NTNU
Det skapende universitet

Bevegelsesplanlegging og regulering av mobile robotmanipulatorer

Audun Korneliussen

Master i teknisk kybernetikk
Oppgaven levert: Juni 2011
Hovedveileder: Geir Mathisen, ITK

Norges teknisk-naturvitenskapelige universitet

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk
Institutt for teknisk kybernetikk

MASTER THESIS

Kandidatens navn: **Audun Korneliussen**

Fag: **Engineering cybernetics**

Oppgavens tittel (norsk): **Bevegelsesplanlegging og regulering av mobile robotmanipulatorer**

Oppgavens tittel (engelsk): **Motion planning and control of mobile robot manipulators**

Bakgrunn:

Mobile robot manipulators will be prevalent in addressing many future needs such as contributing to the caring for an aging population, as well as monitoring environmental pollution and removing contamination.

In order to fulfil this vision such robots need to be able to navigate safely and autonomously in unstructured environments. In other words, mobile robot manipulators must be able to seamlessly coordinate and navigate their "hands and feet" in environments containing both static and moving objects. Such navigation is the topic of this thesis assignment.

Tasks:

1. Perform a literature survey on
   - State-of-the-art sample-based and feedback-based motion planning techniques.
   - Control of mobile manipulators.
2. Design a set of algorithms for integrated motion planning and control of mobile robot manipulators based on the framework given by Yang and Brock [1].
   - Identify shortcomings of Yang and Brock's work, and supplement and suggest extensions where necessary.
3. Implement the algorithms on a mobile robot manipulator simulator software package.
   - The implementation should preferably be compatible with the Seekur Jr robot and the Schunk LWA 3 robot manipulator available at SINTEF. These systems have ROS integration. Use of the ROS is thereby suggested.
4. Analyse the behavior and performance of the implemented system.
   - Create a test suite which demonstrates key functionality of the motion planner.

[1] Y. Yang and O. Brock, “Elastic roadmaps – motion generation for autonomous mobile manipulation”, Autonmous Robots 28, 2010.

Oppgaven gitt: 10.01.2011
Besvarelsen leveres: 06.06.2011
Besvarelsen levert:
Utført ved Institutt for teknisk kybernetikk
Veileder: Forsker Sigurd A. Fjerdingen, SINTEF Anvendt Kybernetikk
Forsker Aksel A. Transeth, SINTEF Anvendt Kybernetikk

Trondheim, den 10.01.2011

Geir Mathisen
Faglærer

Forord

Under arbeidet med fjorårets prosjektoppgave ble jeg av oppgaveveileder Sigurd Fjerdingen overlevert en artikkel som beskrev en ny og interessant metode for bevegelsesplanlegging. Artikkelen omhandlet *det elastiske veikartet* [23] og ble grunnlaget for denne oppgaven.

Gjennom arbeidet med denne rapporten har veiledere Aksel A. Transeth og Sigurd Fjerdingen hele veien gitt uvurderlig støtte og tilbakemelding. Ikke minst har faglærer Geir Mathisen gitt svært konstruktiv tilbakemelding både etter fjorårets prosjektoppgave og denne rapportens tidlige utkast. Jeg er svært takknemlig for støtten de alle har gitt.

Trondheim, 6. juni 2011
Audun Korneliussen

Når en mobil robot begir seg ut i et dynamisk miljø for å utføre en oppgave er det mye den må ta hensyn til. Ikke bare må roboten holde rede på hvor den er og hvor den skal, den må også sørge for at kinematiske, dynamiske og oppgavebestemte begrensninger overholdes. Enda viktigere er det at roboten ikke uforvarende støter borti mennesker, dyr eller gjenstander som befinner seg i området. Denne rapporten presenterer et design og en implementasjon av et algoritmesystem som skal håndtere utfordringene en slik robot står overfor. Systemet er basert på det *elastiske veikartet* utformet av Yang og Brock [23], som er en nyskapende og effektiv tilnærming til problemet bevegelsesplanlegging. Systemet presenteres i sammenheng med en bred oversikt over litteratur og teori som behandler planleggingstemaet, med fokus på samplings- og tilbakekoblingsbasert planleggingsmetodikk.

