NTNU
Det skapende universitet

Synkronisering av robotmanipulator og virtuelt kamera

Erik Hogstad

Master i teknisk kybernetikk
Oppgaven levert: Juli 2010
Hovedveileder: Geir Mathisen, ITK
Biveileder(e): Sigurd Fjerdingen, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Oppgavetekst

Med utgangspunk i StatoilHydros robotlab for topside operasjoner under utvikling av SINTEF Anvendt Kybernetikk, ta for seg strategier for autonom styring av et virtuelt kamera i et simulert miljø for på best mulig måte a vise operasjonen til operatøren, noe som vil avhenge av hvilken operasjon som skal utføres (kontekst).
Oppgaven består av følgende punkter:
-Se på metoder for master/slave-synkronisering av robotmanipulatorer og intelligent kamerastyring innen spillteknologi.
-Foreslå en, eller et sett, strategier for kontekstbasert autonom styring av et virtuelt "flytende" kamera.
 -a. Kartlegge hvile grunnleggende typer operasjoner som eksisterer i robotlaben.
 -b. Transisjoner mellom forskjellige visninger må skje på fornuftig vis og til fornuftige tider.
-Implementere designet som en modul med et grensesnitt som passer robotlabens eksisterende grensesnitt.
-Teste og analysere Implementasjonen.

Oppgaven gitt: 16. februar 2010
Hovedveileder: Geir Mathisen, ITK

Sammendrag

Denne masteroppgaven tar for seg synkronisering av robotmanipulatorer og autonom styring av et virtuelt kamera med utgangspunkt i Statoils robotlab for topside operasjoner under utvikling av SINTEF Anvendt Kybernetikk. Masteroppgaven beskriver hvilke operasjoner som kan utføres i laben, beskriver metoder for master/slave synkronisering av robotmanipulatorer, tar for seg strategier for autonom styring av et virtuelt kamera i et simulert miljø. En strategi for styring av et virtuelt kamera er implementert og testet.

Innhold

Figurer

Introduksjon

Denne masteroppgaven tar utgangspunkt i Statoils robotlab for topside operasjoner som er under utvikling av SINTEF Anvendt Kybernetikk. Robotlaben består av to arbeidsroboter som kan utføre en rekke operasjoner på en ubemannet platform. Det er ønskelig å kunne benytte en av robotene i laben til å styre et autonomt kamera som viser et bilde av operasjonen den andre roboten utfører. Et autonomt kamera er ønskelig fordi det vil lette arbeidsoppgavene til operatøren som styrer arbeidsroboten og det vil kunne gi et bedre bilde enn om kameraet styres manuelt. Masteroppgaven beskriver først hvilke operasjoner som kan utføres i robotlaben, metoder for master/slave synkronisering av robotmanipulatorer og strategi for kamerastyring ut i fra hva som vises i kamerabildet. Deretter beskrives strategier for virtuell kamerastyring i spill og virtuelle filmmiljøer. En simulator for en arbeidsrobot og et virtuelt kamera er laget, og en strategi for autonom styring av et virtuelt kamera er implementert og testet i simulatoren. Et virtuelt kamera er valgt for å gjøre det mulig å teste strategier uten at man trenger å bekymre seg for begrensningene til en robotarm. Dette gjør den initielle utviklingen og implementasjonen enklere. Målet med simulatoren er å teste en av de beskrevne strategiene for styring av et autonomt virtuelt kamera. Implementasjonen ble gjort i C++ og simulatoren ble laget ved hjelp av et simuleringsbibliotek for stive legemer, og et 3D bibliotek for å vise visualisere simuleringen. Resultatene av simuleringen viser at det autonome virtuelle kameraet følger arbeidsroboten og holder arbeidsverktøyet til denne roboten innenfor kamerabildet under operasjonen.

Del I

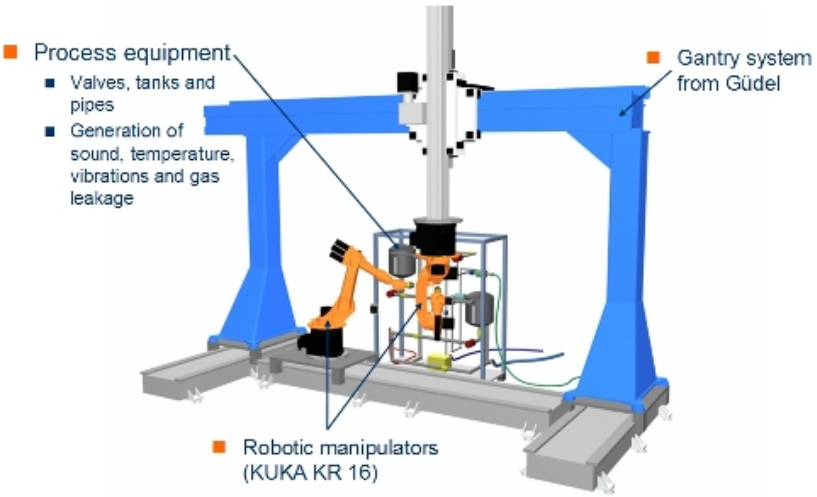
Bakgrunn

Kapittel 1

Robotlaben

1.1 Robotene

Robotlaben er basert på et nytt konsept for en fjernstyrt olje og gassplatform utviklet av Statoil. Med finansiering fra Statoil utviklet SINTEF og NTNU robotlaben for å drive forskning på robotplatformen. «Hensikten med robotlaben er å demonstrere og støtte opp om forskning på fjernstyrt inspeksjon og vedlikehold av prosessutstyr.»[Kyr08]. Robotlaben består av en traverskran som kan plassere robotmanipulatorer på et vilkårlig sted innenfor arbeidsområdet til kranen. Robotlaben består av en traverskran, gantry, med tre lineære akser, og to KUKA KR 16 robotmanipulatorer med seks frihetsgrader, Figur 1.1.



Figur 1.1: Robotlaben med prosessutstyr

I tillegg til robotsystemet består laben av en seksjon med prosessutstyr som benyttes til å gjenomføre forskjellige operasjoner med en rekke spesialverktøy utviklet av SINTEF.

Robotlaben kan fjernstyres og gir operatøren sensordata, kamerabilder og 3D-modeller i sanntid, operatøren kan utføre forhåndsdefinerte oppgaver, eller ta manuell kontroll over robotens bevegelser.

1.2 Operasjoner

Roboten kan utføre flere typer operasjoner i laben, og det er utviklet en rekke spesialverktøy som er plassert i et verktøybyttesystem slik at roboten selv kan endre verktøy for de forskjellige oppgavene. Operasjonene arbeidsroboten kan utføre er de følgende.

- Vibrasjonsmåling ved hjelp av laser vibrometer.
- Vibrasjon- og temperatur-måling med kombinert temperatur og vibrasjonssensor.
- Temperaturmåling ved hjelp av et varmefølsomt (IR) kamera.
- Gassdeteksjon med en CO_2 detektor.
- Åpne/Lukke ventiler
- Gripe objekter

Operasjonene for å åpne og lukke ventiler utføres ved å bruke et verktøy spesialutviklet ved SINTEF. Et generelt gripeverktøy er også tilgjengelig. Med dette gripeverktøyet er det mulig å bruke roboten til å plukke opp en ball fra gulvet automatisk.

1.3 Virtuelt Kamera

For å vise operatøren hvordan operasjonene utføres, og muliggjøre manuell kontroll av vedlikeholdsroboten er det ønskelig å ha et følgekamera på sekudærroboten. Et autonomt kamera på sekundærroboten ønskes for at operatøren skal få et bedre bilde av situsjonen enn det stasjonære kameraer kan gi. Med et autonomt kamera kan man oppnå dette uten at operatøren trenger å flytte fokus fra operasjonen som skal utføres. Et virtuelt kamera er valgt for å utforske hvordan kameraføringen kan være uten at det er nødvendig å styre en robot. Det virtuelle kameraet vil ikke være begrenset

av arbeidsrommet til sekundærroboten og vil dermed være en god måte å utforske strategier for autonom styring uten at man uten at man trenger å ta hensyn til arbeidsrobotens begrensninger. Det virtuelle kameraet skal følge vedlikeholdsrobotens bevegelser og gi et godt bilde til operatøren. Målsetningene for det virtuelle kameraet er at det skal følge arbeidsverktøyet til robotarmen og ha dette i bildet under hele operasjonen. En optimal oppførsel for det virtuelle kameraet må oppfylle følgende.

- Må følge arbeidsrobotens bevegelser
- Må ha arbeidsverktøyet i kamerabildet
- Må ikke kollidere med arbeidsroboten eller prosessutstyret
- Må ha et fornuftig bildeutsnitt
- Må holde en fornuftig avstand fra robotarmen
- Må ha en fornuftig synsvinkel i forhold til arbeidsverktøyet
- Må ha kontinuerlig bane

At det virtuelle kameraet må ha en kontinuerlig bane vil si at man ikke kan ha plutselige hopp i kameraposisjon. For å unngå dette er det nødvendig å sørge for at det virtuelle kameraet alltid befinner seg i en posisjon det kan bevege seg bort i fra samtidig som de andre kravene er oppfyllt. Avstanden det virtuelle kameraet holder fra arbeidsverktøyet må være slik at kravene til bildeutsnitt oppfylles. Bildeutsnittet skal være slik at man til enhver tid kan se arbeidsverktøyet. Det er også ønskelig å vise hvor arbeidsverktøyet befinner seg i arbeidsrommet til roboten og hvor det er i forhold til prosessutstyret i laben. For å oppnå dette er det ikke ønskelig at kameraet befinner seg lenger fra arbeidsvertøyet enn maksimalt en meter.

