Sum av kvadrert feil for posisjon blir multiplisert med faktoren 5, som fører til at algoritmen vil minimere avviket for posisjon og tillate mer avvik for de resterende egenskapene i banen. En kan sammenligne dette resultatet med resultatet i tabell 1, hvor det er valgt å ikke optimalisere banen for en gitt egenskap. Som resultatet i tabellen 3 viser, er det tydelig at avviket for posisjon i banen er redusert, men da på bekostning av avviket i orientering. Det kan også ses at avviket for hastighet er forbedret, noe som er naturlig da hastigheten er avhengig av posisjonsendring. Mindre avvik i posisjon fører da til mindre avvik i hastighet. Vekt-parameteren er satt til [5 1 1] kun for å illustrere hva resultatet fra algoritmen blir dersom den optimaliseres for posisjon. Det kan godt tenkes at andre valg av vekter som ikke favoriserer posisjon i så stor grad er å foretrekke i en reell situasjon.



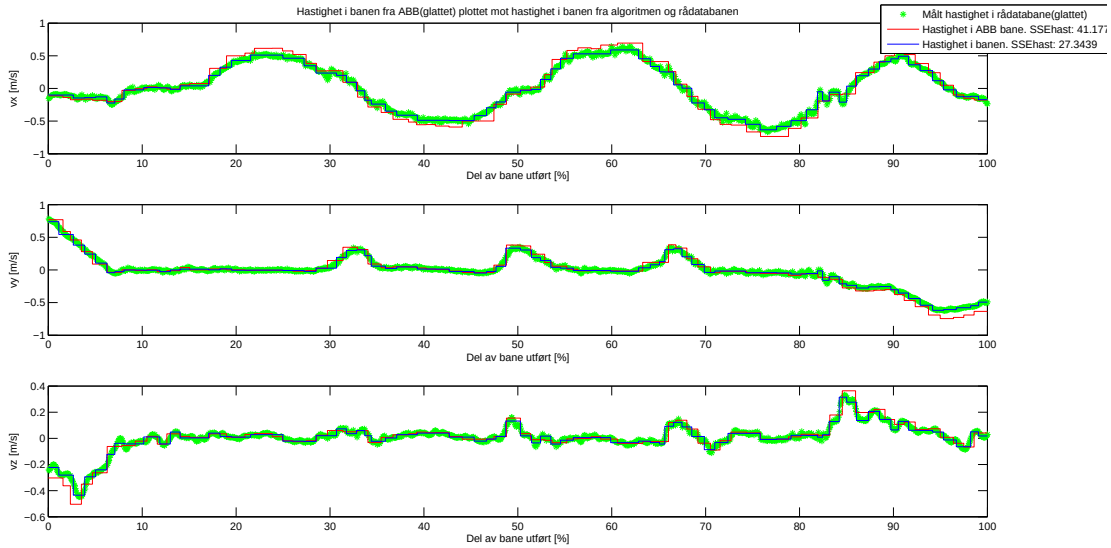
Figur 51: 3D-plot av bane



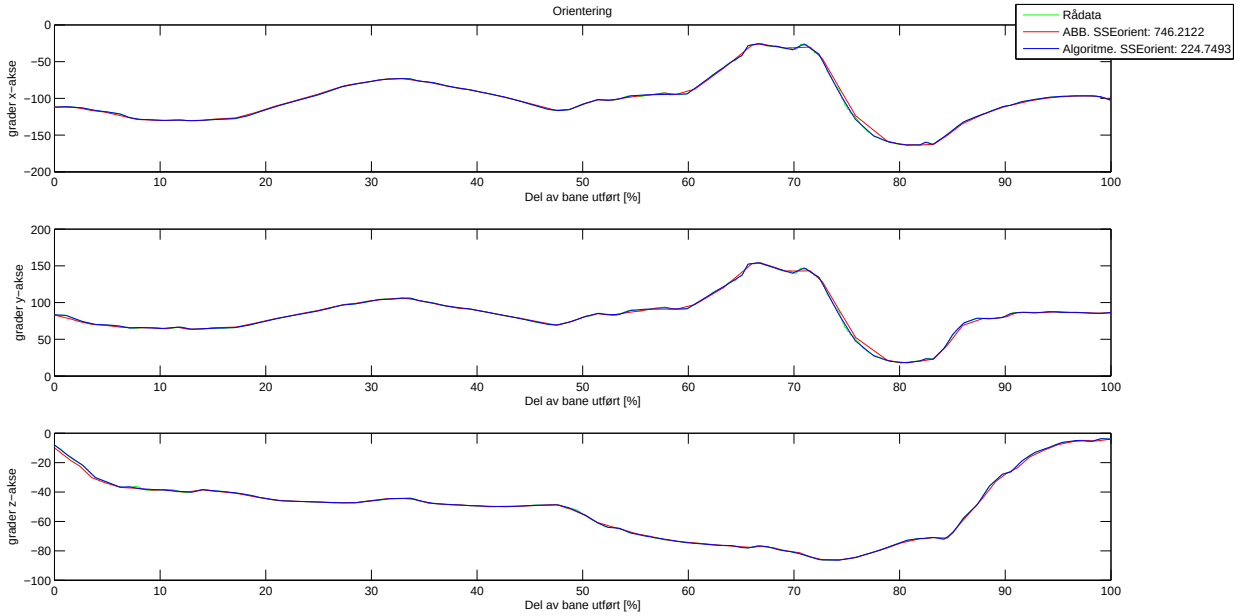
Figur 52: 2D-plot av posisjon i bane



Figur 53: 2D-plot av hastighet i bane



Figur 54: 2D-plot av hastighet i bane mot glattet hastighet i rådatabane



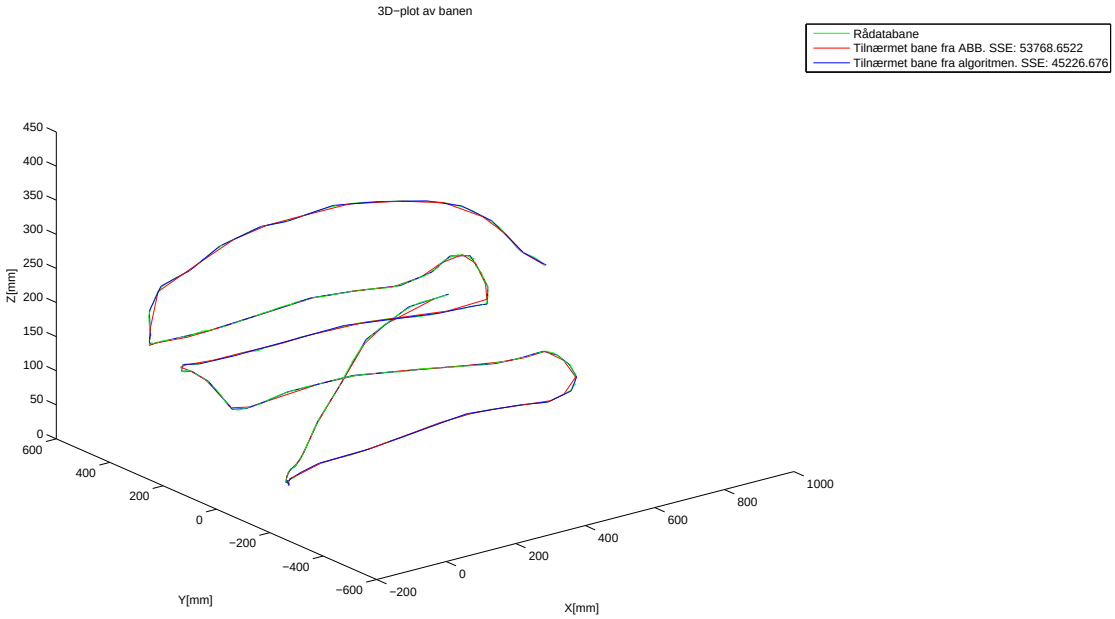
Figur 55: 2D-plot av orienteringen i bane

Måling	Bane fra algoritme	Referansebane
Sum av kvadrert feil for alle dimensjonene i dataen	44052	53769
Sum av kvadrert feil posisjon	1202,6	18290
Sum av kvadrert feil hastighet	38,5914	52,6629
Sum av kvadrert feil orientering	224,7493	746,2122
Gjennomsnittlig avvik posisjon	0,8206	2,1286
Gjennomsnittlig avvik hastighet	0,1267	0,1540
Gjennomsnittlig avvik orientering	0,2850	0,4656
Maksimalt avvik posisjon	4,3792	48,1178
Maksimalt avvik hastighet	0,5654	0,5799
Maksimalt avvik orientering	2,6877	4,7551
Lengdeforhold	98,1088	96,1622
Orienteringsforhold	93,6624	89,7764
Antall punkter i banen	72	72

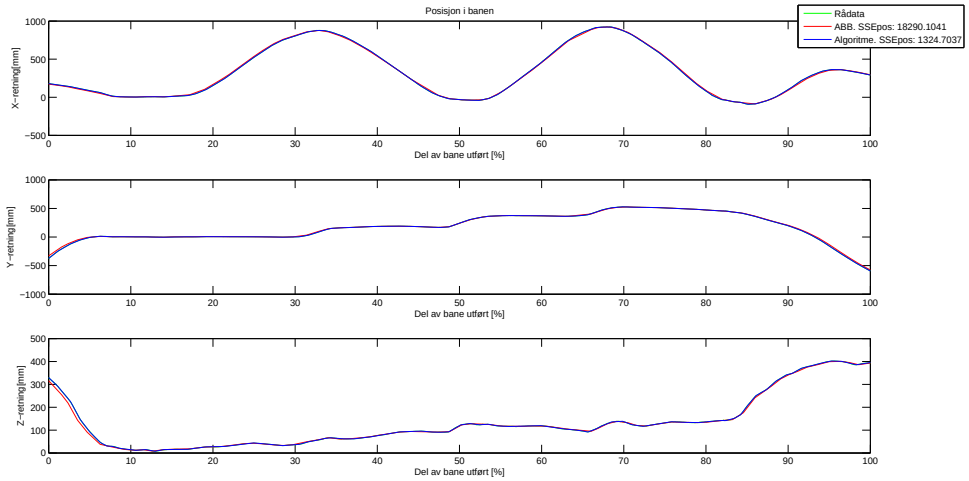
Tabell 3: Mål på godhet for referansebane og bane utviklet med algoritme, ved å spesifisere at banen skal optimaliseres for posisjon

5.2.4 Resultat fra algoritmen ved å optimalisere for hastighet i banen. Banen inneholder like mange punkter som referansebanen

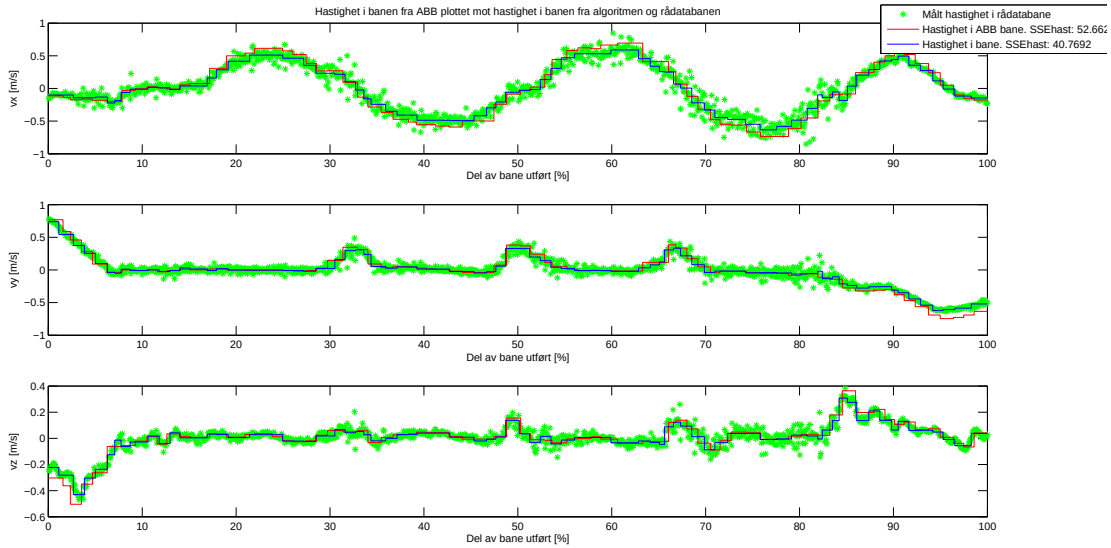
Her illustreres resultatet fra algoritmen ved å spesifisere vekt-parameteren som vekt = [1 1 5], som gir en optimalisering av hastighet i banen ved å tillate mer feil for de resterende egenskapene. Hastighet er som kjent avhenging av posisjonsendring og tidsbruk, så ved å optimalisere banen for både posisjon og tid, vil resultatet bli en bane som har mindre avvik i hastighet. Sum av kvadrert feil for posisjon og tid blir multiplisert med faktoren 5, som fører til at algoritmen vil minimere avviket for posisjon og tid og tillate mer avvik for de orienteringen i banen. En kan sammenligne dette resultatet med resultatet i tabell 1 . Som resultatet i tabellen 4 viser, er det tydelig at avviket for hastigheten i banen er redusert, men da på bekostning av avviket i orientering. Det ses også en liten forbedring i avviket for posisjon. Dette har en sammenheng med at når banen optimaliseres for hastighet, vil avviket i posisjon minimeres, sammen med avviket for tid.
Dersom resultatet sammenlignes med tabell 3, vil det ses at avviket for hastighet er mindre for banen som er optimalisert for posisjon, enn for banen er optimalisert for hastighet. Dette er igjen på grunn av sammenhengen mellom optimalisering av posisjon og optimalisering av hastighet. Selv om dette er tilfellet her, viser det seg i del 5.5.4 at banen fra algoritmen som optimaliserer hastighet kan gi et bedre resultat for avviket i hastighet. Dette er dersom banen skal inneholde så få punkter som mulig. Sum av kvadrert avvik i hastighet for en bane med 55 punkter, når banen optimaliseres for hastighet er 51,1111. Tilsvarende for en bane med 55 punkter som optimaliseres for posisjon blir 52,2923. Vekt-parameteren er satt til [1 1 5] kun for å illustrere hva resultatet fra algoritmen blir dersom den optimaliseres for hastighet. Det kan godt tenkes at andre valg av vekter som ikke favoriserer hastighet i så stor grad er å foretrekke i en reell situasjon.