Det implementerte algoritmesystemet demonstreres og analyseres gjennom en rekke simulerte testscenarioer. Resultatet er et utvidbart og fremtidsrettet planleggingssystem som kan planlegge og utføre en bevegelse mellom en start- og målposisjon, og kan i sanntid håndtere endringer i robotens omgivelser.

Innhold

1. Innledning

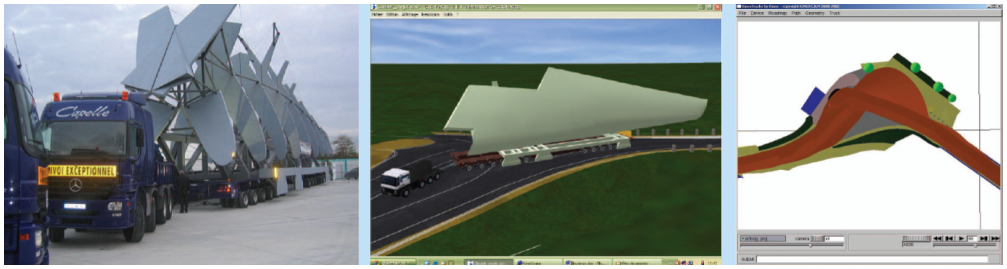
Dette kapittelet gir et innblikk i bakgrunnen for det arbeidet som rapporten presenterer. Arbeidet som er utført settes i sammenheng med eksisterende litteratur og problemstillinger før det helhetlige resultatet av innsatsen presenteres.

Rapporten baserer seg på litteratur som er i all hovedsak engelskspråklig. Der det er hensiktsmessig og forståelig oversettes engelske uttrykk til norsk. Når ord og uttrykk oversettes tydeliggjøres både det originale ordet samt oversettelsen.

1.1. Motivasjon

Roboter har lenge vært et vanlig syn langs produksjonslinjer og i menneskefiendtlige miljø. De har lært kunsten å gå i trapper og har utforsket solsystemet. Riktignok er manipulatorene man så ofte forbinder med industrien fastmontert i et forutsigbart miljø, mens hver hjulrotasjon mars-roveren tar er nøye vurdert av dens menneskelige operatører. Betydelig innsats har blitt lagt ned i utviklingen av planleggings- og kontrollsystemer for roboter. Blant annet har utformingen av planleggingsalgoritmer som gir presise og optimale geometriske kurver som roboten kan følge ført til store besparelser i industrien. LaValle [15, s. 7] nevner hvordan ingeniører kan på høyt nivå spesifisere oppgaver for en robot som forsegler bildeler, uten å måtte utforme den fullstendige banen roboten skal følge. På tilsvarende måte er planleggingsalgoritmer til stor hjelp når en spesialdesignet lastebil med tjue par styrehjul skal frakte deler av et Airbus A380-fly gjennom franske småbyer [15, s. 16], som vist i figur 1.1. Felles for de fleste roboter er tett instruksjon fra mennesker og detaljert kunnskap om området de befinner seg i. Fullstendig autonom oppgaveutførsel for en robot er foreløpig utenfor rekkevidde [23].

Innledningen gir et innblikk i den teoretiske bakgrunnen for bevegelsesplanlegging og tilhørende problemstillinger som omtales mer utdypende i kapittel 2. Oppgaven resulterer i et design og implementasjonen av et system for bevegelsesplanlegging basert på algoritmerammeverket «Elastic Roadmaps» som er beskrevet av Yang og Brock [23]. Rammeverket muliggjør spesifisering av oppgaver for en robot på et høyt nivå og gir roboten en større grad av autonomi under oppgaveutførselen.



Figur 1.1.: Ruteplanlegging for en flydelfraktende lastebil med 20 par styrehjul. Bilderettigheter tilhører Kineo CAM.