Del II

Literaturstudie

Kapittel 2

Master/Slave synkronisering

Master/slave synkronisering er basert på en kommunikasjonsmodell hvor en enhet eller prosess kontrollerer en eller flere enheter eller prosesser. Begrepet master/slave synkronisering brukes i sammenhenger mest relevante for robotlaben om et system hvor en robotmanipulator opererer som master, og en eller flere robotmanipulatorer er slaver.

2.1 Posisjonsbasert styring

Denne type robotstyring benyttes ofte for å koordinere bevegelsene til robotmanipulatorene og dermed minimere avvik i bevegelsene til manipulatorer som skal bevege seg synkront. Synkronisering av roboter har blitt benyttet i situasjoner hvor en enkelt robot ikke er nok til å utføre en oppgave, eksempler på dette er å flytte objekter som er for tunge for en robot, og konstruksjon av avanserte deler. Samarbeid mellom to eller flere roboter gir en mer avansert problemstiling med tanke på kinematiske begrensninger og regulering av dynamikk[YNZ89]. McClamroch[McC86] viser at et robotsystem hvor to roboter griper en felles last kan beskrives ved en enhetlig dynamisk ligning. McClamroch viser at dynamikken til to robotmanipulatorer og bevegelsen til lasten kan beskrives slik

$$\mathbf{M}_1(\mathbf{q}_1)\ddot{\mathbf{q}}_1 + \mathbf{G}_1(\mathbf{q}_1, \dot{\mathbf{q}}_1) = \mathbf{T}_1 + \mathbf{J}_1^{\mathbf{T}}(\mathbf{q}_1)\mathbf{F}_1(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{T}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2, \mathbf{T}_2) \quad (2.1)$$

$$\mathbf{M}_2(\mathbf{q}_2)\ddot{\mathbf{q}}_2 + \mathbf{G}_2(\mathbf{q}_2, \dot{\mathbf{q}}_2) = \mathbf{T}_2 + \mathbf{J}_2^{\mathbf{T}}(\mathbf{q}_2)\mathbf{F}_2(\mathbf{q}_2, \dot{\mathbf{q}}_2, \mathbf{T}_2, \mathbf{q}_2, \dot{\mathbf{q}}_2, \mathbf{T}_2) \quad (2.2)$$

$$m\ddot{\mathbf{p}} = -\mathbf{F}_1(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{T}_1, \mathbf{q}_2, \dot{\mathbf{q}}_{12}, \mathbf{T}_2) - \mathbf{F}_2(\mathbf{q}_1, \dot{\mathbf{q}}_2, \mathbf{T}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2, \mathbf{T}_2) - mg \quad (2.3)$$

Hvor $\mathbf{q}$ er robotens leddvinkler, $\mathbf{F}_1$ og $\mathbf{F}_2$ beskriver kontaktkreftene til robotene, $\mathbf{M}$ er treghetsmatrise, $\mathbf{G}$ beskriver krefter som virker på roboten

utenfra, $\mathbf{T}$ er dreiemoment som input til robotleddene, $\mathbf{p} = \mathbf{H}(\mathbf{q})$ er kinematisk beskrivelse av roboten, og $\mathbf{J}$ er Jacobian matrise av $\mathbf{p}$.

For å minimere bevegelsesfeil mellom robotene i et slikt system beskriver Zheng og Luh[ZL86] en master-slave tilnærming hvor slaveroboten følger bevegelsene til masteren. Med denne tilnærmingen bestemmes bevegelsen til robotene ved at masteren kontrolleres, slaven følger da bevegelsene til masteren.

Bondhus et.al[AKB04] viser et observer-controller system for å synkronisere to robotarmer med kun posisjonsmålinger tilgjengelig. Det benyttes en observer til å estimere hastigheten til robotene og de viser at en tilnærming med Lyapunov teori og et modifisert small-gain teorem gir semi-global uniform ultimate boundedness for observer og synkroniseringsfeil. Med en standard regulator for en master robot modelleres slaveroboten som følger

$$\dot{\mathbf{x}}_1 = \mathbf{x}_2 \tag{2.4}$$

$$\dot{\mathbf{x}}_2 = \mathbf{M}^{-1}(\mathbf{x}_1)\boldsymbol{\tau}_s + \boldsymbol{\beta}(\mathbf{x}_1, \mathbf{x}_2, \boldsymbol{\theta}) \tag{2.5}$$

Med

$$\boldsymbol{\beta}(\mathbf{x}_1, \mathbf{x}_2, \boldsymbol{\theta}) = \mathbf{M}^{-1}(\mathbf{x}_1[-\mathbf{C}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{x}_2 - \mathbf{g}(\mathbf{x}_1)] - \mathbf{M}^{-1}(\mathbf{x}_1)\mathbf{f}(\mathbf{x}_2, \boldsymbol{\theta}) \tag{2.6}$$

Hvor $\boldsymbol{\theta}$ er paramertre i friksjonsmodellen til roboten, $\mathbf{x}$ er en vektor av posisjon og orientering av robotens ledd, $\boldsymbol{\theta}_s$ er kreftene på leddene, $\mathbf{M}$ er treghetsmatrise, $\mathbf{C}$ er sentripetal og coreoliskrefter, $\mathbf{g}$ er gravitasjonskrefter og $\mathbf{f}$ er friksjonskoeffisient. Med dette definert viser Bondhus et.al. at følgende $\boldsymbol{\tau}_s$ oppfyller stabilitetskravene ovenfor.

$$\boldsymbol{\tau}_s = \mathbf{M}_0(\mathbf{f}_s(\mathbf{s}) - \mathbf{g}_2 + \mathbf{g}_4 - \boldsymbol{\Lambda}(\mathbf{g}_1 - \mathbf{g}_3)) \tag{2.7}$$

Hvor s er en koordinattransformasjon og $\mathbf{f}_s(\mathbf{s}) = -\mathbf{A}_s\mathbf{s}$ hvor $\mathbf{A}_s$ er konstand, diagonal og positiv definitt.

2.2 Visuell styring

Denne type kamerakontroll er basert på at kameraposisjonen oppdateres ut i fra om kameraet er plassert på ønsket posisjon i forhold til et element i bildet. Gleicher og Witkin [GW92] beskriver et *through-the-lens* system hvor en bruker kan styre et virtuelt kamera ved å velge elementer sett gjennom kameraet. Gleicher og Witkin kan med denne kontrolleren styre kameraet etter et punkt på skjermen, avstanden mellom to punkt på skjermen,

orienteringen til to punkter i bildet eller forholdet mellom avstander på skjermen. Court og Marchand [MC02] viser en teknikk for kamerakontroll hvor kameraet kontroleres etter elemeter i bildet. Oppgavene som skal løses velges i *2D-rom* og kamerabevegelsene oversettes for å bli utført i et 3D-rom. Systemet til Court og Marchand er også i stand til å reagere på endringer i 3D-verdenen, for eksempel bevegelige objekter.

2.2.1 Sannsynlighetskart

Bauman et.al[BLCL10] viser en metode som finner en bane til robotmanipulatoren som unngår hindringer for robotmanilpulatoren og samtidig holder et bestemt objekt i kamerabildet. Metoden benytter et *Vision Based Probabilistic Road-map*(VBPRM) som bygger på PRM[KSLO96] og vektede begrensninger på de mulige handlingene. Begrensingene som innføres er at målet skal være i synsfeltet og at målet ikke er tildekket av hindringer i arbeidsrommet. Disse begrensningene sjekkes ved hjelp at to algoritmer, *dynamic visibility cheching*(DVC) og *dynamic occlusion checking*(DOC). En PRM definerer bevegelsesrommet til roboten som en graf $\mathbf{G}(\mathbf{N}, \mathbf{E})$[KSLO96] hvor nodene $\mathbf{N}$ er et set konfigurasjoner for roboten og kantene $\mathbf{E}$ er bevegelser mellom to konfigurasjoner som er tilgjengelige for baneplanleggeren. Grafen $\mathbf{G}$ bygges opp ved at tilfeldige konfigurasjoner legges til i $\mathbf{N}$ og det sjekkes om den nye konfigurasjoen kan nås fra noen nærliggende tidligere konfigurasjoner. Dersom det finnes en bane mellom disse konfigurasjonene lagres de i $\mathbf{E}$. For å begrense antallet nabokonfigurasjoner som sjekkes, settes det en maks avstand mellom nodene, og et maks antall nabonoder. Om antallet noder som ble valgt er stort nok vil man ha en ganske uniform dekning av konfigurasjonerommet til roboten. Dersom det er områder i $\mathbf{G}$ som har lite dekning utføres et en utvidelse av grafen. Konfigurasjoner i dette området velges og sjekkes mot nærliggende noder som i konstruksjonsfasen.