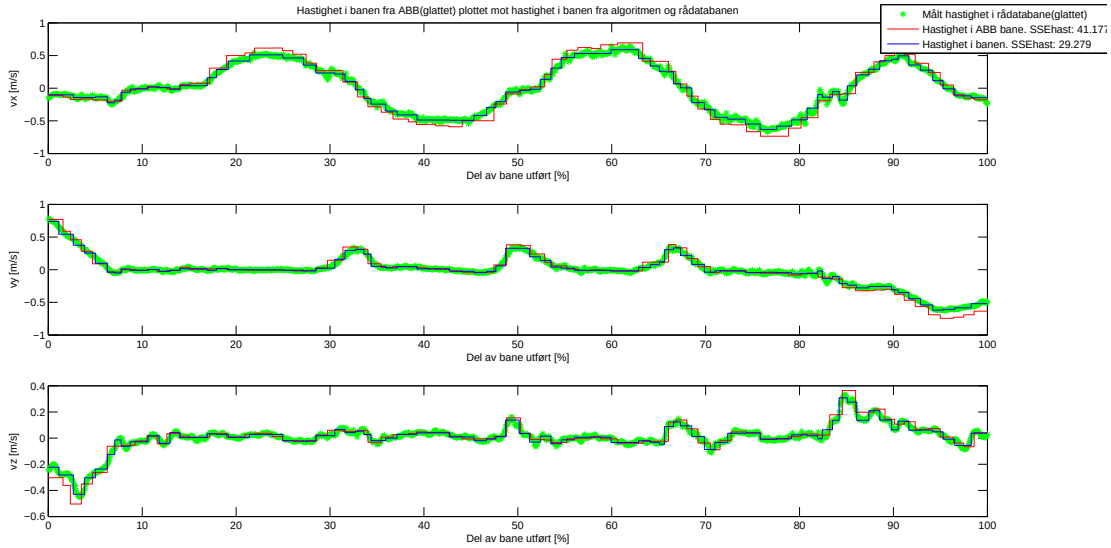
Figur 56: 3D-plot av bane



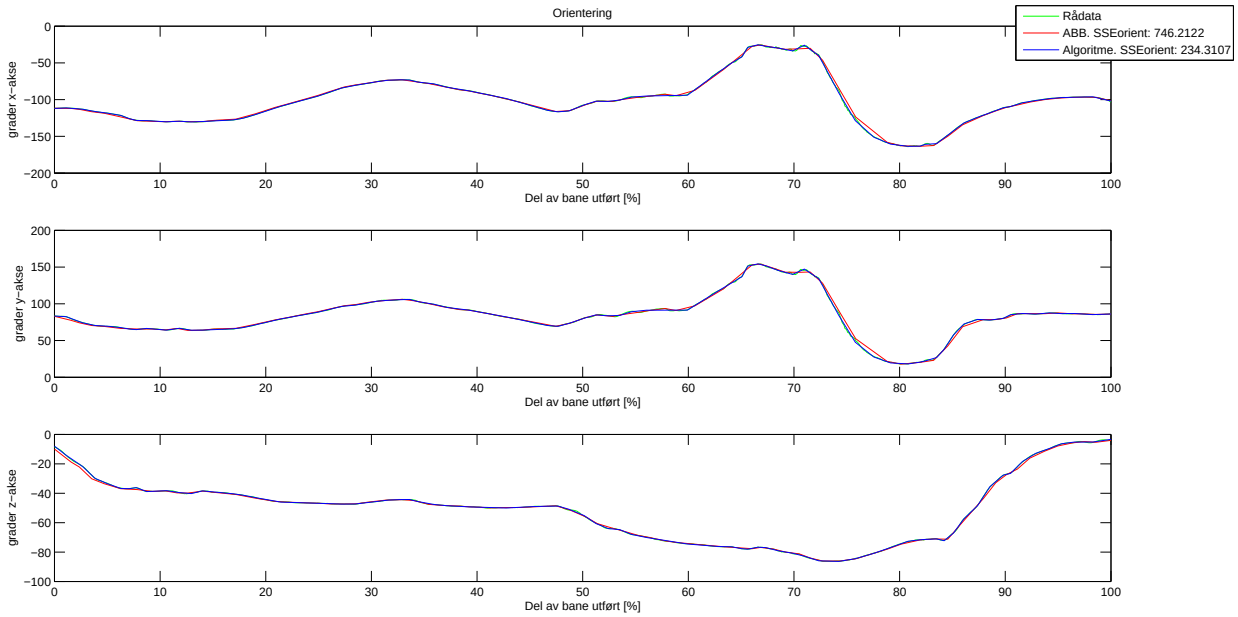
Figur 57: 2D-plot av posisjon i bane



Figur 58: 2D-plot av hastighet i bane



Figur 59: 2D-plot av hastighet i bane mot glattet hastighet i rådatabane



Figur 60: 2D-plot av orienteringen i bane

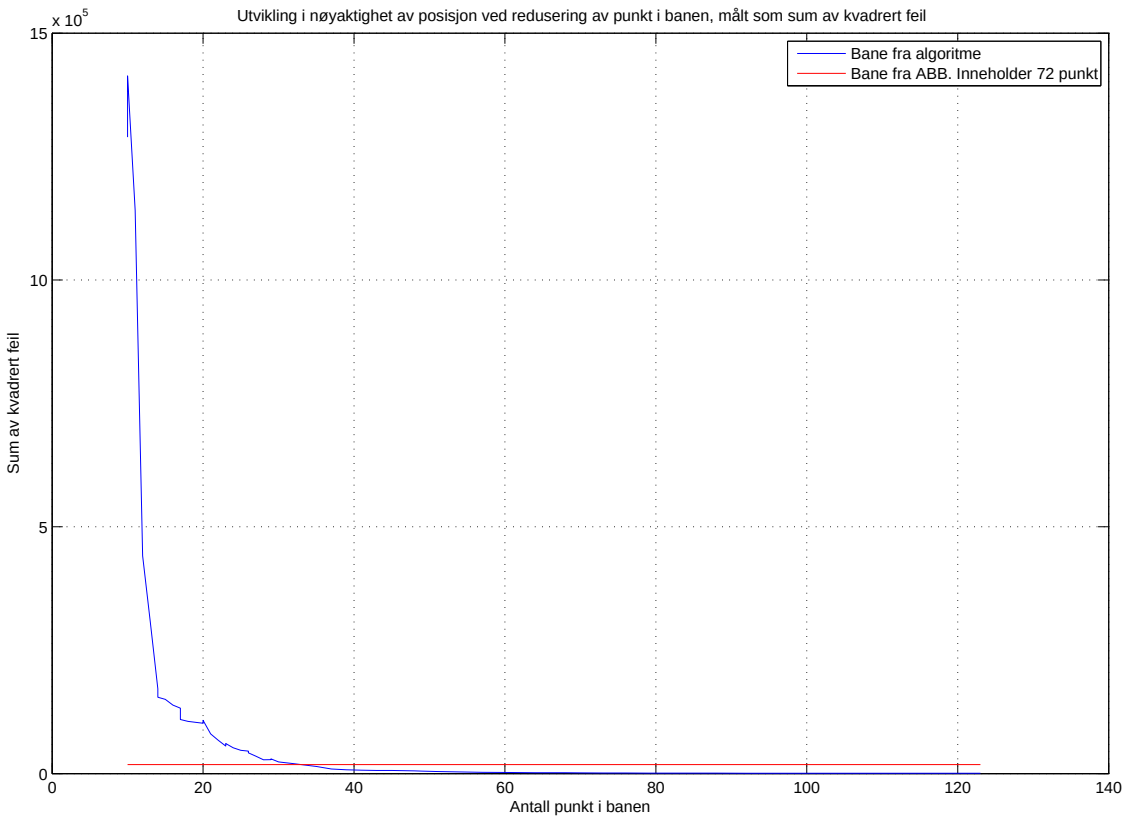
Måling	Bane fra algoritme	Referansebane
Sum av kvadrert feil for alle dimensjonene i dataen	45226	53769
Sum av kvadrert feil posisjon	1324,7	18290
Sum av kvadrert feil hastighet	40,7692	52,6629
Sum av kvadrert feil orientering	234,3107	746,2122
Gjennomsnittlig avvik posisjon	0,8484	2,1286
Gjennomsnittlig avvik hastighet	0,1278	0,1540
Gjennomsnittlig avvik orientering	0,2954	0,4656
Maksimalt avvik posisjon	4,3187	48,1178
Maksimalt avvik hastighet	0,5945	0,5799
Maksimalt avvik orientering	2,6520	4,7551
Lengdeforhold	98,1076	96,1622
Orienteringsforhold	93,2756	89,7764
Antall punkter i banen	72	72

Tabell 4: Mål på godhet for referansebane og bane utviklet med algoritme, ved å spesifisere at banen skal optimaliseres for hastighet

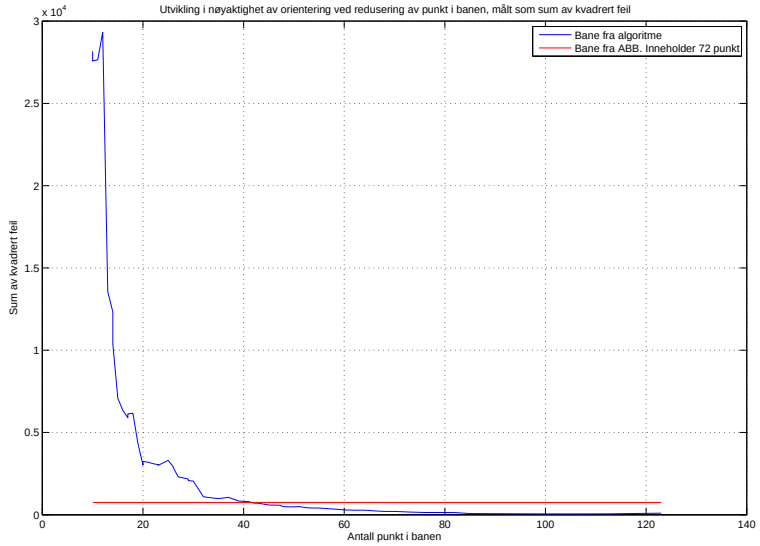
5.3 Utvikling for avviket ved reduksjon av antall punkter i banen

I denne delen undersøkes utviklingen av avviket ved reduksjon av antall punkter i banen. For hver gang algoritmen reduserer antall punkter i banen, blir avviket sjekket. Utviklingen av avviket er undersøkt for fire baner.: En optimalisert for posisjon, en optimalisert for hastighet og en optimalisert for orientering, i tillegg til en bane der ingen optimalisering er spesifisert. Avviket blir sammenlignet med avviket for referansebanen. Fra figurene presentert her er det mulig å se hvor mange punkter som kan benyttes i en bane, før avviket overstiger avviket i referansebanen. Dette gjelder for hvert valg av egenskaper banen skal optimaliseres for. Alle vekter er satt slik at avviket for valgt egenskap blir multiplisert med 5 i algoritmen. For hastighet betyr dette en multiplisering av avviket i posisjon og tid med 5. Det målte avviket som presenteres i figurene under er uten vekting.

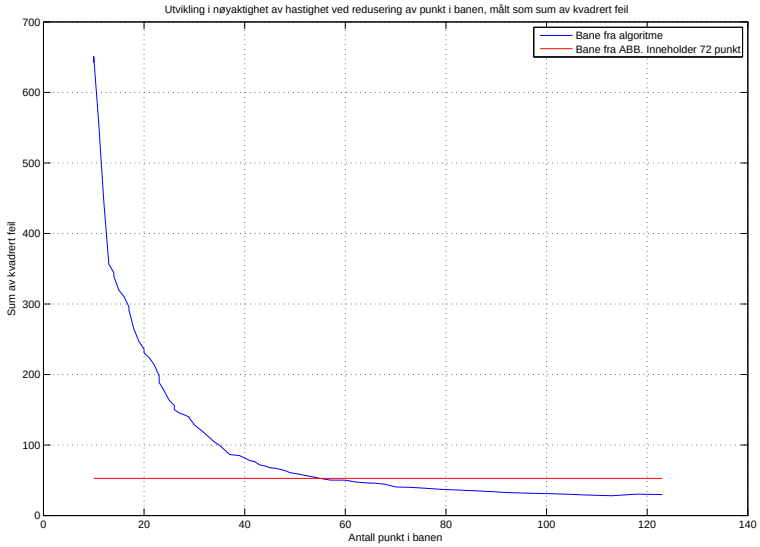
5.3.1 Utvikling av avviket i banen ved reduksjon av antall punkter, dersom banen ikke optimaliseres for en gitt egenskap



Figur 61: Utviklingen i avviket for posisjon ved reduksjon av antall punkter i banen

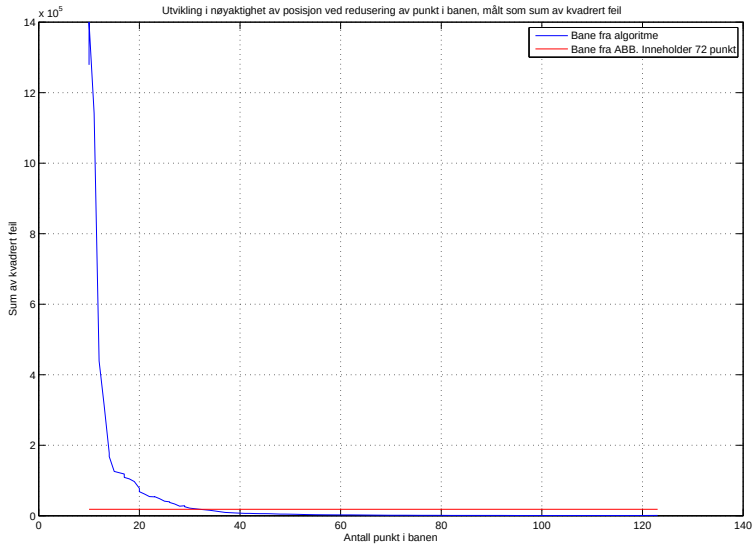


Figur 62: Utviklingen i avviket for orientering ved reduksjon av antall punkter i banen