En kant i PRM sier at roboten kan bevege seg langs banen uten å kollidere med miljøet, hver kant kan vektes ved ut i fra hvor ønskelig det er at denne banen velges. For å konstruere et VBPRM velges det en rekke konfigurasjoner i konfigurasjonsrommet $\mathbf{Q}$, og om disse er kollisjonsfrie lagres de som $\mathbf{V}$. Deretter kobles nabohjørner sammen med kanter $\mathbf{E}$ dersom kantene er kollisjonsfrie. Hver kant sjekkes for om målet er synlig fra denne posisjonen. Til slutt blir hver kant vektet ut i fra hvor mye av målet som er dekket dersom kameraet beveger seg langs den. Når grafen er ferdigstillt gjøres det et søk ved hjelp av Dijkstras algoritme[CLRS01] for å finne den en bane med god synlighet for målet.

2.2.1.1 Nabonoder

For å velge hvilke noder man kan bevege seg til fra en gitt node velger Kavraki[KSLO96] ut et sett nabonoder N_c. En node c velges ut og det sjekkes om det er mulig å konstruere en bane fra c til de nærliggende nodene i N_c. For å begrense antall nabonoder som sjekkes bestemmes en maks avstand mellom nabonoder.

$$N_c = \tilde{c} \in N \| D(c, \tilde{c}) \leq maxdist \tag{2.8}$$

I tillegg til dette settes det at maks antall nabonoder for å begrense kjøretiden til denne operasjonen.

Avstandsfunksjonen D (lign.2.9) benyttes til å lage og sortere N_c og er definert slik at for et nodepar (n,c) returnerer D sannsynligheten for at det ikke kan konstrueres en bane mellom disse punktene.

$$D(c, n) = \max_{x \in robot} \|x(n) - x(c)\| \tag{2.9}$$

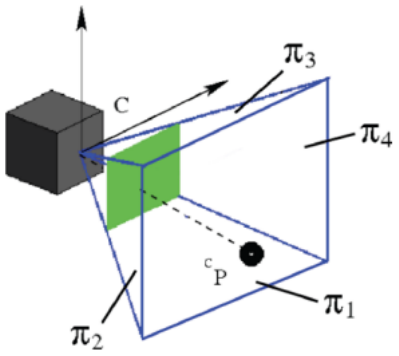
2.2.1.2 Kollisjonssjekk

Algoritmen for kollisjonssjekk i [BLCL10] fungerer ved at man for en bevegelse mellom to konfigurasjoner sjekker minste avstand fra manipulatoren til en hindring og den lengste distansen en del av manipulatoren begever seg mellom konfigurasjonene. Dersom den lengste avstanden et punkt på manipulatoren begever seg er kortere enn minste avstand til en hindring er bevegelsen kollisjonsfri. La $\mathbf{q}_i$ og $\mathbf{q}_f$ være henholsvis start og sluttposisjon for en bevegelse og $d_{DCC}(\mathbf{q}_i)$ og $d_{DCC}(\mathbf{q}_f)$ minste avstand fra manipulatoren i disse punktene og $l_{DCC}(q(t))$, hvor $q(t)$ er en bevegelse, er den lengste distansen et punkt på manipulatoren beveger seg. Manipulatoren vil da ikke kollidere med en hindring dersom

$$l_{DCC}(\mathbf{q}(t)) < d_{DCC}(\mathbf{q}_i) + d_{DCC}(\mathbf{q}_f) \tag{2.10}$$

2.2.1.3 Synlighetssjekk

For sjekk av synligheten til målpunktet benytter Baumann et.al. seg av Dynamic Visibility Checking(DVC)[LCL08]. En synsfeltbegrensning krever at et målpunkt befinner seg innenfor synsfeltet til et kamera montert på robotmanipulatoren. Målpunktet er et punkt $\mathbf{P}^c = [X^c, Y^c, Z^c]$ beskrevet i et kamerakoordinatsystem C som er festet på ytterste ledd på roboten. Synsfeltet til kameraet begrenses av fire plan, $\pi_1, \pi_2, \pi_3, \pi_4$ (fig.2.1). Dersom $\mathbf{P}^c$ kolliderer med et av planene $\pi_1, \pi_2, \pi_3, \pi_4$ har målpunktet beveget seg



Figur 2.1: Kamera med målpunk og plan som bestemmer synsfeltet

utenfor synsfeltet til kameraet. Med utgangspunkt i 2.10 kan det formuleres en løsning på synsfeltproblemet. Korteste distanse fra $\mathbf{P}^c$ til planene $\pi_1, \pi_2, \pi_3, \pi_4$ finnes ved å evaluere

$$d_{DVC} = \min_{1 \leq k \leq 4} \frac{\mathbf{n}_{\pi_k}}{\|\mathbf{n}_{\pi_k}\|} \mathbf{P}^c \tag{2.11}$$

Den siste delen av 2.10 er lengden av banen, i dette tilfelle vil det si bevegelsen $\mathbf{P}^c$ får i $\mathbf{C}$ når robotmanipulatoren beveger seg, dvs.

$$l_{DVC}(\mathbf{q}(t)) = \int_0^1 |\dot{\mathbf{P}}^c(\mathbf{q}(t))|_2 dt \tag{2.12}$$

For å slippe å beregne dette intervallet estimeres $l_{DVC}(\mathbf{q}(t))$ med en øvre grense $O(l_{DVC}(\mathbf{q}(t)))$, dette gir fra [BLCL10]

$$l_{DVC}(\mathbf{q}(t)) \leq \sum_j l_{DVC}(\mathbf{q}_j(t)) \tag{2.13}$$

Gitt 2.13 finner Baumann et.al en konfigurasjon som maksimerer $\mathbf{P}^c$. Kantene i $\mathbf{E}$ vektes ut i fra dette etter hvor lenge målpunktet befinner seg utenfor synsfeltet i løpet av bevegelsen langs kanten.

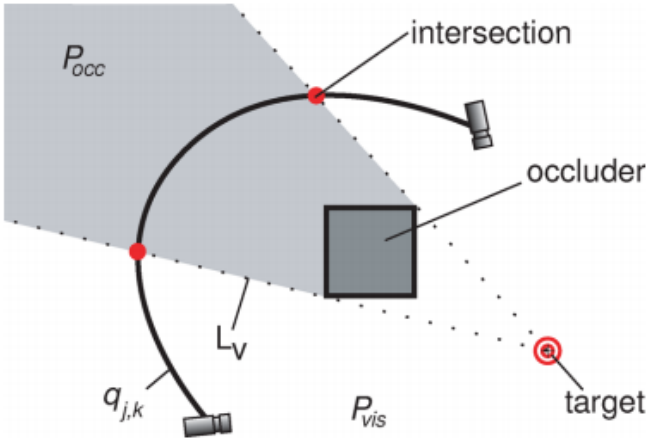
2.2.1.4 Hindringer

En *occlusion constraint* er en begrensning for hindringer i synsfeltet mellom kameraet og målpunktet. Når målpunktet er kjent kan det regnes ut en region i romet hvor målet ikke er synlig. Tarabanis et.al [TTK96] beskriver en metode for å beregne området hvor målet ikke er synlig. Avstanden til

grensen hvor målet ikke er synlig beregnes ved hjelp av

$$d_{DOC}(\mathbf{q}) = \|FK(\mathbf{q}) + \mathbf{P}_{L_v}^{Pmin}\| \tag{2.14}$$

Hvor L_v er grensen mellom synlighet og ikke synlighet av målet, FK($\mathbf{q}$) er kinematisk beregning av kameraets optiske senter i konfigurasjon q og $\mathbf{P}^P min_{L_v}$ er det nærmeste punktet på L_v til FK($\mathbf{q}$). Se Fig 2.2.1.4.



Figur 2.2: Kamera og målpunkt med en hindring i arbeidsrommet.

Tarabanis' metode for å beregne om et målpunkt er synlig fra et kamera definerer en *viewing cone*, synskjegle, som er et sett av alle linjer mellom synspunktet og målet som skal filmes. Synlighetsregionen defineres som alle punkter i det frie romet hvor hele målet er synlig, det hindrede området består av alle punkter hvor målet ikke er synlig. Et synspunkt er dermed i synlighetsregionen til et målpunkt hvis og bare hvis det ligger i det frie romet og synlighetskjeglen til synspunktet ikke kolliderer med noe i romet. Objektene i verdenen antas å være polygoner, og et målobjekt er en side på et mangesidet polygon. Dersom flere sider av målopolygonet er av interesse kan den samlede synlighetsregionen uttrykkes som en union av delregionene.

Et superset L_v av synlighetregionen defineres først som $L_v = L_{v_1} \cup L_{v_2}$ hvor L_{v_1} er deler av grensen til objekter i romet, og L_{v_2} er punkter med synlighetspyramider som er på en tangent til et eller flere objekter. Et superset av L_{v_1} defineres som alle objekter i romet, dette supersettet kan begrenses ved at bare de sidene på objektene som ligger mot målobjektet påvirker synlighetsregionen. L_{v_2} er synspunktene som ligger på en tangent til et eller flere objekter. Dette vil si at et punkt $\mathbf{p}$ i L_{v_2} vil ligge på en linje

qr hvor q er tangentpunktet og p er et hjørne eller et punkt på en kant på målpunktet.

Baumann et.al benytter seg av dette for å bestemme om det er hindringer mellom kameraet og målpunktet. Ved hjelp av d_{DOC}(2.14) beregnes det hvor mye av kameraets synsfelt som ligger innenfor et omdråde hvor målpunktet er skult. Kantene i $\mathbf{E}$ vektes da etter hvor lenge målpunktet er skjult i løpet av bevegelsen lang kantene.