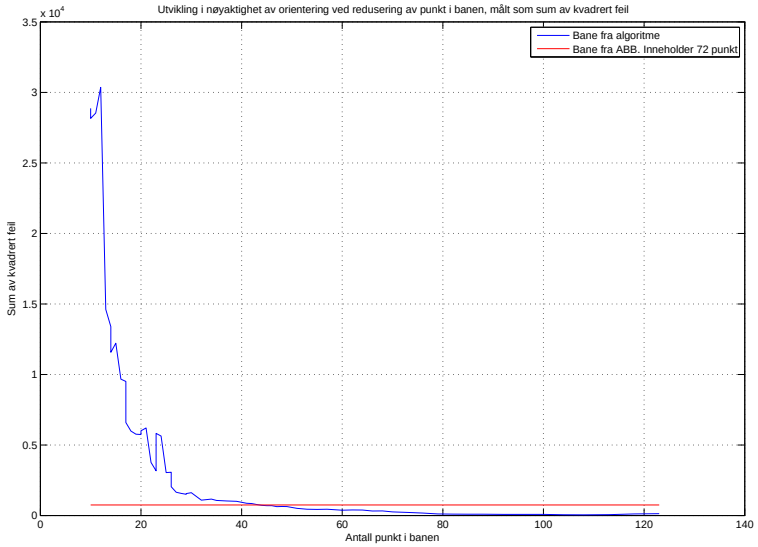


Figur 63: Utviklingen i avviket for hastighet ved reduksjon av antall punkter i banen

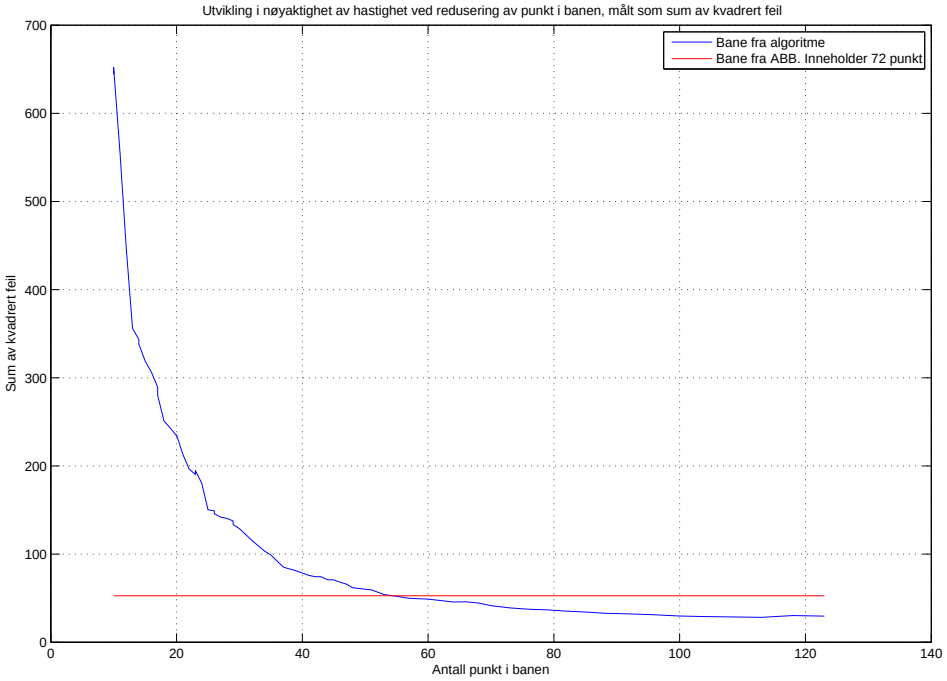
5.4 Utvikling av avviket i banen ved reduksjon av antall punkter, dersom banen optimaliseres for posisjon



Figur 64: Utviklingen i avviket for posisjon ved reduksjon av antall punkter i banen

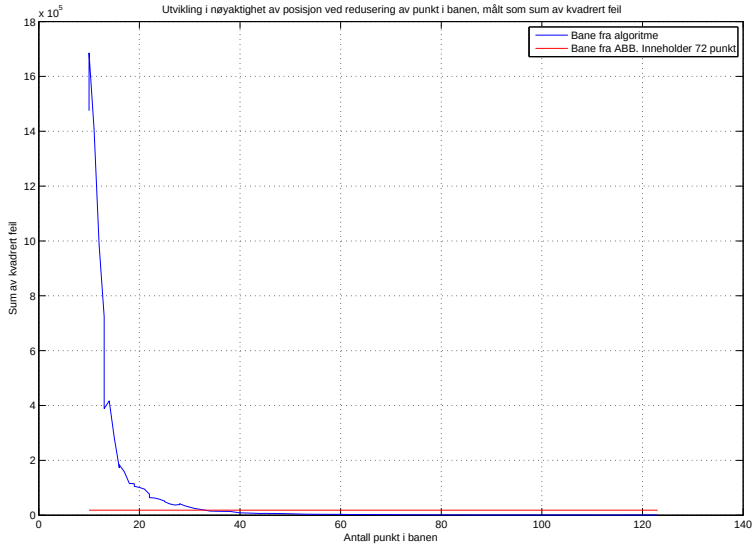


Figur 65: Utviklingen i avviket for orientering ved reduksjon av antall punkter i banen

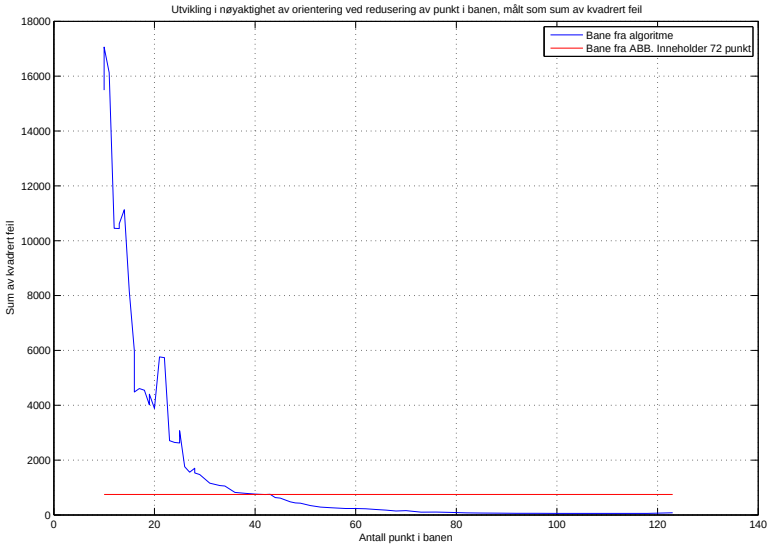


Figur 66: Utviklingen i avviket for hastighet ved reduksjon av antall punkter i banen

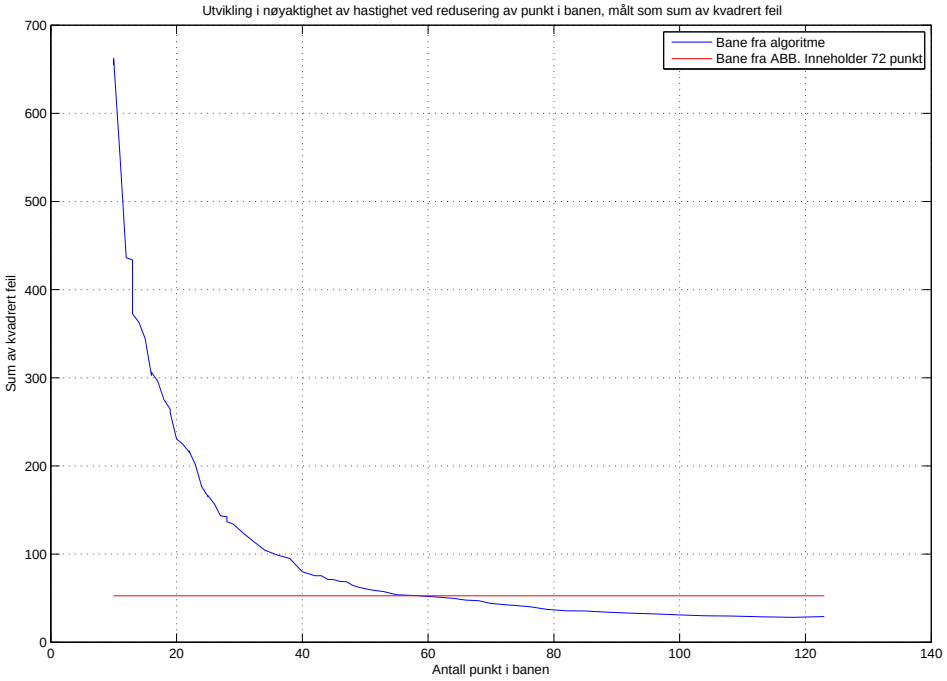
5.4.1 Utvikling av avviket i banen ved reduksjon av antall punkter, dersom banen optimaliseres for orientering



Figur 67: Utviklingen i avviket for posisjon ved reduksjon av antall punkter i banen

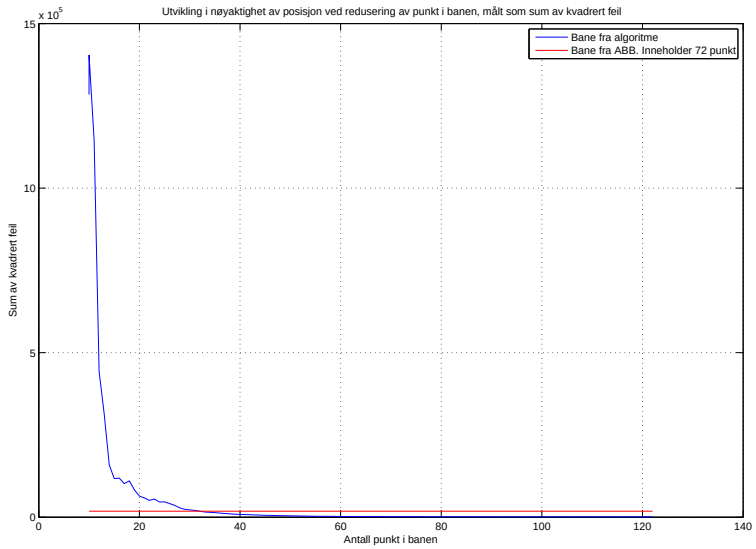


Figur 68: Utviklingen i avviket for orientering ved reduksjon av antall punkter i banen

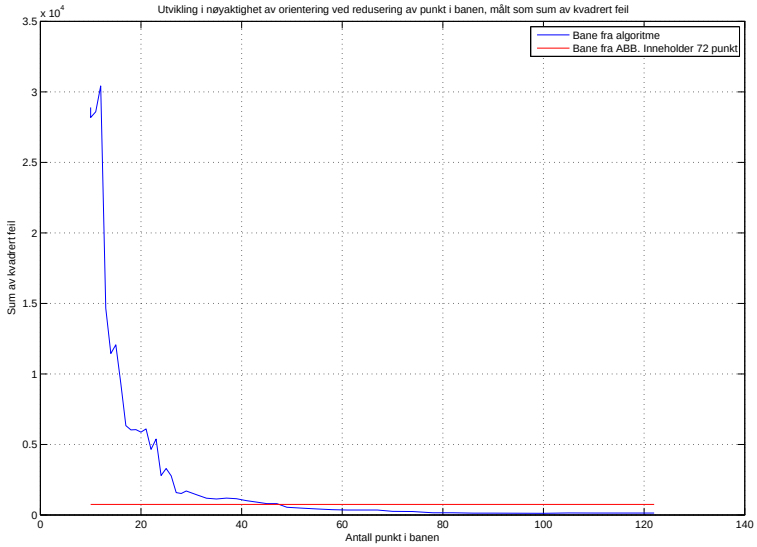


Figur 69: Utviklingen i avviket for hastighet ved reduksjon av antall punkter i banen

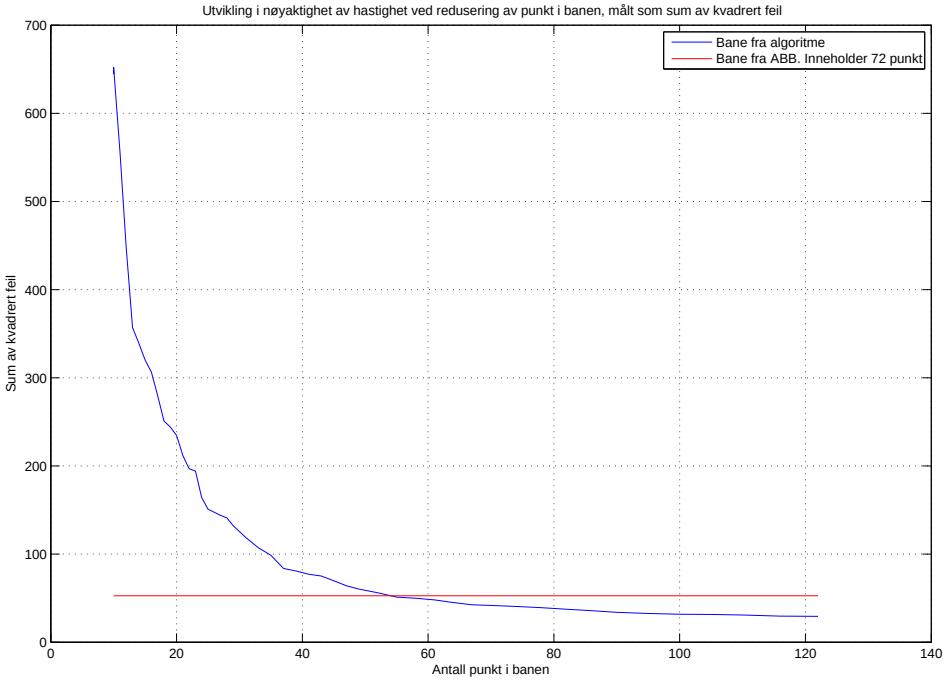
5.4.2 Utvikling av avviket i banen ved reduksjon av antall punkter, dersom banen optimaliseres for hastighet



Figur 70: Utviklingen i avviket for posisjon ved reduksjon av antall punkter i banen



Figur 71: Utviklingen i avviket for orientering ved reduksjon av antall punkter i banen



Figur 72: Utviklingen i avviket for hastighet ved reduksjon av antall punkter i banen

5.5 Resultat bane2 med så få punkter som mulig før feilen i banen blir like stor som i referansebanen

Det ønskes her å poengtere hvor få punkter som kan benyttes i banen fra algoritmen for å få tilsvarende avvik som for referansebanen. Det blir produsert et resultat for hvert valg av vekt-parameter i algoritmen. I resultatet hvor banen blir optimalisert for posisjon, vil det undersøkes hvor få punkter som kan brukes før avviket i posisjon blir større enn i referansebanen. Tilsvarende for orientering og hastighet. Dette vil illustrere mulighetene for bruk av algoritmen, dersom èn av egenskapene er vesentlig viktigere å optimalisere enn de andre. For resultatet fra algoritmen der banen ikke optimaliseres for en gitt egenskap, vil avviket for den egenskapen som først nærmer seg størrelsen på avviket i referansebanen være en begrensende faktor på antall punkter i banen. Antall punkter som skal benyttes i banen finnes ved å se på figurene i del 5.3.

5.5.1 Resultat fra algoritmen uten å spesifisere hvilken egenskap banen skal optimaliseres for. Banen inneholder så få punkter som mulig før avviket for noen av egenskapene blir vesentlig større enn tilsvarende avvik for referansebanen

Her forsøkes det å representere en bane med så få punkter som mulig, før et av avvikene i banen blir vesentlig større enn tilsvarende avvik i referansebanen. Banen blir laget med algoritmen presentert i denne rapporten, uten å spesifisere hvilken egenskap som banen ønskes optimalisert for. Dette betyr at vekt-parameteren er satt lik [1 1 1]. Fra figurene i del 5.3 er det klart at avviket for hastighet i banen når samme størrelse som avviket for hastighet i referansebanen, før de resterende avvikene gjør det samme. Fra figurene ser det ut til at antall punkt som kan benyttes i banen er 58. Dette gir en reduksjon i antall punkter fra den originale banen på 95,47%. Se tabell 5 for en sammenligning mellom godheten i banen fra algoritmen og referansebanen.

Måling	Bane fra algoritme	Referansebane
Sum av kvadrert feil posisjon	2384,4	18290
Sum av kvadrert feil hastighet	49,4814	52,6629
Sum av kvadrert feil orientering	319,3971	746,2122
Gjennomsnittlig avvik posisjon	1,1418	2,1286
Gjennomsnittlig avvik hastighet	0,1480	0,1540
Gjennomsnittlig avvik orientering	0,3585	0,4656
Maksimalt avvik posisjon	4,9987	48,1178
Maksimalt avvik hastighet	0,5945	0,5799
Maksimalt avvik orientering	2,4990	4,7551
Lengdeforhold	97,9554	96,1622
Orienteringsforhold	92,2898	89,7764
Antall punkter i banen	58	72

Tabell 5: Mål på godhet for referansebane og bane utviklet med algoritme, uten å spesifisere at banen skal optimaliseres for en egenskap

5.5.2 Resultat fra algoritmen ved å spesifisere at banen skal optimaliseres for posisjon. Banen inneholder så få punkter som mulig før avviket for posisjon blir vesentlig større enn tilsvarende avvik for referansebanen

Her forsøkes det å representere en bane med så få punkter som mulig, før avviket for posisjon i banen blir vesentlig større enn avviket for posisjon i referansebanen. Banen blir laget med algoritmen presentert i denne rapporten, ved å spesifisere at banen ønskes optimalisert for posisjon. Vekt-parameteren i algoritmen er spesifisert som vekt = [5 1 1]. Fra figurene i del 5.3 ser det ut til at antall punkt som kan benyttes i banen er 32. Dette gir en reduksjon i antall punkter fra den originale banen på 97,5%. Se tabell 6 for en sammenligning mellom godheten i banen fra algoritmen og referansebanen.

Måling	Bane fra algoritme	Referansebane
Sum av kvadrert feil posisjon	17640	18290
Sum av kvadrert feil hastighet	114,9066	52,6629
Sum av kvadrert feil orientering	1160,3	746,2122
Gjennomsnittlig avvik posisjon	3,0916	2,1286
Gjennomsnittlig avvik hastighet	0,2534	0,1540
Gjennomsnittlig avvik orientering	0,7137	0,4656
Maksimalt avvik posisjon	11,3080	48,1178
Maksimalt avvik hastighet	0,7378	0,5799
Maksimalt avvik orientering	4,2536	4,7551
Lengdeforhold	98,3920	96,1622
Orienteringsforhold	88,4779	89,7764
Antall punkter i banen	32	72

Tabell 6: Mål på godhet for referansebane og bane utviklet med algoritme, ved å spesifisere at banen skal optimaliseres for posisjon

5.5.3 Resultat fra algoritmen ved å spesifisere at banen skal optimaliseres for orientering. Banen inneholder så få punkter som mulig før avviket for orientering blir vesentlig større enn tilsvarende avvik for referansebanen

Her forsøkes det å representere en bane med så få punkter som mulig, før avviket for orientering i banen blir vesentlig større enn avviket for orientering i referansebanen. Banen blir laget med algoritmen presentert i denne rapporten, ved å spesifisere at banen ønskes optimalisert for orientering. Vekt-parameteren i algoritmen er spesifisert som vekt = [1 5 1]. Fra figurene i del 5.3 ser det ut til at antall punkt som kan benyttes i banen er 40, før avviket for orientering i banen blir vesentlig høyere enn i referansebanen. Dette gir en reduksjon på antall punkter fra den originale banen på 96,88%. Se tabell 8 for en sammenligning mellom godheten i banen fra algoritmen og referansebanen.

Måling	Bane fra algoritme	Referansebane
Sum av kvadrert feil posisjon	8382,1	18290
Sum av kvadrert feil hastighet	78,6910	52,6629
Sum av kvadrert feil orientering	766,5023	746,2122
Gjennomsnittlig avvik posisjon	2,1197	2,1286
Gjennomsnittlig avvik hastighet	0,1974	0,1540
Gjennomsnittlig avvik orientering	0,5857	0,4656
Maksimalt avvik posisjon	10,4312	48,1178
Maksimalt avvik hastighet	0,6012	0,5799
Maksimalt avvik orientering	3,3895	4,7551
Lengdeforhold	98,2507	96,1622
Orienteringsforhold	91,2365	89,7764
Antall punkter i banen	40	72

Tabell 7: Mål på godhet for referansebane og bane utviklet med algoritme, ved å spesifisere at banen skal optimaliseres for orientering

5.5.4 Resultat fra algoritmen ved å spesifisere at banen skal optimaliseres for hastighet. Banen inneholder så få punkter som mulig før avviket for hastighet blir vesentlig større enn tilsvarende avvik for referansebanen

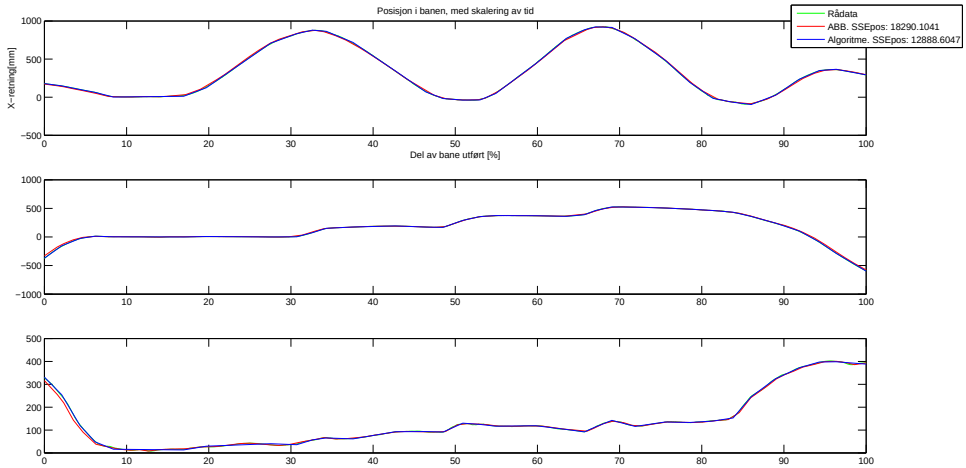
Her forsøkes det å representere en bane med så få punkter som mulig, før avviket for hastighet i banen blir vesentlig større enn avviket for hatighet i referansebanen. Banen blir laget med algoritmen presentert i denne rapporten, ved å spesifisere at banen ønskes optimalisert for hastighet. Vekt-parameteren i algoritmen er spesifisert som vekt = [1 1 5]. Fra figurene i del 5.3 ser det ut til at antall punkt som kan benyttes i banen er 55, før avviket for orientering i banen blir vesentlig høyere enn i referansebanen. Dette gir en reduksjon på antall punkter fra den originale banen på 95,71%. Se tabell 8 for en sammenligning mellom godheten i banen fra algoritmen og referansebanen.

Måling	Bane fra algoritme	Referansebane
Sum av kvadrert feil posisjon	2896,3	18290
Sum av kvadrert feil hastighet	51,1111	52,6629
Sum av kvadrert feil orientering	426,8496	746,2122
Gjennomsnittlig avvik posisjon	1,2568	2,1286
Gjennomsnittlig avvik hastighet	0,1521	0,1540
Gjennomsnittlig avvik orientering	0,4136	0,4656
Maksimalt avvik posisjon	6,0895	48,1178
Maksimalt avvik hastighet	0,5945	0,5799
Maksimalt avvik orientering	2,6980	4,7551
Lengdeforhold	97,9833	96,1622
Orienteringsforhold	91,5289	89,7764
Antall punkter i banen	55	72

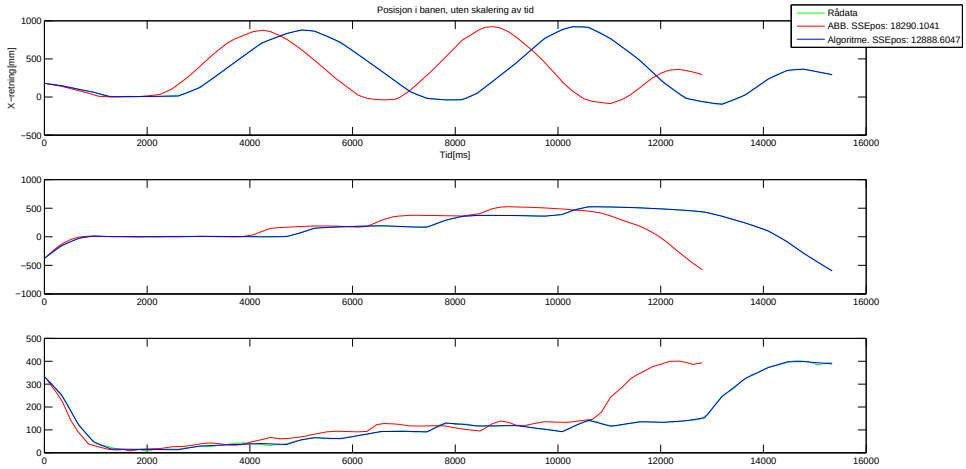
Tabell 8: Mål på godhet for referansebane og bane utviklet med algoritme, ved å spesifisere at banen skal optimaliseres for hastighet

5.6 Sammenligning av referansebane, rådatabane og bane fra algoritmen, uten skalering av tid

Her vises to figurer som illustrerer feilen som kommer av at referansebanene har feil hastighet og bruker mindre tid enn rådatabanen. Det er tydelig at utviklingen i posisjon i referansebanen er forskjøvet i forhold til posisjon i banen og rådatabanen. Dette kommer av at referansebanen har høyere hastighet enn rådatabanen. Første figur 73 viser posisjonen i banen ved skalert tid, og viser at referansebanen følger posisjonen i rådatabanen ganske godt. Den andre figuren 74 viser posisjon i banen uten skalering av tiden og illustrerer hvordan utviklingen i posisjon i banen er forskjøvet i tid. Bevegelsene i referansebanen er ferdig utført på kun 83,17% av tiden som brukes i rådatabanen. Til sammenligning bruker banen fra algoritmen like lang tid på bevegelsene.



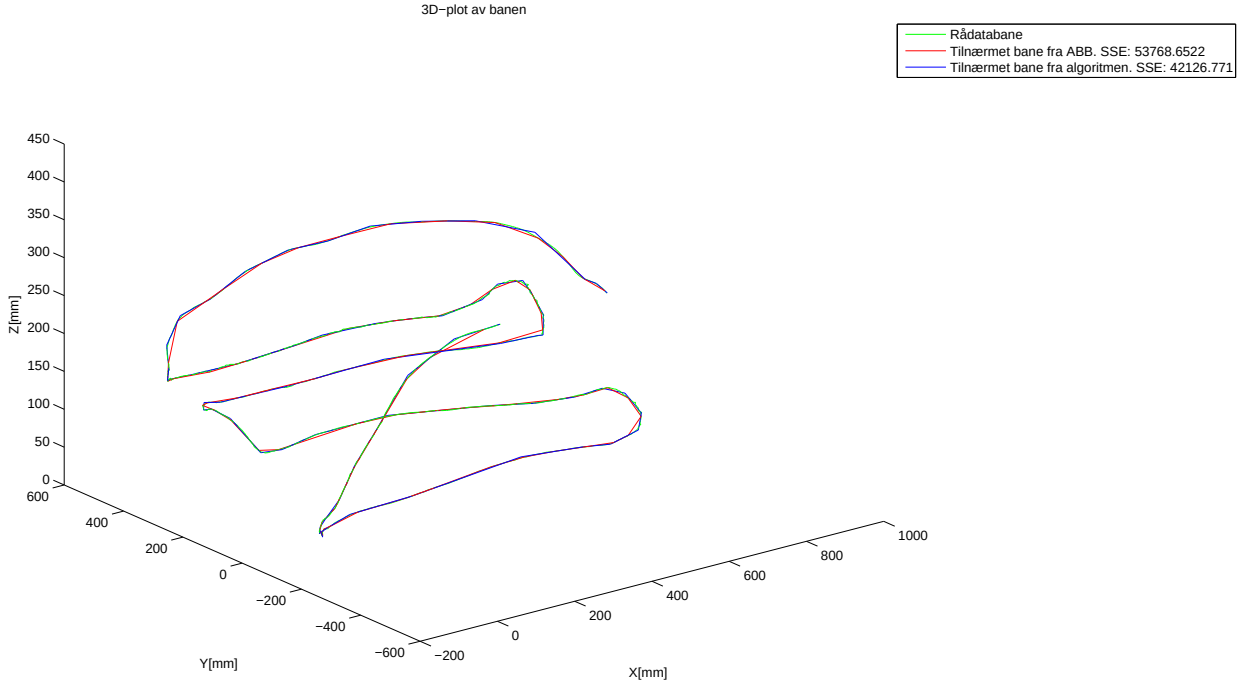
Figur 73: 2D-plot av posisjon i banen mot posisjon i referansebanen, med skalering av tiden. Punkter referansebane = 72, punkter bane = 34.