Kapittel 3

Kamerastyring i spillteknologi og virtuelle filmmiljøer

3.1 Kameraperspektiv i spill

I spill er det viktig å gi spilleren et godt bilde av miljøet spillkarakteren befinner seg i. Dette kan innebære å plassere kameraet på en slik måte at spilleren bli ledet mot et bestemt mål, formidle en bestemt stemning eller hjelpe til med å fortelle en historie. Det er i spill brukt forskjellige metoder for å oppnå dette og gi spilleren en best mulig opplevelse.

I de tidlige spillene som *Pong* frem til 2D-platformspill som *Super Mario* var ikke kameraplassering en stor utfordring, perspektivet var statisk og sentrert rundt spillbrettet eller spillkarakteren. I spill med et 3D-miljø er det noen grunnleggende typer kameraplassering som er vanlige. En standard måte å dele inn perspektiver for 3D spill er:

- Førsteperson
- Tredjeperson
- Fugleperspektiv
- Cinematisk

Det enkleste av disse er førstepersonsperspektiv, hvor spilleren ser hva spillkarakteren ser og har full kontroll over kameraføringen. Statiske kameraer plassert på bestemte steder i hver scene er enkleste form for tredjepersonsperspektiv. Denne type kameraplassering begrenser karakterens handlingsrom, men gjør det mulig å ha full kontroll over hva spilleren ser til enhver tid. En vanligere måte å bruke tredjepersonsperspektiv på er å ha et

kamera som følger spillkarakteren. Et fugleperspektiv gir et bilde ovenfra og gir spilleren god oversikt over miljøet, men distanserer samtidig spilleren fra karakteren. Et følgekamera brukes der hvor det er ønskelig å gi et bedre inntrykk av hva spillkarakteren ser. Følgekameraet følger vanligvis spillkarakteren i bevegelsesretningen i en bestemt avstand, problemer med dette oppstår når det er hindringer, for eksempel hvis spillkarakteren rygger mot en vegg. Kameraet kan beveges i disse situasjonene, men å gi spilleren et nytt perspektiv uten å forstyrre spillopplevelsen er en utfordring. Kameraposisjonen endres i noen situasjoner også for å formidle informasjon til spilleren. Dette kan være for å gi bedre oversikt i åpne eller nye områder, gi et mer fremoverrettet blikk i lukkede områder, eller å gi hint om hvor et objekt befinner seg. I dette perspektivet blir kameraet i de fleste situasjoner styrt direkte etter spillerens bevegelser.

3.2 Automatisk kamerakontroll i virtuelle miljøer

Det har også vært gjort endel arbeid rundt virtuelle 3D miljøer i andre områder enn spill. Mye av dette har sentrert rundt virtuell filminnspilling, hvor kameraplassering er viktig for å formidle en historie samtidig som stemning og følelsene til aktørene skal formidles.

For å styre et kamera i et virtuelt miljø introduserer Drucker og Zeltzer[DZ94] et konsept de kaller en *kameramodul* som ligner på en scene i filminnspilling. En scene representerer kameraparametre over en gitt tidsperiode, og styringen av kameraet innen denne tidsperioden. Dette gjør det mulig å få en kontunuerlig bevegelse mellom scener. En kameramodul benyttes for å beskrive oppførselen til kameraet i situasjoner som på forhånd er definert av brukeren. Kameramodulen inneholder en rekke begrensninger for kameraet, en regulator, lokal tilstand og en initialiserer. Begrensningene må oppfylles når modulen er aktiv og kan beskrive kameraets posisjon, orientering og bane. Regulatoren styrer kameraet på bakrunn av input fra brukeren og disse begrensningene.

He et.al.[HCS96] utvider kameramodulkonseptet i et virtuelt filminnspillingsmiljø. For å bedre beskrive hvilke begrensninger som må følges, og hvordan en scene skal se ut innfører He et.al. idiomer som beskriver hver type scene. Hvert idiom bestemmer hordan scener skal se ut og transisjoner mellom scener. Idiomene benytter seg av en kameramodul for å plassere kameraet i det virtuelle miljøet. For å beskrive kameraplasseringen implementerer kameramodulene en *line of interest*, som er en vektor mellom to

aktører som skal være med i scenen, orientert i bevegelsesretningen til en aktør eller i retningen en aktør er orientert. Kameramodulene beskriver også hvilket utsnitt som ønskes i en scene. Idiomer benyttes for å sette sammen scener og er organisert hierarkisk, slik at generelle idiomer kan gi kontrollen til med spesifike idiomer i gitte situasjoner.

Tomlinson et.al[TBN00] innfører et *CameraCreature* i det virtuelle filminnspillingsmiljøet for å oppnå kontinuitet i scenene og mulighet for større interaktivitet i kameraføringen enn det idiomene til He et.al er i stand til. Hvordan en tagning skal utføres bestemmes ut i fra hvilken motivasjon og hvilke følelser en scene skal formidle. En tagning er her det viktigste elementet i filmskapningen og inneholder informasjon om aktører og bevegelsesmønster for kameraet. CameraCreature benyttes som en aktør bak kameraet som er i stand til å innhente informasjon om aktørene foran kamera. CameraCreature kan dermed med informasjon om handlingene og motivasjonen til aktørene, og hva som skal formidles i en tagning, bestemme kamerabevegelsene. For transisjoner mellom tagninger benytter Tomlinson seg av *whip-pan* som gjør at kameraet raskt beveger seg til en ny posisjon for en ny tagning. I motsetning til et kutt sørger whip-pan for en kontinuerlig scene uten hopp i kameraposisjon.

3.3 Synlighet av fokuspunkt

Samtidig som det er viktig at de riktige elementene er i bildet, må man unngå at det er hindringer mellom kameraet og målet.

Dette er et komplekst problem som det er blitt foreslått flere løsninger på. Disse metodene strekker seg fra *Ray-Casting*[Rot82] som fungerer ved at en stråle kastes fra kameraet mot målet, på denne måten kan man finne ut hva som er direkte foran kameraet. For å få et bedre bilde av hva som befinnner seg foran kameraet gjentas dette søket rundt kamera-aksen. Ulemper med denne metoden er at det er vanskelig å dekke hele synsfeltet til kameraet ettersom hver stråle kun et lite område.

Bounding volumes er en metode presentert av Marchand og Courty [MC02] som består i at et volum omslutter kameraet, fokuspunktet og området direkte i mellom disse. Ved å hindre andre objekter i å bevege seg inn i dette volumet opprettholder man fri sikt.

Ved å benytte seg av hardware rendering teknikker er det mulig å estimere synligheten til målet på en mer nøyaktig måte [CO09]. Dette brukes ved at man rendrer målet og hindringen, og så bruker for eksempel pixel shadere til å måle om de overlapper. En Pixel Shader er en hardwarefunksjon implementert i grafikkprosessorer.

Del III

Design

Kapittel 4

Styring

4.1 Kontekstbasert Styring

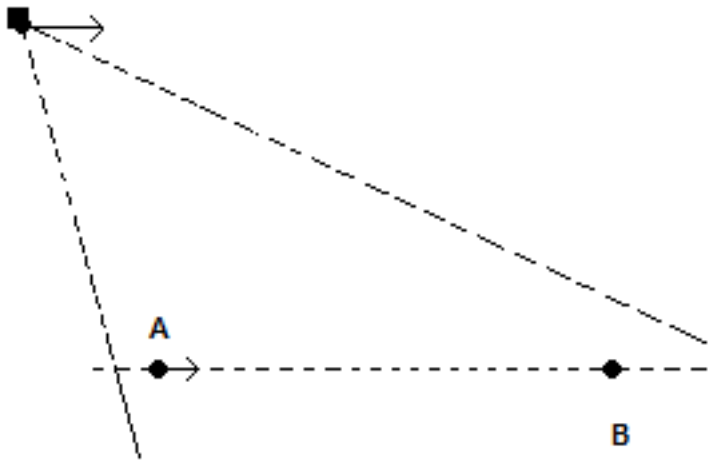
Kontekstbasert kamerastyring vil si at kameraet skal kunne oppføre seg forskjellig ut i fra hvilke operasjoner som utføres og hvor i utførelsen av en operasjon man er. Med automatisk kamerastyring vil operatøren avlastes og kan fokusere på oppgaven som utføres. Det er spesielt viktig når en robot opererer nær sensitivt utstyr at operatøren til enhver tid vet hva som foregår slik at ulykker kan unngås. I robotlaben er det under en operasjon spesielt to utfordringer for kameraplassering. Når robotarmen nærmer seg utstyret den skal interaktere med, og når robotarmen kommer i veien for kameraet.

For å oppnå en oversiktlig og robust metode for kontekstbasert styring vil det være mulig å basere dette systemet på idiomer og kameramoduler fra He[HCS96]. Idiomene vil inneholde informasjon nødvendig for å hvilken type tagning som skal gjøres, og når transisjonene mellom disse skal gjøres. Dette bestemmes ut i fra hvilken operasjon som skal utføres og hvor i utførelsen man befinner seg. Kameramodulene vil utføre kameraplasseringen i de forskjellige situasjonene.

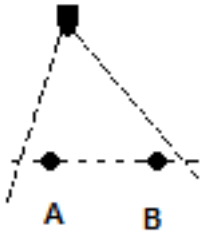
Kameramoduler som kan brukes kan baseres på hva He viser, men noe modifisert blant annet for å unngå hopp i kameraposisjon.