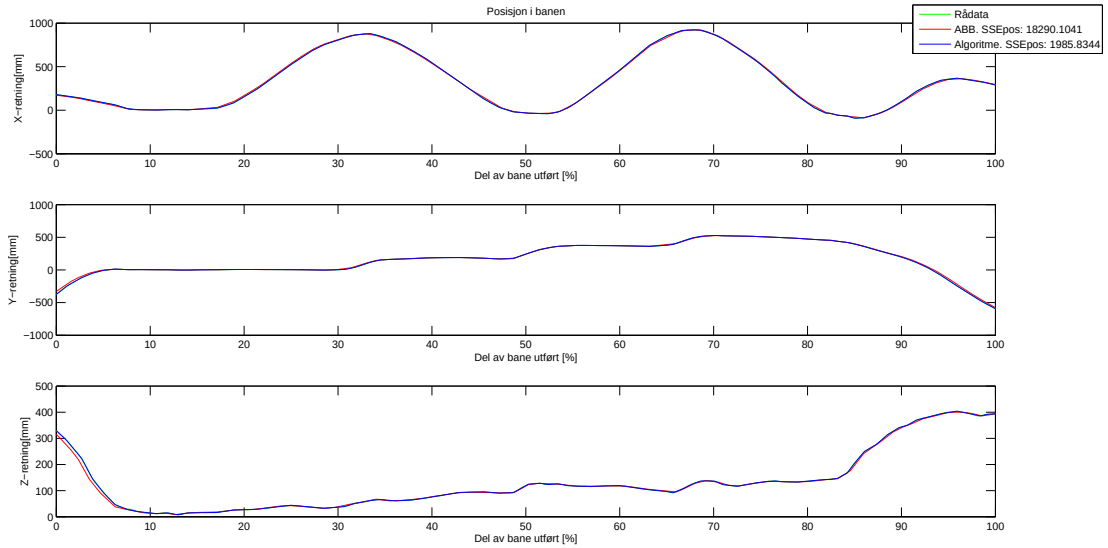
Figur 74: 2D-plot av posisjon i banen mot posisjon i referansebanen, uten skalering av tiden. Punkter referansebane = 72, punkter bane = 34.

5.7 Utvikling av bane med algoritmen hvor margin-parameteren er spesifisert

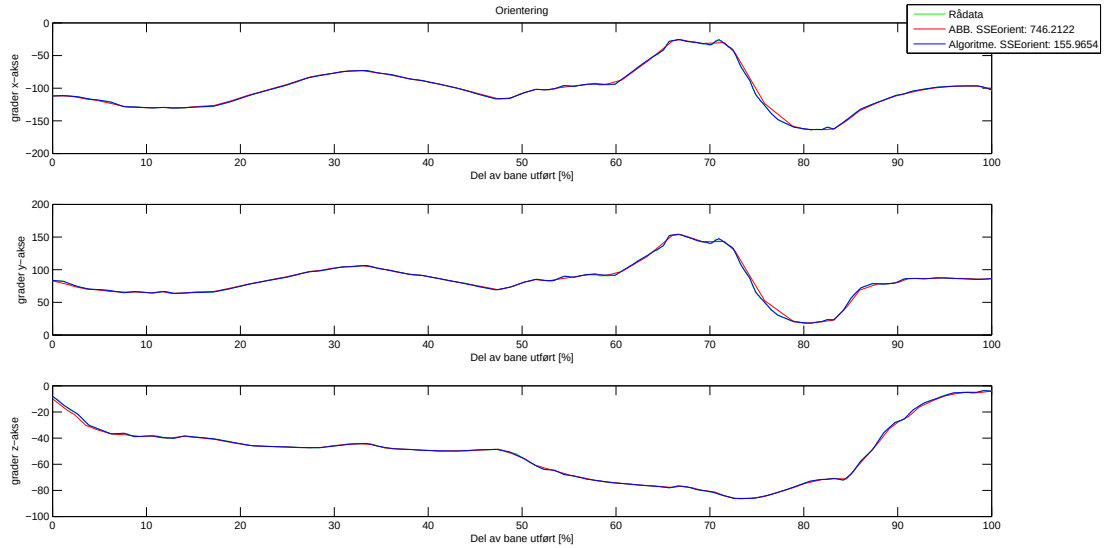
Under vises en figur for utviklingen i banen ved å spesifisere margin-parameteren i algoritmen til 1,2. Dette betyr at det tillates en 20% økning i feilen i forhold til første utkast av banen, som jo inneholder mange punkter og har god nøyaktighet. Hviken egenskap som det måles økning i feil for, bestemmes av spesifisert vekt-parameter. For illustrasjonen under er vekt-parameteren valgt som : vekt = [1 5 1], som optimaliserer banen ved å minimalisere avviket i orientering. Avviket funnet for orienteringen blir multiplisert med faktoren 5, som betyr at algoritmen velger å tilpasse punktene i banen slik at avviket i orientering blir minst mulig, samtidig som det tillates et større avvik for posisjon og hastighet. Som det kommer frem i tabellen under 9, blir resultatet ved å tillate en 20% økning i kvadrert avvik for orienteringen en bane med 78 punkter. Ved første tilnærming til en bane i algoritmen var sum av kvadrert feil for orienteringen 79,7565. Endelig bane har en sum av kvadrert feil for orienteringen lik 94,1685. Dette tilsvarer en økning i sum av kvadrert feil på 18,07%, altså omtrent 20% økning, som forventet.



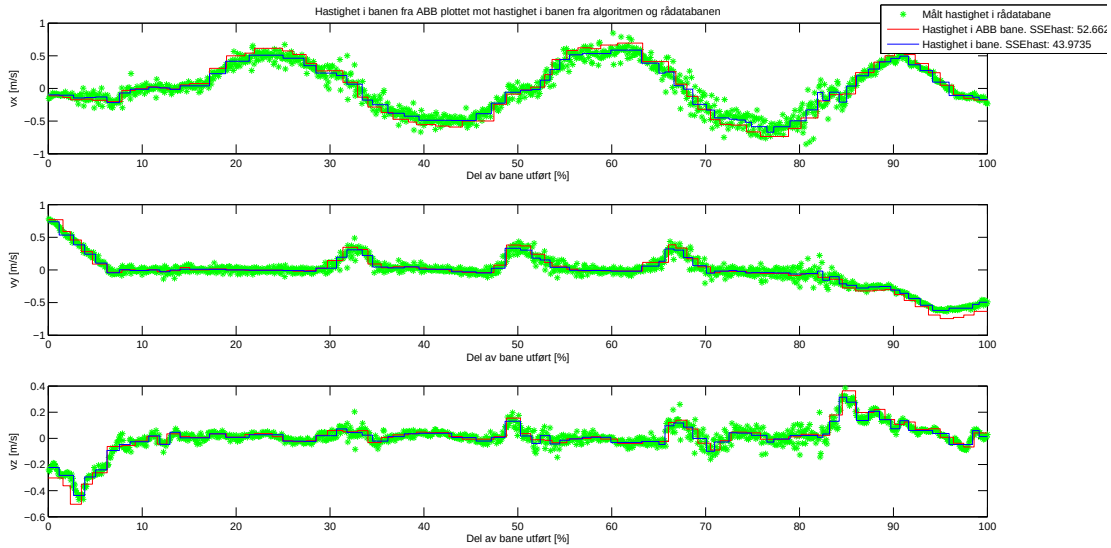
Figur 75: 3D-plot av bane



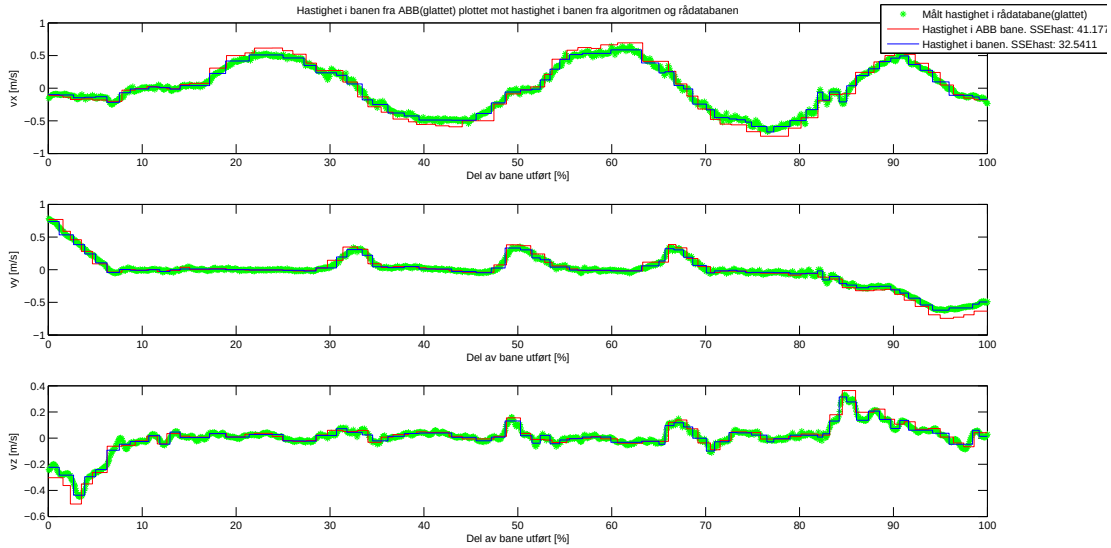
Figur 76: Posisjon i banen for x-,y- og z-retning



Figur 77: Orientering i banen



Figur 78: Hastighet i banen



Figur 79: Hastighet i banen mot glattet hastighet i rådatabanen

Måling	Bane fra algoritme	Referansebane
Sum av kvadrert feil for alle dimensjonene i dataen	42127	53769
Sum av kvadrert feil posisjon	1211,5	18290
Sum av kvadrert feil hastighet	13,64	484,1053
Sum av kvadrert feil orientering	94,1685	746,2122
Gjennomsnittlig avvik posisjon	0,8142	2,1286
Gjennomsnittlig avvik hastighet	0,0801	0,4700
Gjennomsnittlig avvik orientering	0,2158	0,4656
Maksimalt avvik posisjon	5,0835	48,1178
Maksimalt avvik hastighet	0,5028	1,4091
Maksimalt avvik orientering	1,3169	4,7551
Lengdeforhold	98,1278	96,1622
Orienteringsforhold	95,7903	89,7764
Antall punkter i banen	78	72

Tabell 9: Mål på godhet for referansebane og bane utviklet med algoritme, ved å spesifisere margin-parameteren til å tillate 20% økning i avvik for orienteringen i banen

5.8 Tidsbruk av algoritme

Tidsbruken til algoritmen varierer. Den er avhengig av antall punkter i rådatabanen som det skal tilpasses en bane til, og valg av parametere i banen. Parameteren lengde bestemmer hvor få punkter banen skal inneholde, og er med på å bestemme tidsbruken i algoritmen. Å lage en bane med et høyt antall punkter vil være raskere enn å lage en bane med få punkter. Margin-parameteren bestemmer hvor stor økning i avvik som tillates i banen, og med en stor tillat økning kan antall punkter i banen reduseres kraftig. Dette gir igjen en bane med få punkter, som tar lenger tid å lage. Dette er på grunn av at første utkast av en bane begynner med et likt antall punkt, uavhengig av valgt av parametre, gitt at samme rådatabane brukes. Ønskes en bane med få punkter, må da algoritmen redusere antall punkter flere ganger. Antall punkter i første utkast av en bane er avhengig av antall punkter rådatabanen. Det betyr at det tar lenger tid å tilpasse en bane til en rådatabane som inneholder mange punkter, enn å tilpasse en tilsvarende bane til en rådatabane som inneholder færre punkter. Tidsbruken vil også avhenge av hvor kraftig datamaskinen som kjører algoritmen er. Ved å utføre forsøk på bane2, som inneholder 1281 punkter, er en gjennomsnittlig tidsbruk for algoritmen funnet. Banen som er tilpasset rådatabanen inneholder 50 punkter, som gir en reduksjon i antall punkter på 96,1%. Margin-parameteren er ikke brukt under disse forsøkene. Det er brukt forskjellige vekter i algoritmen under forsøkene. Tidsbruken for algoritmen når banen skal optimaliseres for hastighet er vesentlig lavere enn ellers. Dette er på grunn av oppbyggingen av algoritmen. Gjennomsnittlig tidsbruk for å generere en bane med 50 punkter, tilpasset rådatabane 2 med 1281 punkter er funnet å være: 72.3 sekunder.

5.9 Mulige feilkilder

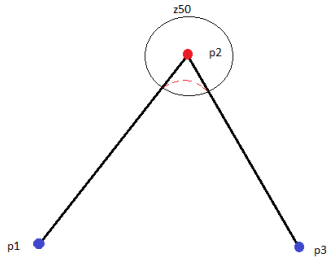
Sensoren som er brukt for å spille inn rådatabanene er ikke perfekt, og målingene fra sensoren vil naturligvis inneholde støy og unøyaktigheter. Det vil også være noe støy fra bevegelsene i hånden til den personen som har spilt inn banene. Dette kan føre til at banen blir mer hakkete og mindre jevn enn hva den perfekte banen ville vært. Dette fører igjen til at det trengs flere punkter for å beskrive all dynamikken i banen.

6 Diskusjon og forslag til videre arbeid

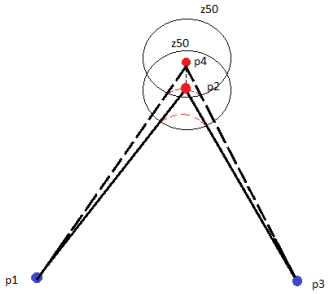
6.1 Endring av punkter ved spisse vinkler, i kombinasjon med valg av sonedata

Da sonedataen må defineres etter hvilken dynamikk roboten har, og hvor stor last den bærer på for å kunne holde en konstant hastighet fra punkt til punkt i banen, blir reell bane noe avvikende fra programmert bane. Dette gjelder spesielt for de punktene som ligger som midterste punkt i to sammenhengende vektorer som danner en spiss vinkel. Da vil en stor verdi for sonedata føre til at roboten "kutter svingen" og avviker fra programmert bane. Dette kan gi et større avvik fra rådatabanen, enn hva den programmerte banen ville gitt, dersom den ble fulgt korrekt. ABB har i sine baner satt sonedataen til den forhåndsdefinerte *z50* som passer bra nok til å følge hastigheten i banen, med en bestemt robots dynamikk. For å forbedre nøyaktigheten når en stor verdi for sonedata brukes, kunne vært en idè å flytte litt på de punktene som ligger mellom to sammenhengende vektorer som danner en spiss vinkel. Punktene kan flyttes slik at de ligger litt utenfor opprinnelig programmerte punkter. Dette kan føre til at roboten treffer opprinnelig programmert bane bedre enn dersom punktene ikke ble flyttet, gitt at det er spesifisert en sonedata som fører til at roboten kutter svingen.

De innspilte rådatabanene inneholder støy og unøyaktigeheter i målingene, samt at de kan være laget av en person med en ustø hånd. Det viser seg at de fleste spisse vinklene i banen er dannet av utstikkere, eller nærmere bestemt punkter, eller en samling av punkter som ligger utenfor en ellers rett linje eller glatt kurve. Det er derfor funnet at å implementere en slik funksjon for alle punktene i banen ikke nødvendigvis er ønskelig. Det vil være mer aktuelt å se på dette i forhold til en glattet rådatabane, hvor mye av støyen blir fjernet.
ABB har også utviklet et grafisk grensesnitt som muliggjør endringen av banen ved å flytte på punkter, etter at banen er ferdig generert. Dette kan gi brukeren muligheten til å flytte på punktene slik at de liggger optimalt plassert i forhold til spesifisert sonedata.



Figur 80.a: Original bane



Figur 80.b: Punkt p2 erstattes av punkt p4

Figur 80: Skisse av resultat ved å erstatte et punkt p2 med et utenforliggende punkt p4, begge med sonedata z50. Svart strek viser programmert bane, rød stiplet strek viser reell bane

6.2 Implementering av sirkelbuer i robotbanen

Det finnes èn annen bevegelse enn lineære bevegelser som det er mulig å bygge opp en robotbane med den dag i dag. Denne bevegelsen er en sirkelbue, se del 2.2.1. Det kunne tenkes at det hadde vært nyttig å benytte seg av en sirkelbevegelse i de tilfeller rådatabanen svinger seg gjennom rommet og danner en sirkelbue. Dette kan resultere i at banen kan få bedre nøyaktighet enn ved bruk av lineære bevegelser, i tillegg kan dette gi en bane som kan uttrykkes med færre punkter.

6.3 Tilpassing av vekter for å få optimalt resultat

Det kan arbeides videre med å tilpasse vektene som brukes i algoritmen, slik at det blir brukt noen vekter som er optimale for ønsket resultat. Dette krever kunnskaper og informasjon om hvilken oppgave roboten skal utføre og hvilke egenskaper i banen som bør optimaliseres. Samtidig krever det kunnskaper om hvor stort avvik som kan tolereres i de resterende egenskapene i banen. Det bør også tas stilling til om rådatabanen kan filtreres med et lavpassfilter før det det tilpasses en bane til denne med algoritmen.

7 Konklusjon

Ved å se på resultatene fra algoritmen presentert i denne rapporten 5, er det klart at algoritmen klarer å representere rådatabanen med færre punkter enn referansebanene, og fremdeles ha mindre avvik for hastighet, posisjon og orientering enn referansebanene. I tillegg er det mulig for algoritmen å optimalisere banen for en gitt egenskap. Dette fører til at rådatabanen kan representeres med enda færre punkter, dersom det tillates litt mer avvik for de resterende egenskapene i banen. Dette er en mulighet som ikke finnes ved generering av referansebanene gjennom algoritmen fra ABB.
En betydelig feil i referansebanene til ABB er tidsbruken i banen, og dermed også hastigheten. Referansebanene som er tilpasset rådatabanene blir gjennomført på betydelig kortere tid enn tiden som brukes for å gjennomføre rådatabanene. Da referansebanene beveger seg over omtrent samme lengde i rommet som rådatabanene, vil hastigheten i referansebanene være høyere. Se 5.6. For eksempel er rådatabane nr.2 15.33 sekunder lang, mens ABBs referansebane er 12.75 sekunder lang. Dette gir en reduksjon i tidsbruk på 16.83%. Banen fra algoritmen er 15.33 sekunder lang som tilsvarer en reduksjon i tidsbruk på 0.0%.

Tiden som brukes på å generere en bane gjennom algoritmen er avhengig av lengden på rådatabanen og ønsket lengde på tilpasset bane. Det viser seg også at det går raskere å lage en bane når den optimaliseres for hastighet. Det er gjennom forsøk funnet at gjennomsnittlig tidsbruk for generering av en bane med 50 punkter, tilpasset en rådatabane med 1281 punkter er 72,3 sekunder. Det er godt mulig at tidsbruken kan forbedres ved å skrive algoritmen i et annet programmeringsspråk, og ved å effektivisere koden videre. Ser man på resultatet i forhold til målet med Simplified Robot Programming i del 2.1.2 så finnes det at tidsbruken er innenfor rammene for hva som er akseptabelt.

Det kan dermed konkluderes med at algoritmen som er presentert i denne rapporten presterer bedre enn algoritmen som er brukt for å lage referansebanene.

Vedlegg A Forklaring til hvordan .m-filene virker

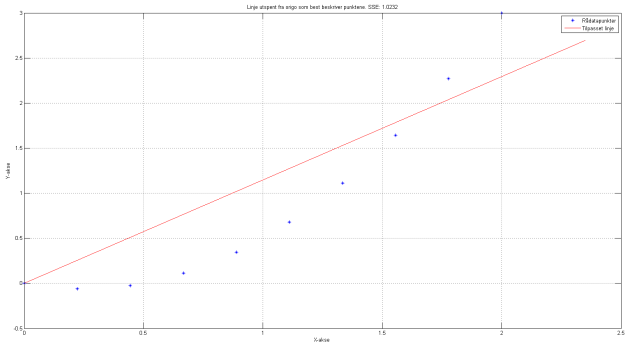
Her presenteres virkemåten for de ulike funksjonene. De generelle funksjonene fra veileder Karl Skretting er presentert i del A.1 - A.7. Del A.8 - A.29 forklarer funksjonene brukt i algoritmen.

A.1 dist2lines.m

Funksjonen brukes til å beregne kvadrert avstand fra alle punktene i rådataen til en generert banene. Hvert par med punkter i banen danner en lineær linje mellom seg, og ved å projisere alle punktene i rådataen ned på linjene som dannes i banen, beregnes den kvadrerte avstanden mellom banen og alle punktene i rådataen. Se del 2.6.3 for teori om projeksjon av punkter ned på en linje. Funksjonen returnerer også en t-vektor som forteller hvor punktene i rådatabanen er plassert i forhold til punktene i den genererte banen, sett i forhold til tid.

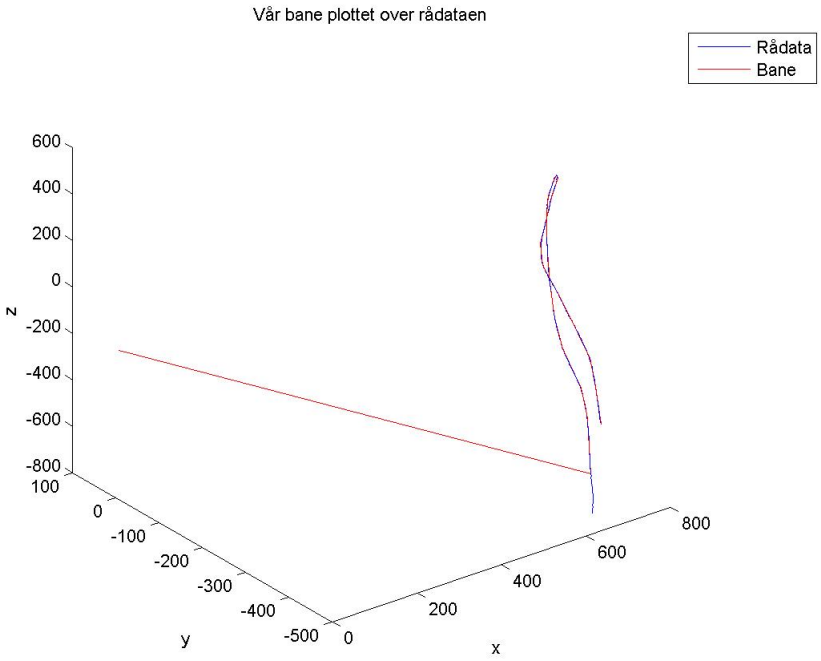
A.2 fitOneEnd.m

Funksjonen tar et sett med k punkter, X, som inngangsparameter og regner ut en linje som best beskriver punktene, fra origo og ut i rommet til et punkt P. Den baserer seg på minste kvadraters løsning. Punktet P som blir returnert er like langt ute som det punktet som er lengst ute i X. Funksjonen blir brukt for å tilpasse et endepunkt i banen, p_1 eller p_k og endepunktets nærmeste punkt p_2 eller p_{k-1} slik at linjen som dannes av punktene best beskriver punktene i rådatabanen.



Figur 81: Linje utspent fra origo som best beskriver punktene

Funksjonen virker for det meste til sin hensikt, men dersom funksjonen blir brukt for å tilpasse et par med punkter, hvor det kun eksisterer ett punkt i rådatabanen mellom disse punktene, blir resultatet veldig dårlig. Funksjonen ender da opp med å skalere et av punktene med en konstant, og virker da mot sin hensikt ved at plasseringen av punktet blir feil. Denne feilen oppstår kun når funksjonen brukes på feil måte, noe den viser seg å gjøre i *fitP.m*A.4. Se figur 82 og 83 for en illustrasjon av feilen som oppstår ved feil bruk av funksjonen. Se også del 2.3.5 for informasjon om K-means algoritmens virkemåte.



Figur 82: 3D-plot av bane. Feil grunnet funksjonen *fitOneEnd.m* ses tydelig

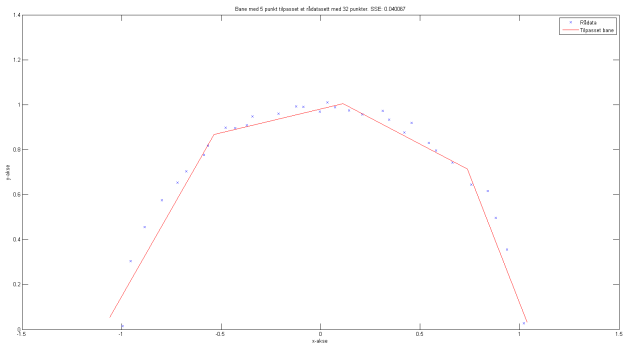
```
Start fitP 16-Mar-2015 20:14:05, X has size 409x8.
P has 25 points, we want it to have 23.
  Initial sse (K = 25) is 243.7943
It seems that replacing point 1 and point 2 by one point will change sse by 48.1773. Do this.
  sse after decrease of size of P (K = 24) is 1099276.5141
It seems that replacing point 1 and point 2 by one point will change sse by -1097902.2967. Do this.
  sse after decrease of size of P (K = 23) is 4866587.1366
Adjusting P in loop 1 gives   sse = 4866585.2996
  Final sse (K = 23) is 4866585.2996
```

Figur 83: Endring i sum av kvadrert feil i banen, grunnet feil fra funksjonen *fitOneEnd.m*

A.3 initP.m

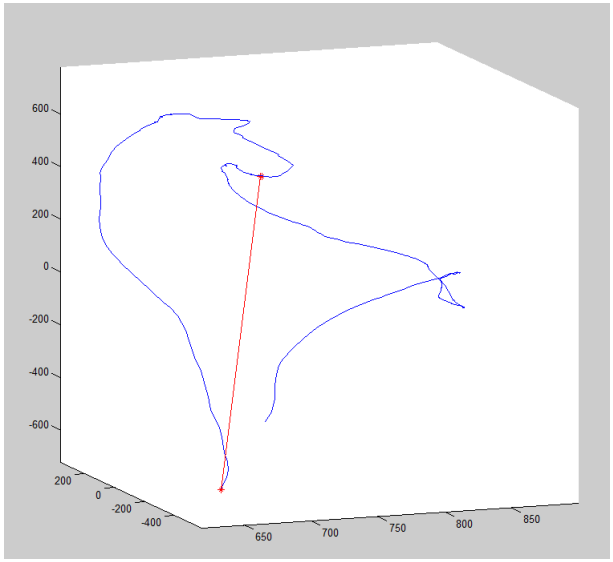
Funksjonen tar i bruk K-means clustering for å finne et utgangspunkt for å tilpasse en bane til rådataene, med K punkter. Den deler opp rådatapunktene i K-1 grupper, der alle punktene i hver gruppe hører sammen. Deretter finner den 2K-1 linjer som går tvers gjennom hver gruppe, fra endepunkt til endepunkt. Dette resulterer i en bane bestående av 2K punkter og K-1 linjer, men linjene henger ikke sammen. Videre knyttes to og to linjer sammen for å danne en kontinuerlig bane ved å finne de punktene i banen som ligger nærmest hverandre, og erstatte disse med et punkt som tilsvarer gjennomsnittet av det paret med punkter som knyttes sammen. Dette resulterer i en bane med K punkter og K-1 linjer. Se figur 84 for en illustrasjon av resultatet fra *initP.m*

Funksjonen gir ofte et tilfredstillende resultat, men grunnet inndelingen av punkter med K-means algoritmen kan den til tider produsere et uønsket resultat med vesentlige feil for enkelte punkter. Dette kommer av at punkter som ligger langt fra hverandre i tid fremdeles kan grupperes i samme gruppe gjennom K-means algoritmen, da den ser på likhet i punktene, og ikke avstand i tid. Denne feilen kjennetegnes ved at det trekkes en linje fra et punkt i banen til et annet punkt i banen som ligger langt borte fra det første



Figur 84: 5 punkts bane laget med initP. Rådatabane består av 32 punkter

punktet. Dette skjer oftest for de banene som svinger seg gjennom rommet og når en posisjon som er ganske lik et punkt tidligere i banen. Se figur 85 for en illustrasjon av feilen som kan oppstå.

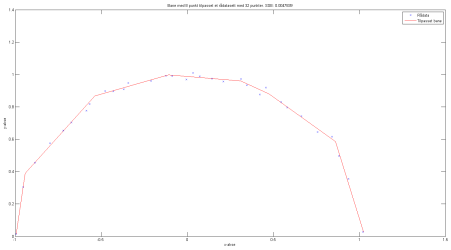


Figur 85: 3D-plot av bane. Feil i banen gitt av initP.m og K-means algoritmen er markert med rødt

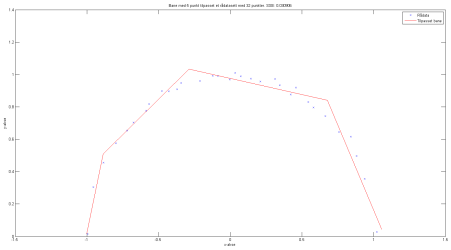
A.4 fitP.m

Funksjonen bruker et eksisterende forslag til en bane til å utforme en bane som passer bedre til rådataen. Funksjonen har mulighet til å legge til punkter, eller trekke fra punkter i den eksisterende banen og krever dermed at ønsket antall punkter i banen spesifiseres. Funksjonen legger til eller trekker fra punkter, dersom dette er spesifisert, der det er størst ønskelig-, eller minst uønsket innvirkning på feilen, målt i vektet sum av kvadrert feil. Til slutt går funksjonen gjennom alle punktene i banen og ser om noen av punktene kan flyttes litt og dermed gi en ønsket innvirkning på feilen i banen. For hver gang funksjonen flytter de punktene som gir ønsket innvirkning på feilen i banen, vil en ny gjennomgang

av punktene muligens gi flere punkter som kan flyttes, da den nye kombinasjonen av punkter sannsynligvis også har noen punkter som kan flyttes. Det blir derfor utført en flytting av punktene i banen et bestemt antall ganger, som spesifisert av parameteren *maxLoops*.