- *Follow* er en hybrid av *external* og *track* fra He. Denne kameramodulen vil følge robotarmen og holde et vidt perspektiv så deler av robotens arm og produksjonsutstyret er synlig, Figur 4.1.

- *Close* vil bevege kameraet fra følgeposisjonen og gi et nærbilde av produksjonsutstyret og robotverktøyet, Figur 4.2.



Figur 4.1: Kamera følger robotarm(A) og holder prosessutstyr(B) i fokus



Figur 4.2: Kamera holder nært fokus på robotarm(A) og prosessutstyr(B)

4.2 Transisjoner

For å bestemme når tranisjoner mellom visninger skal gjøres vil det kunne benyttes idiomer. Et idiom vil assosieres med hver operasjon i robotlaben, og idiomene bestemmer når transisjoner mellom visninger skal skje. Et idiom vil bestemme dette ut i fra begrensninger som settes på forhånd. He implementerer et idiom som en tilstandsmaskin hvor hver tilstand tilsvarer en tagning, hver tilstand inneholder betingelser som fører til en endring til en annen tilstand.

Kapittel 5

Autonom Kamerakontroll

For autonom kamerakontroll er det valgt et styresystem basert på *vision based probabilistic road map*[BLCL10]. Denne metoden for baneplanlegging vil over en rekke steg konstruere et kart over arbeidsområdet til roboten med mulige baner og velge en god bane fra de mulige som er funnet.

5.1 Vision Based Probabilistic Roadmap

Veikartet som skal konstrueres for roboten er en graf $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ hvor nodene $\mathbf{V}$ er konfigurasjoner for det virtuelle kameraet, og $\mathbf{E}$ er kanter (a,b) som beskriver en bane mellom nodene a og b. Man starter med en tom graf $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ og velger en ny konfigurasjon $\mathbf{N}$ i arbeidsrommet til roboten og legger denne til i $\mathbf{V}$. Når $\mathbf{V}$ er fulstendig fylt av gyldige konfigurasjoner vil man iterere gjennom alle nodene og en rekke nærliggende noder til hver $\mathbf{V}$ velges og det vil gjøres en sjekk for om det kan konstrueres en bane mellom $\mathbf{V}$ og nabonodene. Dersom det lar seg gjøre å konstruere en bane mellom en node c og den valgte nabonoden n blir kanten (c,n) lagt til i $\mathbf{E}$. For å begrense antall noder n som blir forsøkt koblet sammen med c velges det kun et bestemt antall noder som ligger nær c. Gitt at N_c beskriver nabonoder for hver node V, og *kollisjon* sier om kameraet kolliderer med noe i denne konfigurasjonen viser følgende pseudokode hvordan VBPRM initialiseres.

```
V=Ø
E=Ø
K=Ø
N_c=Ø
loop
        c=konfigurasjon
```

```
        if not kollisjon:
                V=V+c
for n in V:
                for c in V.N_c:
                        if bane fra c til n
                                E=E+(c,n)
```

Det benyttes i denne oppgaven et virtuelt flytende kamera og det er derfor mulig å lagre konfigurasjonene for kameraet i arbeidsromkoordinater i steden for robotens konfigurasjonsrom. Dette gjør oppgaven med å finne konfigurasjoner enklere og det forenkler implementasjoen betydelig når man ikke er avhengig av en gyldig robotkonfigurasjon for plassering av kameraet. For å konstruere et Probabilistic Road Map (PRM) foreslås det å sample konfigurasjoner i absolutte koordinater. Det er ikke ønskelig med en kameraplassering for langt unna arbeidsroboten, så det brukbare arbeidsromet til kameraet kan begrenses til et område innen en viss avstand fra bevegelsesområdet til robotarmen. Kameraets bevegelsesrom kan også begrenses til å ikke inneholde området på baksiden av prosessutstyret.

Et PRM defineres som en representasjon av arbeidsrommet og bygges opp ved at det opprettes en rettet graf $\mathbf{G}(\mathbf{V}, \mathbf{E})$. Hjørnene $\mathbf{V}$ er her konfigurasjonene til det virtuelle kameraet i romkoordinater som benyttes av baneplanleggeren som veipunkter for kamerabevegelsen. Kantene i grafen, $\mathbf{E}$, er baner mellom konfigurasjoner som er tilgjengelige for baneplanleggeren. Dersom det finnes en kolisjonsfri bane mellom $\mathbf{q}_k$ og $\mathbf{q}_j$ er $e_{j,k} \in E$. En lokal planlegger benyttes for å sjekke om det finnes en bane mellom konfigurasjonene, for at den lokale planleggeren skal være så rask som mulig vil den kun sjekke enkle baner.

Utvelgelsen av konfigurasjoner foreslås gjort ved at det lages et rutenett innenfor arbeidsrobotens arbeidsområde. Når en ny konfigurasjon $\mathbf{q}_k$ velges vil denne legges til i V og når grafen er fyllt opp av noder gjøres det en sjekk for om at nodene ikke kolliderer med det miljøet. For en operasjon i robotlaben vil hvert punkt i arbeidsrommet sjekkes for hver posisjon roboten vil ha i løpet av utførelsen av operasjonen. Statusen for kollisjon mellom en konfigurasjon og robotarmen eller prosessutstyret i et gitt tidspukt lagres i $\mathbf{V}$. Dersom det finnes en bane mellom q_k og en valgt nabonode q_j vil denne kanten (q_j, q_k) legges til i $\mathbf{E}$. Kollisjonssjekkingen i opprettelsen av G foreslås gjort ved hjelp av innebygde kollisjonssjekkingsrutiner i ODE, på denne måten vil man lett kunne oppdage kollisjoner mellom objekter selv om man kun trenger å forholde seg til en punktplassering av objektet.

5.1.1 Nabonoder

Valget av nabonoder N_c til c begrenses ved at det kun velges et visst antall noder som ligger nær c. Denne begrensningen er satt til å være de 124 nærmeste nodene, det vil si at man vil velge nabonoder i et $5 * 5 * 5$ rutenett rundt noden man befinner seg i. Dette er valgt for å gi kameraet stor bevegelsesfrihet og det gir også mulighet til å bevege seg presist samtidig som det gir mulighet for lengre bevegelse over færre steg.

5.2 Kollisjonssjekk

For å sjekke om det er en kollisjon mellom det virtuelle kameraet og arbeidsroboten eller prosessutstyret vil det før en operasjon utføres gjøres en sjekk for hver konfigurasjon. Kollisjonsjekkingen utføres ved at hver konfigurasjon for det virtuelle kameraet sjekkes for hver posisjon arbeidsroboten kan ha i løpet av en operasjon. Statusen for hver node i hvert tidsskritt vil lagres i $\mathbf{V}$.

5.3 Synlighet av målpunkt

Målpunktet er et punkt $P^r = [X^r, Y^r, Z^r]$ i absolutte koordinater og befinner seg ytterst på armen til arbeidsroboten. Synsfeltet til kameraet kan bestemmes av synsvinkelen til kameraet. Dersom vinkelen mellom aksen til kameraet og målpunktet er mindre enn synsvinkelen til kameraet befinner målpunktet seg innenfor synsfeltet. Dette kan beregnes ved prikkproduktet av en normalisert vektor for kameraaksen, og en vektor fra kameraets origo til målpunktet.

$$\begin{aligned} \hat{\psi} &= \frac{\psi}{\|\psi\|} \\ \hat{\phi} &= \frac{\phi}{\|\phi\|} \\ \theta &= \arccos \hat{\psi}\hat{\phi} \end{aligned} \tag{5.1}$$

Når VBPRM opprettes vil også kun noder som befinner seg innenfor et visst område rundt arbeidsverktøyet til arbeidsroboten settes som brukbare noder. For hvert steg arbeidsroboten tar i utførelsen av en operasjon vil de noder som befinner seg for langt unna arbeidsverktøyet eller i en posisjon hvor arbeidsroboten befinner seg mellom kameraet og arbeidsverktøyet lagres som noder hvor en det er en kollisjon.

Del IV

Implementasjon

Kapittel 6

Implementasjon

For implementasjon ble det valgt å benytte C++ og utviklermiljøet Microsoft Visual Studio. For å lage et miljø for å teste en robotmanipulator og et kamera ble Open Dynamics Engine(ODE) valgt som simuleringsverktøy. ODE er et kraftig verktøy for simulering av stive legmers dynamikk og er platformuavhengig med et C/C++ grensesnitt. For å få en grafisk fremstilling av simuleringen ble Irrlicht valgt som 3D motor. Irrlicht er en avansert motor for å lage 3D miljøer og kan brukes sammen med ODE for å vise hvordan simuleringene oppfører seg. Begge disse verktøyene er fri programvare distribuert under lisenser som gjør at de kan benyttes fritt og vidredistribueres enten som kildekode eller binærfiler.