Figur 86.a: bane utvidet til 8 punkt med fitP



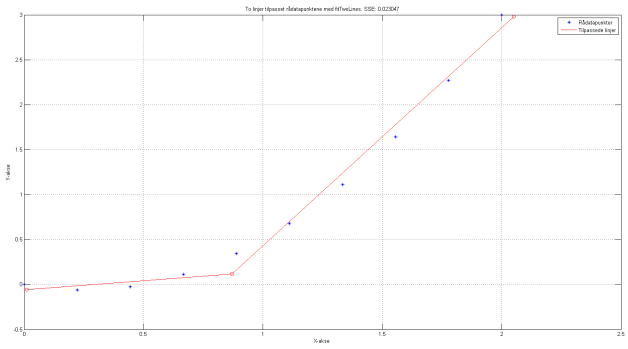
Figur 86.b: bane redusert til 5 punkt med fitP

Figur 86: En bane med 5 punkter utvidet til 8 punkter og redusert igjen til 5 punkter med *fitP.m*. Originalbanen fra *initP.m* er vist i figur 84

Funksjonen virker stort sett til sin hensikt, men kan til tider produsere resultater som virker mot sin hensikt og resulterer i vesentlige feil. Den ene feilen kommer av at funksjonen kaller en annen funksjon som heter *fitOneEnd.m* med kun 1 punkt i rådatabanen som inngangsparameter. Feilen som oppstår er forklart og illustrert i del A.2. Den andre feilen ved funksjonen er at den har muligheten til å fjerne og erstatte første og andre punkt p_1 og p_2 i banen eller siste- og nest siste punkt p_k og p_{k-1} i banen med et nytt punkt. Dette er ikke ønskelig da det ønskes at banen skal starte fra første punkt i rådatabanen og ende i siste punkt i rådatabanen.
Det er også en feil i koden som fører til feilmeldinger når funksjonen *fitMiddle2.m* kalles. Denne feilmeldingen oppstår når funksjonen prøver å tilpasse et midterste punkt p_k som ligger mellom to andre punkter p_{k-1} og p_{k+1} til rådataen, når det kun eksisterer ett punkt i rådataen mellom p_{k-1} og p_{k+1}.

A.5 fitTwoLines.m

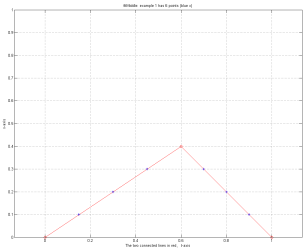
Funksjonen finner tre punkter P1, P2 og P3 som sammen danner to linjer som best beskriver et sett med rådatapunkter. Dersom det legges ved en inngangsvariabel P med 3 punkter av samme dimensjon som rådataen, vil funksjonen ta utgangspunkt i disse punktene for å danne de to linjene som beskriver dataen. Dersom det ikke legges ved en P som beskrevet vil funksjonen finne de to punktene som ligger lengst fra gjennomsnittet i rådataen, bruke disse som endepunkter og finne midterste punkt ved hjelp funksjonen *fitMiddle2.m* A.7. Både midtpunktet og endepunktene justeres for å minimere sum av kvadrert feil ved hjelp av funksjonen *fitOneEnd.m*A.2. Se figur 87 for illustrasjon av resultatet fra funksjonen. Funksjonen brukes kun i funksjonen *initP.m* A.3, og kun dersom ønsket lengde på banen er 3 punkter. Denne funksjonen vil derfor sannsynligvis ikke blir brukt da det ikke er realistisk å tilpasse en bane til en rådatabane med kun 3 punkter.



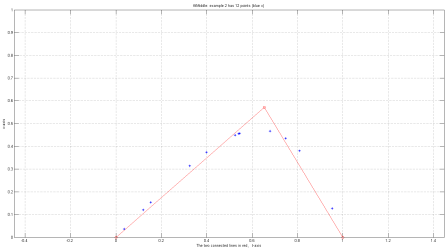
Figur 87: To linjer tilpasset et sett med punkter med funksjonen fitTwoLines

A.6 fitMiddle.m

Funksjonen finner et passende midtpunkt for å tilpasse to linjer til et sett med punkter med en lineær spline.Funksjonen returnerer midtpunktet p_m som er funnet optimalt for rådataen, og en objektfunksjon $f(t)$ som definerer linjene fra et startpiunkt p_0 til midtpunktet p_m og fra p_m til sluttpunktet p_1. Funksjonen baserer seg på å minimere sum av kvadrert feil. Ofte må settet med punkter som det skal tilpasses linjer til normaliseres, skaleres og roteres. Dette gjøres ved hjelp av funksjonen *fitMiddle2.m* A.7 før funksjonen *fitMiddle.m* kalles. Se figur 88 for en illustrasjon av resultatet fra *fitMiddle.m*.



Figur 88.a: fitMiddle

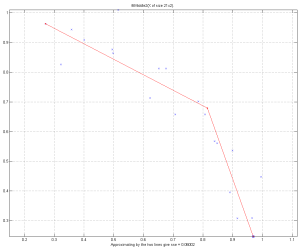


Figur 88.b: fitMiddle

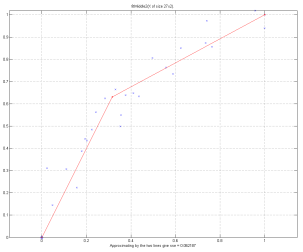
Figur 88: Midtpunktet funnet med fitMiddle gir beste tilnærming til to sammenhengende linjer som best beskriver punktene

A.7 fitMiddle2.m

Funksjonen finner et passende midtpunkt for å tilpasse to linjer til et sett med punkter, gitt at startpunkt for den første linjen og sluttpunkt for den andre linjen er spesifisert. Med andre ord finner funksjonen et punkt P2 som ligger mellom P1 og P3, som sammen danner to linjer, en fra P1 til P2, og en fra P2 til P3. P2 finnes ved hjelp av funksjonen *fitMiddle.m* A.6 slik at sum av kvadrert feil for de to linjene blir minimert. Funksjonen *fitMiddle2.m* tar seg også av normalisering, skalering og rotering før *fitMiddle.m* kalles. Se figur 89 for illustrasjon av resultatet fra *fitMiddle2.m*.



Figur 89.a: fitMiddle2



Figur 89.b: fitMiddle2

Figur 89: Midtpunktet funnet med fitMiddle2 gir beste tilnærming til to sammenhengende linjer som best beskriver punktene

A.8 dist2lines2.m

Funksjonen brukes til å beregne vektet kvadrert feil fra alle punktene i rådataen til en generert bane. Hvert par med punkter i banen danner en lineær linje mellom seg, og ved å projisere alle punktene i rådataen ned på linjene som dannes i banen, beregnes den vektede kvadrerte feilen mellom banen og alle punktene i rådataen. Se del 2.6.3 for teori om projeksjon av punkter ned på en linje. Se del 2.4.3 for informasjon om veketet sum av kvadrert feil. Funksjonen kan brukes på et sett med data av hvilken som helt dimensjon, enten det er punkter som inneholder posisjon og orientering, eller kun posisjon. Funksjonen returnerer også en t-vektor som forteller hvor punktene i rådatabanen er plassert i forhold til punktene i den genererte banen, sett i forhold til tid.

A.9 fitOneEnd2.m

Denne funksjonen tar et sett med k punkter, X, som inngangsvariabel og regner ut en linje som best beskriver punktene, fra origo og ut i rommet til et punkt P. Den baserer seg på minste kvadraters løsning. Punktet P som blir returnert er like langt ute som det punktet som er lengst ute i X. Funksjonen blir brukt for å tilpasse et endepunkt i banen, p_1 eller p_k og endepunktets nærmeste punkt p_2 eller p_{k-1} slik at linjen som dannes av punktene best beskriver punktene i rådatabanen. Funksjonen baserer seg på koden i *fitOneEnd.m* A.2, men returnerer vektet sum av kvadrert feil istedet for vanlig sum av kvadrert feil.

A.10 initP2.m

Denne funksjonen baserer seg på koden i *initP.m* A.3, men det er gjort noen få endringer i funksjonen. *initP2.m* bruker ikke K-means algoritmen for å dele inn punktene i banen i grupper slik som i *initP.m*, men deler de istedet inn i grupper i forhold til hvor punktene ligger i tid. Den er også skrevet om for å ta i bruk vektet sum av kvadrert feil ved optimalisering av punktene i banen, istedet for vanlig sum av kvadrert feil som i *initP.m*.

A.11 fitP2.m

Denne funksjonen brukes et eksisterende forslag til en bane til å utforme en bane som passer bedre til rådataen. Funksjonen har mulighet til å legge til punkter, eller trekke fra punkter i den eksisterende banen og krever dermed at ønsket antall punkter i banen spesifiseres.
Funksjonen legger til eller trekker fra punkter, dersom dette er spesifisert, der det er størst ønskelig-, eller minst uønsket innvirkning på feilen, målt i vektet sum av kvadrert feil. Til slutt går funksjonen gjennom alle punktene i banen og ser om noen av punktene kan flyttes litt og dermed gi en ønsket innvirkning på feilen i banen. For hver gang funksjonen flytter de punktene som gir ønsket innvirkning på feilen i banen, vil en ny gjennomgang av punktene muligens gi flere punkter som kan flyttes, da den nye kombinasjonen av punkter sannsynligvis også har noen punkter som kan flyttes. Det blir derfor utført en flytting av punktene i banen et bestemt antall ganger, som spesifisert av parameteren *maxLoops*.

Funksjonen baserer seg på koden i *fitP.m* A.4, men det har blitt utført noen endringer som fører til at problemene som fantes ved bruk av *fitP.m* ikke oppstår. Se A.4 for videre informasjon om problemene som er ordnet. Funksjonen tar som forklart også i bruk vektet sum av kvadrert feil som optimaliseringskriterie, istedet for vanlig sum av kvadrert feil som brukes i *fitP.m*. Dette fører til at en av egenskapene ved banen kan optimaliseres på bekostning av de andre egenskapene.

A.12 fitMiddle_2.m

Denne funksjonen finner et passende midtpunkt for å tilpasse to linjer til et sett med punkter med en lineær spline. Funksjonen returnerer midtpunktet p_m som er funnet optimalt for å beskrive rådataen, og en objektfunksjon $f(t)$ som definerer linjene fra et startpunkt p_0 til midtpunktet p_m og fra p_m til sluttpunktet p_1. Funksjonen baserer seg på å minimere vektet sum av kvadrert feil og baserer seg på koden i *fitMiddle* A.6. Endringen som er gjort i funksjonen er å implementere en vektet sum av kvadrert feil, i stedet for en vanlig sum av kvadrert feil. Ofte må settet med punkter som det skal tilpasses linjer til normaliseres, skaleres og roteres. Dette gjøres ved hjelp av funksjonen *fitMiddle2_2.m* A.13 før funksjonen *fitMiddle.m* kalles.

A.13 fitMiddle2_2.m

Denne funksjonen finner et passende midtpunkt for å tilpasse to linjer til et sett med punkter, gitt at startpunkt for den første linjen og sluttpunkt for den andre linjen er spesifisert. Med andre ord finner funksjonen et punkt P2 som ligger mellom P1 og P3, som sammen danner to linjer, en fra P1 til P2, og en fra P2 til P3. P2 finnes ved hjelp av funksjonen *fitMiddle_2* A.12, slik at vektet sum av kvadrert feil for de to linjene blir minimert. Funksjonen baserer seg på koden i *fitMiddle2.m* A.7, men tar i bruk vektet sum av kvadrert feil som optimaliseringskriterie, i stedet for vanlig sum av kvadrert feil.

A.14 quaternionMath.m

Denne funksjonen inneholder matematiske formler for å konvertere kvaternioner til eulervinkler og omvendt, samt å addere/subtrahere orienteringer, enten de er uttrykt som kvaternioner eller eulervinkler. Funksjonen baserer seg på de matematiske formlene for konvertering mellom kvaternioner og eulervinkler i del 2.6.4

A.15 lesLog.m

Denne funksjonen er laget for å lese dataen som finnes i en .log fil. Funksjonen leser filen som en kommaseparert tekstfil. Det spesifiseres at hvert kommaseparert element er et flyttall, og tallene lastes inn i en cellestruktur. Cellestrukturen konverteres så til en matrise, for enklere å kunne arbeide med dataen. Data og antall punkter i rådatabanen returneres.

A.16 lesMod.m

Filene som leses med denne funksjonen er delt inn i to deler. Èn del hvor alle punkter, verktøy og arbeidsobjekt er definert, og èn main-funksjon hvor alle bevegelsene er definert. I den første delen hentes informasjon om posisjon og orientering for alle punkt, mens tidsbruk fra punkt til punkt finnes i main-funksjonen. Ved å lese inn tekststrengene med riktig formatspesifikasjon, er det mulig å hente ut relevant informasjon og forkaste urelevant informasjon. For å hente ut tidsbruken for hver bevegelse, er det også nødvendig å beregne hvor i dokumentet main-funksjonen begynner, da denne krever en annen formatspesifikasjon for å lese relevant data.

A.17 SjekkVinkel.m

Funksjonen tar som parameter to tredimensjonale vektorer, eller en samling av to og to tredimensjonale vektorer. Vinkelen mellom vektorene regnes ut, og returneres. Denne funksjonen er brukt for å undersøke vinkelen mellom hvert par med vektorer i rådatabanen og banen generert fra algoritmen i denne rapporten.

A.18 updateP.m

Denne funksjonen flytter punkt i en bane, for å prøve å forbedre sum av kvadrert feil, sett i forhold til rådatabanen. Den baserer seg på koden i *fitP.m* A.4, men det er utført endringer i koden som medfører at problemene i A.4 ikke oppstår.

A.19 increaseP.m

Denne funksjonen legger til punkt i en bane, med den hensikt å forbedre vektet sum av kvadrert feil, sett i forhold til punktene i rådatabanen. Funksjonen baserer seg den delen av koden i *fitP.m* A.4 som oppdaterer punktene i banen. Forskjellen fra *fitP.m* er at denne funksjonen prøver å minimere vektet sum av kvadrert feil, i stedet for vanlig sum av kvadrert feil.

A.20 decreaseP.m

Denne funksjonen fjerner punkter i en bane, med den hensikt å ende opp med så få punkt som mulig i en bane, samtidig som vektet sum av kvadrert feil, sett i forhold til rådatabanen minimeres. Funksjonen baserer seg på den delen av koden i *fitP.m*A.4 som fjerner punkter. Forskjellen fra *fitP.m* er at denne funksjonen prøver å minimere vektet sum av kvadrert feil, i stedet for vanlig sum av kvadrert feil. Det er også gjort noen endringer i koden som fører til at problemene som er beskrevet i A.4 ikke oppstår.

A.21 finnFeil.m

Denne funksjonen finner diverse mål på godheten til en bane. Her blir hastighet, posisjon og orientering evaluert. Det blir i tillegg undersøkt hvor stor del av endringene i orientering som utføres i banen, i forhold til i rådatabanen. Lengden på alle linjene i banen blir også undersøkt, da dette forteller hvor stor del av tilbakelagt posisjonsendring i rådatabanen som blir gjenskapt i banen. For å finne avviket i hastighet kalles funksjonen *finnFeilHast2.m* A.23. Funksjonen har også mulighet til å lage figurer som illustrerer hastighet, posisjon og orientering i banen.

A.22 finnFeil2.m

Denne funksjonen er en utvidet versjon av *finnFeil.m* A.21 . Funksjonen kan beregne mål på godheten til to baner, og returnere disse. I tillegg kan funksjonen lage figurer som illustrerer hastigheten, orienteringen og posisjonen i banene. Funksjonen kan også lagre figurene på en spesifisert filbane. Denne funksjonen er brukt for å produsere en del av figurene som brukes i resultatdelen i denne rapporten.

A.23 finnFeilHast2.m

Denne funksjonen beregner avviket i hastighet i en bane, sett i forhold til hastigheten i en rådatabane. Avviket beregnes ved først å finne ut hvor i tid alle punktene i rådatabanen er plassert i forhold til punktene i banen. Hastigheten holdes konstant fra punkt til punkt i banen, så avviket finnes ved å finne differansen i hastighet fra hvert par med punkter i rådatabanen til hastigheten i tilhørende par i banen. Differansen kan så kvadreres og

summeres for å returnere sum av kvadrert feil for hastigheten. Gjennomsnittlig avvik og maksimalt avvik i hastighet finnes også.

A.24 utvikling_feil.m

Dette er et matlabskript som er laget for å sjekke utviklingen i feilen i en bane ved reduksjon av antall punkter i banen. Den bruker funksjonen *algoritme.m* A.28 for å generere en bane, og for hver gang det utføres en reduksjon i antall punkter i banen, blir feilen for banen undersøkt. Til slutt returneres en struct-variabel med mål på feilen i banen for ulikt antall punkter i banen. Skriptet lager så figurer som illustrerer utviklingen av feilen i banen, både for hastighet, posisjon og orientering. Det lages også et par figurer som illustrerer hvor stor del av lengden i rådatabanen som utføres i banen, og hvor stor del av endringen i orientering i rådatabanen som utføres i banen.

A.25 LagRapidBane.m

Denne funksjonen lager en *RAPID*-kode av en generert bane slik at den kan kjøres på en robot, eller simuleres i robotstudio. Ved å bruke riktig syntaks og formatering av teksten, blir punktene i banen definert med posisjon og orientering i starten av dokumentet. Det skrives også en main-funksjon til dokumentet som definerer alle bevegelsene fra punkt til punkt i banen, med ønsket tidsbruk for bevegelsen. *RAPID*-koden kan lastes inn i *Robotstudio* og simuleres der, så lenge roboten som brukes i simuleringen har en rekkevidde som rekker alle punktene, og et arbeidsobjekt som kalles *wBord* blir satt opp. Punktene i banen er spesifisert i forhold til koordinatsystemet i arbeidsobjektet, og arbeidsobjektet vil derfor bestemme hvor i rommet punktene er plassert.

A.26 lesABBbane.m

Denne funksjonen er laget for å få en rask innlesning av en rådatabane og tilhørende referansebane. Algoritmen kan så brukes for å tilpasse en bane til rådatabanen. Referansebanen kan brukes for å evaluere banen fra algoritmen. Brukeren får opp en filvelger som brukes til å finne frem til ønsket rådatafil av typen .Mod. Referansebanen som tilhører rådatafilen blir så lastet inn automatisk.

A.27 baneMedSpesifisertLengde.m

Denne funksjonen er laget for å raskt og enkelt lage en bane, med tilhørende figurer som illustrerer avvik og egenskaper i banen. Funksjonen har også en mulighet for å lagre figurene automatisk. Flere av figurene i resultatdelen i denne rapporten er laget med denne funksjonen. Funksjonen kaller *Algoritme5* A.28 , med ønskede parametre og lager en bane. Videre kalles funksjonen *finnFeil2.m* A.22 som produserer og lagrer figurene. Valg av rådatabane skjer gjennom funksjonen *lesABBbane.m* A.26 , som lar brukeren

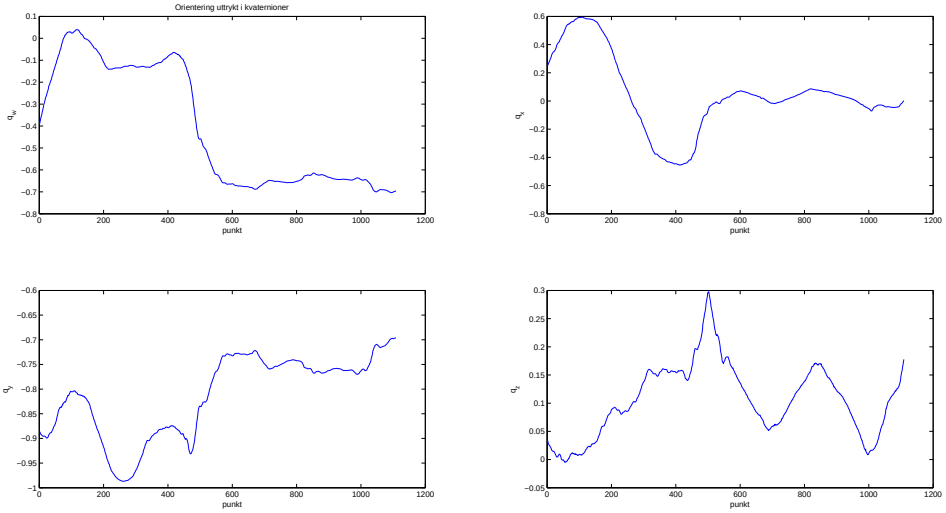
finne frem til en rådatabane ved hjelp av en filvelger. Det er mulighet til å spesifisere en vekt som ønskes brukt i algoritmen, sammen med ønsket lengde på banen eller verdi for margin-parameteren.

A.28 Algoritme5.m

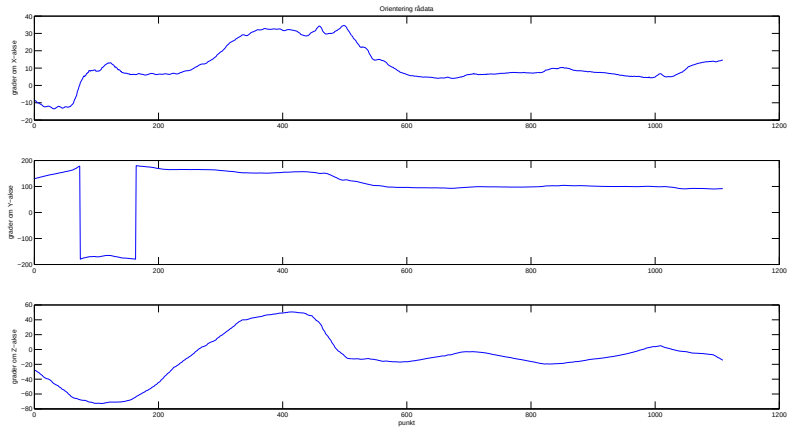
Dette er algoritmen som er utviklet i forbindelse med denne rapporten. Algoritmen baserer seg i stor grad på flere funksjoner skrevet av Karl Skretting ved Universitetet i Stavanger. Virkemåten for algoritmen er beskrevet i del 4.

A.29 korrigerData.m

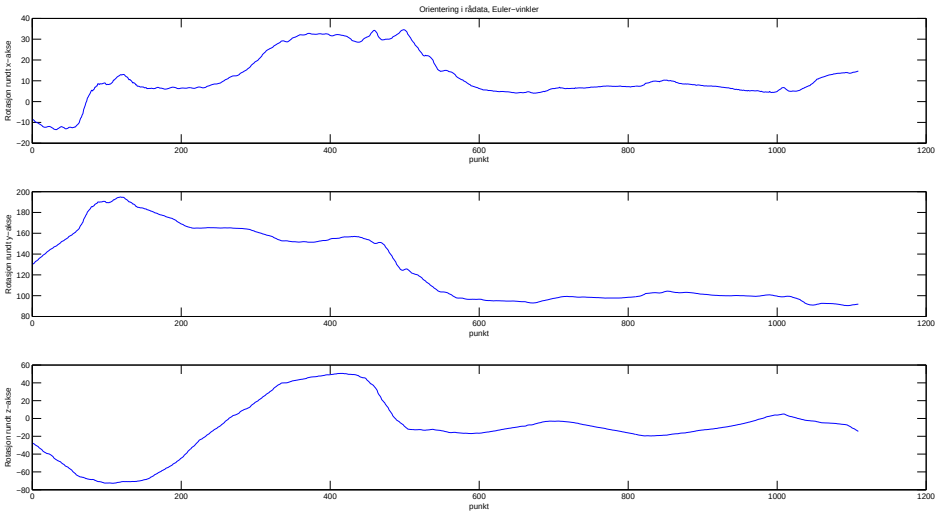
Når det tilpasses en bane til et sett med rådata, blir orienteringen uttrykt i kvaternioner. Dette er for å få en entydig beskrivelse av orienteringen i rommet. Videre når banens godhet skal evalueres, blir orienteringen konvertert til Euler-vinkler for å kunne tolkes på en intuitiv måte.
Når orienteringen konverteres til Euler-vinkler, blir orienteringen uttrykt som en rotasjon i området $[-180°, 180°]$ rundt X-, Y- og Z-aksen. Dette betyr at en rotasjon på f.eks. $\alpha = 200°$ rundt en bestemt akse i et bestemt punkt tolkes som $\alpha = -160°$. Når avviket i orienteringen skal beregnes for dette punktet, blir resultatet $avvik = 200° - (-160°) = 360°$. Se fig. 91 og fig. 90. Dette er et stort og falskt avvik som må håndteres for å bestemme banens godhet korrekt.
Ved å identifisere hvilke punkter som er påvirket av denne feilen, kan det adderes på 360°, slik at rotasjonen i punktet uttrykt i euler-vinkler blir $\alpha = -160° + 360° = 200°$ og avviket beregnes til: $avvik = 200° - 200° = 0°$. Se fig. 92
Denne korreksjonen blir utført kun for å få en riktig beskrivelse av avviket i orientering for hvert punkt, og påvirker ikke orieteringen i hvert punkt i banen. Orienteringen vil uansett være riktig definert i punktet, slik at ønsket orientering oppnås. Roboten vil ikke snu verktøyet 360°, selv om orienteringen i euler-notasjon er rotert 360° fra ett punkt til et annet. Dette er fordi orienteringen uttrykkes i kvaternioner, som er en entydig beskrivelse av orienteringen i rommet og roboten bestemmer den enkleste endringen i rotasjon rundt aksene for å oppnå denne orienteringen.



Figur 90: Orientering i rådataen, uttrykt i kvaternioner



Figur 91: Feil i orientering når orienteringen blir konvertert fra kvaternioner til eulervinkler



Figur 92: Korrigert orientering ved konvertering fra kvaternioner til eulervinkler

Vedlegg B Zip-fil med resultater for bane2 til bane7, matlabkode og rådatafiler

Det er vedlagt en zip-fil som inneholder kildekode og rådatafiler for banene innspilt 05.11.2013. Det er i tillegg laget et dokument som inneholder figurer som illustrerer resultatet for bane2 til bane7, som også ligger i zip-filen.

Referanser

[1] C. Edwards and D. Penney, "Elementary linear algebra," 1988.

[2] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. Wiley, 2006.

[3] A. Robotics, *Rapid referance manual Overview On-line*. Västerås: ABB Robotics.

[4] ABB Robotics, "Simplified Robot Programming data sheet," 2015.

[5] A. Robotics. (2010) Technical reference manual, RAPID Instructions, Functions and Data types.

[6] M. Kutner, C. Nachtsheim, J. Neter, and W. Li, *Applied linear statistical models*, fifth edit ed. Singapore: Mcgraw-Hill/Irwin, 2005.

[7] R. J. Carroll, D. Ruppert, L. A. Stefanski, and C. M. Crainiceanu, *Measurement Error in Nonlinear Models: A Modern Perspective, Second Edition*. CRC Press, 2006.

[8] M. Sarfraz, *Interactive Curve Modeling*. London: Springer, 2008.

[9] MarianSigler, "Bezier curve," 2006. [Online]. Available: http://en.wikipedia.org/wiki/File:Bezier_curve.svg

[10] R. Duda, P. Hart, and D. Stork, *Pattern classification*, 2012.

[11] Hotmath, "Vektorfigurer." [Online]. Available: http://hotmath.com/hotmath_help/topics/adding-and-subtracting-vectors.html

[12] O. y. Dick, "Projeksjon," 2005. [Online]. Available: https://snl.no/projeksjon/geomatikk

[13] S. norske Leksikon, "Projeksjon_matematikk," 2005. [Online]. Available: https://snl.no/projeksjon%2Fmatematikk

[14] G. Stang, "Projeksjon_figur." [Online]. Available: http://www.ling.upenn.edu/courses/ling525/linear_algebra_review.html

[15] K. E. Aubert. (2011, Oct.) hyperkomplekse tall. [Online]. Available: https://snl.no/hyperkomplekse_tall

[16] J. van Oosten. (2012) Understanding Quaternions. [Online]. Available: http://3dgep.com/understanding-quaternions/

[17] B. Martin, "Maths - Conversion Quaternion to Euler - Martin Baker." [Online]. Available: http://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToEuler/index.htm

[18] S. Arietta, "Quaternions, CS445: Graphics." [Online]. Available: http://www.cs.virginia.edu/~gfx/Courses/2010/IntroGraphics/Lectures/29-Quaternions.pdf

[19] B. Martin, "Maths - Conversion Euler to Quaternion - Martin Baker." [Online]. Available: http://www.euclideanspace.com/maths/geometry/rotations/conversions/eulerToQuaternion/index.htm

[20] Polhemus, "Polhemus Electromagnetics." [Online]. Available: http://polhemus.com/applications/electromagnetics/

[21] A. Robotics, "Simplified Robot Programming - Application Equipment and Accessories - Robotics — ABB."

[22] ——, *SRP Final external*, 2015, no. ABB.