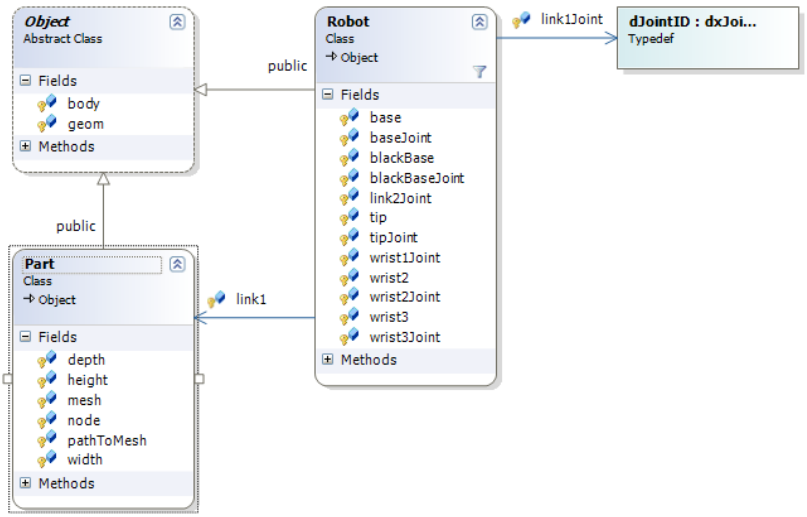
Simuleringsmiljøet i ODE består av en dynamikkverden, *dWorld*, som inneholder alt som inngår i simuleringen. Et eller flere kollisjonsrom, *dSpace*, hvor objekter som skal interaktere befinner seg. Objektene består av en *dBody* som er en representasjon av et stivt legeme, og en *dGeom* som bestemmer geometrien til objektet.

For å gjøre konfigureringen av robotmanipulatoren fleksibel er en robotarm implementert som en klasse *Robot* som inneholder alle deler og ledd robotarmen består av, Figur 6.1. Alle delene til robotarmen er instanser av en klasse *Part* som inneholder dimensjonen til hver del og informasjon om hvordan delen skal se ut i Irrlicht.

Hver del av roboten opprettes sammen med et ledd som skal holde den sammen med den delen som er nærmere basen til roboten. Når alle robotdelene er opprettet, plasseres delene på rett sted i forhold til det forrige leddet og de roteres før et ledd binder det nye leddet sammen med det foregående leddet.

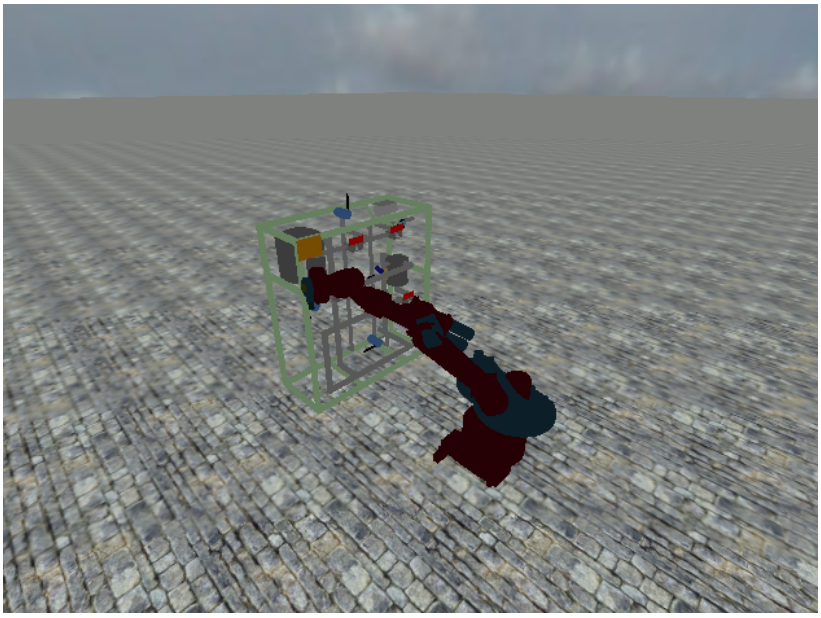
Når roboten er opprettet i ODE blir en Irrlicht *Viewer* startet og roboten lagt til i 3D-miljøet, Figur 6.2.

For hvert steg i simuleringen regner ODE ut nye posisjoner for alle ledd



Figur 6.1: Klasser i robotimplementasjonen

i roboten, og sjekker om det er en kollisjon mellom objekter.



Figur 6.2: Robotarm og Prosessutstyr

6.1 Way point styring av robotarm

For å generere en bane for robotarmen er det valgt en way point modell hvor arbeidspunktet til robotarmen flyttes lineært mellom waypunktene. Det er valgt en standard utgangsposisjon hvor alle operasjoner starter, dette gjør det enkelt å legge inn nye punkter robotarmen skal bevege seg til. Når robotarmen beveger seg mellom to waypunkter regnes det ut hvor langt man er i fra punktet man er på vei til. Deretter beveges robotarmen i en fast steglengde mot dette punktet. Når avstanden til målwaypunktet er under en fastsatt grense fortsetter robotarmen mot neste punkt. For hvert steg roboten tar oppdateres simuleringen i ODE og det grafiske bildet oppdateres. Preudokode av denne bevegelsen er som følger

```
void Robot::moveToPos(endPos, viewer){
 //viewer er en instans av 3D miljøet
 dReal newPos
 dReal *position;
 dReal positionError=0
 dReal largestError=0
 while(viewer oppdateres){
  position=posisjon til arbeidspunktet
  positionError=endPos-position
  largestError=verdien til den største feilen mellom faktisk
               og ønsket posision i x,y eller z retning
  if positionError i x,y, eller z retning:
    newPos = position+abs(positionError)*moveStep
  moveArm(newPos)        //Flytter robotarmen til neste punkt
  moveCamera(nextPosition)
  setCameraViewAngle()
  oppdater ODE simuleringen
  if largestError<grenseverdi: break
  }
}
```

Når robotarmen når frem til det siste waypunktet blir den stående der til den får en ny bane å forholde seg til. Mellom to operasjoner beveges roboten tilbake til utgangsposisjonen slik at waypunkter for kjente operasjoner kan forhåndsdefineres. Utgangsposisjon og posisjon for en operasjon som skal utføres skrives til en tekstfil. Denne tekstfilen leses inn av programmet og robotarmen vil bevege seg først til ugangsposisjonen før den utfører en operasjon. Mellom utgangsposisjonen og punktet hvor en operasjon skal utføres opprettes det noen waypunkter. Disse waypunktene opprettes for at robot-

armen skal få en naturlig og riktig bane inn mot punktet hvor operasjonen skal utføres. Et naturlig waypunkt vil være et punkt rett utenfor der operasjonspunktet er, slik at arbeidsverktøyet føres rett inn mot dette punktet. Det er valgt å legge dette waypunktet i 20cm avstand fra prosessutstyret.

6.2 Probabilistic Road Map

Veikartgrafen G bygges opp ved at en rekke punkter velges i et rutenett innenfor arbeidsområdet til kameraet. Dette betyr at man får en fast lengde mellom punktene og man er sikret at arbeidsområdet er jevt dekket med mulige kameraplasseringer. Et PRM er opprettet som en klasse hvor en struct holder informasjon om plassering og nabonoder for hver konfigurasjon.

```
struct prm{
        konfigurasjon V;
        nabonoder E;
        kollisjon K;
        }G;
```

I implementasjonen er G opprettet som en liste med konfigurasjoner. Her er V en posisjon i arbeidsrommet, E en liste over nabonoder, og K en liste over for hvilke tidsskritt denne konfigurasjonen er gyldig. Når en instans av klassen Vbprm opprettes vil det lages en graf G som utfyller arbeidsrommet med en viss steglengde mellom konfigurasjonene. For å begrense minnebruk er den nødvendig å ha et begrenset antall konfigurasjoner og dermed er det heller ikke mulig med veldig korte avstander mellom konfigurasjoner. Avstanden mellom konfigurasjonene til kameraet er valgt til ligge nær steglengden til robotarmen, dette for å gjøre det mulig for kameraet å følge roboten med en jevn bevegelse. Avstanden mellom konfigurasjoner er valgt til å være 5cm, og arbeidsrommet til det virtuelle kameraet er satt til å være fra robotens base og 2.6m i x-retning, 3m til hver side i y-retning, og fra gulvet og 3m opp i z-retning.

Når det velges nabonoder gjøres dette ved at det itereres gjennom G og de 125 nærmeste nodene legges til som nabonoder. Denne operasjonen gjøres når grafen er fyllt opp av konfigurasjoner for å sikre at det er mulig å bevege seg i alle retninger i grafen. Utvelgelsen av nabonoder foregår ved at man for enhver gyldig konfigurasjon sjekker de nodene som ligger nærmest i et $5 * 5 * 5$ rutenett rundt den aktive noden.

6.2.1 Kollisjonssjekk

Før en operasjon utføres gjøres det en sjekk av hver konfigurasjon for om det er en kollisjon mellom det virtuelle kameraet og robotarmen eller arbeidsverktøyet for hver posisjon robotarmen vil ha under operasjonen. Dette gjøres på forhånd for at det skal kunne beregnes en fullstendig bane for det virtuelle kameraet før den virkelige operasjonen starter. Kollisjonssjekken

vil samtidig begrense hvilke noder som er tilgjengelige for baneplanleggeren til det virtuelle kameraet. Kun noder som befinner seg innenfor en forhåndsbestemt avstand fra arbeidsverktøyet og samtidig ikke kolliderer med noe vil merkes som brukbare noder i hvert tidsskritt.

6.3 Kamera

Det virtuelle kameraet i simuleringen er implementert som en klasse *Camera* i ODE. Denne klassen inneholder informasjon om

- Størrelse
- Posisjon
- Rotasjon
- Synsvinkel
- Målpunkt
- Irrlicht Mesh

Kameraet er uavhengig av andre objekter i simuleringen og kan dermed beveges fritt. Det benyttes i simuleringen også et oversiktskamera som kan styres fritt og som ikke tilhører noe objekt i simuleringen. Dette oversiktskameraet invirker ikke på oppførselen til robotarmen eller det virtuelle flyvende kameraet.

Koblingen mellom kameraobjektet i ODE og Irrlichtobjektet gjør det mulig å koble disse sammen slik at når simuleringen i ODE oppdateres vil det synlige kameraobjektet bevege seg likt med det simulerte objektet. For hvert steg i simuleringen tegner Irrlich bildet på nytt fra hvert kamera. Dette skjer ved at et kamera settes aktivt og bildet fra dette oppdateres, deretter vil det andre kmaraet aktiveres og bildet fra dette blir oppdatert. Kameraklassen har funksjoner for å

- Sette synsvinkel
- Hente synsvinkel
- Sette målpunkt
- Hente målpunkt
- Sette kameraposisjon

- Hente kameraposisjon
- Bevege kameraet til en bestemt node

For å beregne synsvinkelen brukes det en vektor til det nye ønskede målpunktet. En rotasjonsmatrise utregnes og kameraobjektet roteres til den nye synsvinkelen.

$$b = Ra$$

Hvor b er den nye ønskede synsretningen, R er en rotasjonsmatrise og a er den forrige synsvinkelen. Matrisamultiplikasjon og omregning fra synsvektor til en rotasjonsmatrise som ODE ønsker foretas ved hjelp av innebygde funsjoner i ODE. For å forenkle implementasjonen ble det valgt å benytte seg av en funksjon i *irrlicht* for å sette synsvinkelen til det virtuelle kameraet. ODE er ment benyttet ved at man setter krefter som virker på objekter, eller motorer i ledd mellom objekter. Dette gjør det vanskelig å rotere objekter til absolutte vinkler i simuleringen ettersom det ikke er meningen at ODE skal benyttes på denne måten.

6.3.1 Kamerabane

Banen til det virtuelle kameraet opprettes ved at det for hvert steg arbeidsroboten tar sjekkes om posisjonen kameraet befinner seg i vil være gyldig for den neste tidsskrittet. Banen er implementert som en liste av noder. Dersom kameraet befinner seg i en posisjon som er gyldig også i neste tidsskritt vil kameraets posisjon for det neste tidsskrittet forbli uendret. Derimot hvis kameraet ikke kan stå i den samme posisjonen i det neste tidsskrittet vil det velges en nærliggende node som ikke forårsaker en kollisjon. Denne utvelgelsen foretas ved at man iterer over nærliggende noder og sjekker om disse nodene er gyldige for neste tidsskritt. Når en gyldig node er funnet innenfor en maksimumsavstand på 15cm fra der kameraet befinner seg vil denne noden legges til som neste posisjon i banen til kameraet.

Del V

Test og diskusjon

Kapittel 7

Resultat og diskusjon

7.1 Tester

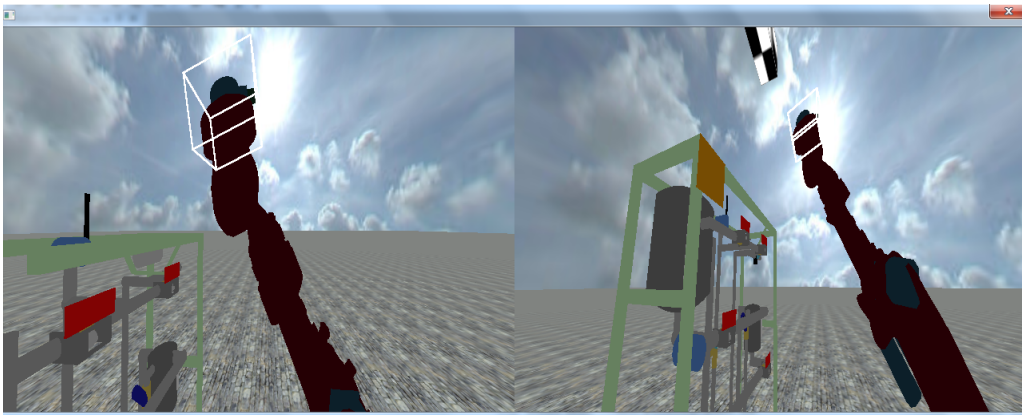
For å teste om det virtuelle kameraet oppfører seg som ønsket er det laget tester for forskjellige parametre under utførelsen av operasjoner. Parametrene som er relevante for oppførselen til det virtuelle kameraet er

- Vinkelen kameraet har i fohold til arbeidsverktøyet til roboten
- Avstand fra det virtuelle kameraet til arbeidsverktøyet
- Om arbeidsverktøyet befinner seg innenfor synsfeltet til det virtuelle kameraet

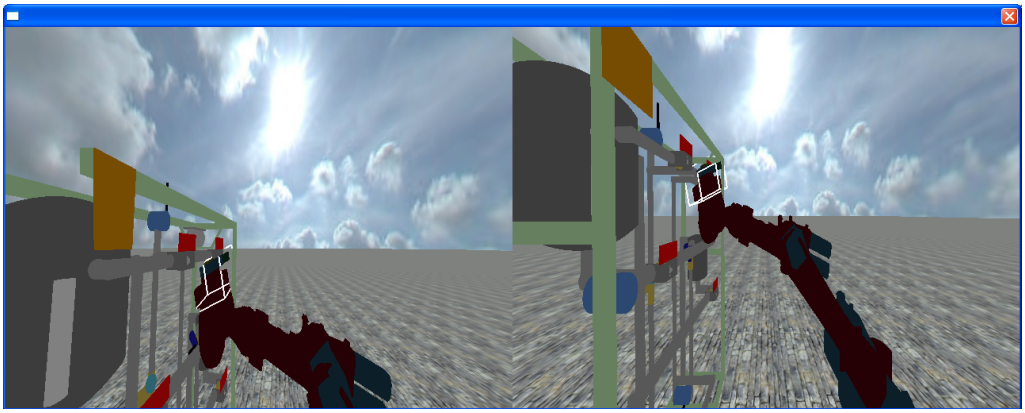
Vinkelen mellom det virtuelle kameraet og arbeidsverktøyet defineres som vinkelen mellom aksen arbeidsverkøyet skal ha når en operasjon skal utføres og aksen i synsretningen til det virtuelle kameraet. Denne vinkelen må være innenfor en viss grense for at man kan være sikker på at robotarmen ikke er en hindring mellom det virtuelle kameraet og arbeidsverktøyet til roboten.

Avstanden mellom det virtuelle kameraet og roboten bestemmer bildeutsnittet kameraet fanger opp og bør være innenfor fastsatte grenser for å gi et tilfredsstillende bilde til operatøren.

For å beregne avstanden kameraet har fra arbeidsverktøyet, og vinkelen mellom de, under en operasjon oppdateres variable for minimum-, maksimum- og gjennomsnitts-avstand og vinkel for hvert steg kameraet og roboten tar under utførelsen av en operasjon.



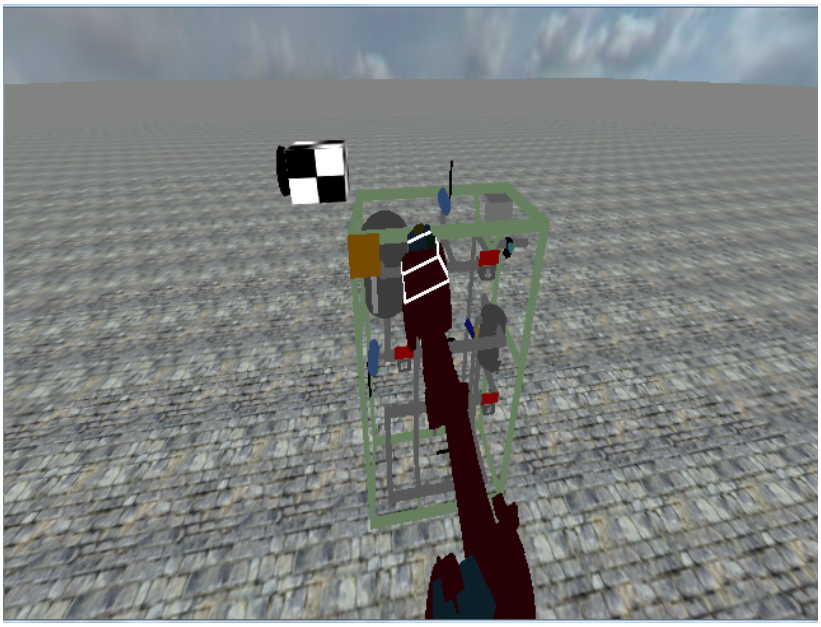
Figur 7.1: Robotarmen og kameraets posisjon ved operasjonenes start



Figur 7.2: Robotarmen og kamerasts posisjon ved operasjonens slutt

7.2 Resultat

For å teste oppførselen til det autonome flyvende kameraet benyttes det baner for robotarmen som genereres ut i fra en start- og slutt-posisjon. Disse banene er laget for å vise følgeegenskapene til det flyvende kameraet og vil ikke være optimale for å utføre operasjoner i robotlaben. Applikasjonen som viser oppførselen til det autonome kameraet består av to synsfelt. Det ene synsfeltet tilhører det autonome kameraet og vil til enhver tid vise hva dette kameraet ser. Det andre synsfelter tilhører det fritt bevegelige testkameraet og er ikke en del av simuleringen. Figur 7.1 og figur 7.2 viser posisjonen til robotarmen og det autonome kameraet ved henholdsvis begynnelsen og slutten til en operasjon. Det autonome kameraet følger roboarmen og holder arbeidsverktøyet i fokus under operasjonen.



Figur 7.3: Robotarm med flyvende kamera

Min Avstand	Maks avstand	Snitt avstand
0.47m	2.31m	0.99m
1.3m	2.26m	1.66m

Tabell 7.1: Avstand under operasjon

Figur 7.3 viser det flyvende kameraet ved siden av robotarmen når en operasjon utføres. Det flyvende kameraet er i simuleringen kun representert ved et svart- og hvit-rutete objekt.

Avstand fra det virtuelle kamereat til arbeidsverktøyet til roboten under to forskjellige operasjoner er vist i tabell 7.1

Disse avstandene viser at kameraet for det meste holder en foruftig avstand til arbeidsverktøyet, men at det er litt avhengig av hvilken bane arbeidsroboten beveger seg i. Ut i fra kamerabildet ser man at kameraet starter i en posisjon i noe avstand fra arbeidsverktøyet før det beveger seg nærmere og følger helt til operasjonen nærmer seg slutten. Når arbeidsverktøyet er nært prosessutsyret på slutten av operasjonen beveger kameraet seg noe lenger bort fra arbeidsverktøyet. Maksavstanden kameraet holder til arbeidsverktøyet overstiger grensen satt tildligere, men når operasjonen observeres utført er ikke dette til hindring for å se arbeidsverktøyet eller operasjonen som utføres. Under det meste av operasjonen holder kameraet

Min vinkel	Maks vinkel	Snitt vinkel
121°	155°	138°
146°	163°	158°

Tabell 7.2: Vinkel under operasjon

seg innenfor grensen, og den overstiger den kun mot slutten av operasjonen når kameraet nærmer seg prosessutstyret.

Vinkelen mellom kameraets synsvinkel og retningen arbeidsverktøyet har når operasjonene utføres ble regnet ut og resultat for minimum-, maksimum- og gjennomsnittsvinkel under operasjonene er listet opp i tabell 7.2

Ut i fra disse vinklene kan man se at kameraet under hele operasjonen befinner seg i en posisjon slik at synsvinkelen er rettet mot arbeidsverktøyet og prosessutstyret. Maksvinkelen er imidlertid ikke så stor at kameraet på noe tidspunkt befinner seg bak arbeidsroboten i forhold til arbeidsverktøyet. Dette er forventet ettersom de mulige posisjonene til kameraet ble begrenset til å kun gjelde posisjoner som vil gi et godt kamerabilde under opprettelsen av kamerabanen. Under den siste operasjonen nærmer kameravinkelen seg en uakseptabel grense, men den overstiger ikke denne.

Oppbygningen av VBPRM før en operasjon utføres er svært tidkrevende. For operasjonene testet her tok utregningen av all nødvendig data 20sekunder på en Intel Core 2 Duo 2.4GHz prosessor. For en operasjon som skal utføres i sanntid er dette ikke optimalt, da en operatør ikke vil ønske å vente på dette for hver operasjon som utføres.

Kapittel 8

Konklusjon

Denne oppgaven har tatt for seg metoder for kamerastyring og implementasjon av et autonomt virtuelt kamera i en simulator. Målet for det virtuelle kameraet var å følge en arbeidsrobot som utfører en operasjon og vise et godt bilde av roboten mens den utfører operasjonen. En løsning basert på VBPRM hvor mulige kameraposisjoner beregnes og evalueres ut i fra hvor godt de oppfyller kravene stillt til kamerabildet, er beskrevet og implementert. Denne løsningen gir gode muligheter for å bestemme begrensninger for kameraplassering og hva som skal være i kamerabildet. Resultatene viser at denne metoden oppfyller kravene for det virtuelle kameraet med tanke på plassering og synsfeltet til kameraet. For å oppnå en god bane er det nødvendig å vite mye om omgivelsene til det virtuelle kameraet. Dette ga store utfordringer under implementeringen av simulatoren. Utregningen av VBPRM er svært tidkrevende og krever mye minne, dette la begrensninger på hvor mye av den opprinnelige metoden det var fornuftig å implementere. Begrensninger for hvor det virtuelle kameraet kan befinne seg i forhold til arbeidsroboten ble derfor bestemt ut i fra hva vi vet om konfigurasjonen i robotlaben og ikke beregnet ut i fra hvilke områder som i virkeligheten ligger i en skygge mellom arbeidsverktøyet og kameraet. Dette hadde ingen annen negativ innvirking på oppførselen til det autonome kameraet enn at det brukbare arbeidsrommet ble noe mer begrenset, da det virtuelle kameraet ikke på noe tidspunkt beveget slik at det var en hindring mellom arbeidsvertøyet og kameraet. Utregningen av VBPRM tar også for lang tid i forhold til hva som er fornuftig i en reell brukssituasjon. Ettersom det ofte skal utføres faste operasjoner kan konfigurasjoner for operasjonene beregnes på forhånd og lastes inn når en operasjon skal utføres.

Kapittel 9

Videre Arbeid

For videre arbeid foreslås det å utvide implementasjonen av VBPRM slik at denne tar bedre hensyn til den faktiske plasseringen av robotmanipulatoren og hvilke områder i arbedsrommet robotmanipulatoren vil være i veien for det autonome kameraet. Dette vil gjøre bevegelsesrommet til det autonome kameraet større, man vil unngå å utelukke kameraplasseringer som nå er utelukket fordi man må forholde seg til posisjoner hvor man er helt sikker på at arbeidsroboten ikke forstyrrer kamerabildet. Det vil også være ønskelig å bedre bestemme bildeutsnittet og dermed også avstanden det virtuelle kameraet holder til arbeidsroboten. Dette vil kunne implementerees som vektede begrensninger i VBPRM. Metoder for å effektivisere beregningen av VBPRM bør undersøkes. Dette vil kunne gjøre det unødvendig å lagre konfigurasjoner for kjente operasjoner, og også gjøre det mulig å benytte seg av VBPRM ved operasjoner som ikke er forhåndsdefinerte.

For å teste metoden i robotlaben er det nødvendig med et grensesnitt mot robotlaben og at kameraet er festet til en arbeidsrobot. VBPRM kan benyttes ved at robotarmkonfigurasjoner lagres i steden for absolutte koordinater for kameraet. Man vil dermet også måtte forsikre seg om at de to arbeidsrobotene ikke kolliderer. For å lage en kamerabane i robotlaben vil det være fornuftig å hente inn de virkelige posisjonene til arbeidsroboten og ikke basere seg på simulerte posisjner for denne.

Bibliografi

[AKB04] Henk Nijmeijer Anne Karin Bondhus, Kristin Y. Pettersen. Master-slave synchronization of robot manipulators. In *Nonlinear Control Systems 2004 IPV - IFAC Proceedings Volume Series*, 2004.

[BLCL10] M. Baumann, S. LÃ©onard, E. A. Croft, and J. J. Little. Path planning for improved visibility using a probabilistic road map. *Robotics, IEEE Transactions on*, 26(1):195–200, Feb. 2010.

[CLRS01] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms, second edition*. Cambridge, Mass. : MIT Press, 3 edition, 2001.

[CO09] Marc Christie and Patrick Olivier. Camera control in computer graphics: models, techniques and applications. In *SIGGRAPH ASIA 09: ACM SIGGRAPH ASIA 2009 Courses*, pages 1–197, New York, NY, USA, 2009. ACM.

[DZ94] Steven M. Drucker and David Zeltzer. Intelligent camera control in a virtual environment. In *In Proceedings of Graphics Interface 94*, pages 190–199, 1994.

[GW92] Michael Gleicher and Andrew Witkin. Through-the-lens camera control. In *SIGGRAPH 92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 331–340, New York, NY, USA, 1992. ACM.

[HCS96] Li-wei He, Michael F. Cohen, and David H. Salesin. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 217–224, New York, NY, USA, 1996. ACM.

[KSLO96] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566 –580, aug 1996.

[Kyr08] Erik Kyrkjebø. Beskrivelse av robotlaben. Web, 2008.

[LCL08] S. Leonard, E.A. Croft, and J.J. Little. Dynamic visibility checking for vision-based motion planning. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2283 –2288, 19-23 2008.

[MC02] E. Marchand and N. Courty. Controlling a camera in a virtual environment. *The Visual Computer*, 18:1–19, 2002.

[McC86] N. McClamroch. Singular systems of differential equations as dynamic models for constrained robot systems. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 21–28, Apr 1986.

[Rot82] Scott D. Roth. Ray casting for modeling solids. In *Computer Graphics and Image Processing*, volume 18, pages 109–144, 1982.

[TBN00] Bill Tomlinson, Bruce Blumberg, and Delphine Nain. Expressive autonomous cinematography for interactive virtual environments. In *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*, pages 317–324, New York, NY, USA, 2000. ACM.

[TTK96] K. Tarabanis, R.Y. Tsai, and A. Kaul. Computing occlusion-free viewpoints. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(3):279 –292, mar 1996.

[YNZ89] Pei F. Jia Yuan N. Zheng, J.Y.S. Luh. Integrating two industrial robots into a coordinated system. *Computers in Industry*, 12(4):285–298, August 1989.

[ZL86] Y. Zheng and J. Luh. Joint torques for control of two coordinated moving robots. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 1375–1380, Apr 1986.