1.2. Bakgrunn

Når en mobil robot skal ta seg inn i ukjent terreng er det mange hensyn som må tas. Bruk av ulike sensorer kan kartlegge området og orientere roboten i terrenget. Desto viktigere er det at den tar hensyn til mennesker, dyr og gjenstander som kan befinne seg i området. Robotens oppgave kan innebære krav og begrensninger av ulik art som kompliserer dens oppførsel ytterligere. La roboten illustrert på forsiden være et eksempel. Den skal hente en kopp kaffe med et tenkt gripeverktøy, og gi koppen til en kaffetørst person som befinner seg i et annet rom. For å utføre oppgaven må ikke roboten bare kjenne til personens plassering, men også bygningens arkitektur og nåværende møbelkonfigurasjon. Den må i sanntid kunne oppdage mennesker og uforutsette hindringer for å unngå kollisjoner. Robotens ledd har kinematiske begrensninger som må overholdes, i tillegg til begrensninger

pålagt av oppgaven: kaffekoppen må holdes vannrett og kan ikke utsettes for stor akselerasjon. Dessuten må roboten begrense kraften som påtrykkes den relativt skjøre kaffekoppen. Alle disse hensynene medfører betydelige krav til prosessering og hyppige sensoravlesninger. For å utføre oppgaven på en god måte trengs det metodikk for planlegging på et høyt abstraksjonsnivå og mulighet til å kontinuerlig ta hensyn til hendelser som oppstår under utførelsen.

Planlegging omfatter i robotikken formulering av en eller annen *plan* som forflytter roboten fra en utgangsposisjon og til en målposisjon. I et dynamisk miljø vil en slik plan være utsatt for hyppige endringer i takt med at robotens virkelighetsbilde oppdateres. Når roboten i tillegg må utvise hensyn som nevnt i eksempelet ovenfor så vil planleggingsalgoritmer som tidligere er brukt på roboter i statiske omgivelser bli for ineffektive. En direkte årsak til det store prosesseringsbehovet ved bevegelsesplanlegging er antallet frihetsgrader, de uavhengige variabler som samlet beskriver posisjonen og orienteringen til ethvert punkt på roboten. En menneskelignende robot kan eksempelvis ha 36 frihetsgrader [9]. Robotens frihetsgrader gir opphav til *konfigurasjonsrommet* [15, kap. 4], det geometriske rommet som inneholder alle gyldige sett med frihetsgradskonfigurasjoner. Hindringer i robotens *arbeidsrom* gir en todeling av konfigurasjonsrommet, der alle gyldige planlagte veier må gå gjennom den hindringsfrie delen av rommet.

En klasse planleggingsalgoritmer tar sikte på å kartlegge konfigurasjonsrommet for så å finne en gyldig vei gjennom den hindringsfrie delen av rommet. *Kombinatoriske* algoritmer [15, kap. 6] beskriver konfigurasjonsrommet fullstendig og oppnår algoritmisk kompletthet på bekostning av høy algoritmisk kompleksitet. Denne kompleksiteten gjør at bruk av slike algoritmer i dag ikke er praktisk gjennomførbar. Hvis man kan diskretisere og redusere konfigurasjonsrommet til en håndterlig datastruktur kan man løse planleggingsproblemet med en enklere søkealgoritme [8]. Eksempelvis «Grid search» [15] beskriver en fremgangsmåte der konfigurasjonsrommet deles opp i en søkbar «grid» av håndterlig størrelse. Slike algoritmer tar i liten eller ingen grad hensyn til robotens dynamikk. *Kurveplanleggere* bruker derimot kunnskap om robotens dynamikk og kinematikk til å utarbeide en mer nøyaktig geometriske kurve i rommet som roboten kan følge. Ofte basert på en allerede løst spesifisert rute og et optimalitetskri-

terium. Kurveplanleggere deles inn i «online»- og «offline»-klasser, der «offline»-algoritmer planlegger en fullstendig kurve før bevegelsen starter mens «online»-algoritmer endrer kurven underveis etter behov [11]. «Elastic strips» [11] er en slik «online»-algoritme som tøyer en planlagt kurve for å unngå hindringer som oppdages underveis. «Online»-algoritmer er også brukt til oppgaver som krever høy presisjon. Der «offline»-algoritmer baserer seg på modellert dynamikk med en viss unøyaktighet kan «online»-algoritmer kompensere for modellunøyaktigheter under bevegelsesutførselen. For eksempel bruker «CNC»-maskiner slike algoritmer for å få stor nøyaktighet i fremstillingen av ulike materialdeler [11].

Når en kurve eller rute er formulert er det opp til *regulatoren* å sørge for at robotens aktuatorer flytter roboten i henhold til planen. Ulike regulatorer kan kompensere for dynamiske ulineariteter eller regulere etter ett eller flere optimalitetskriterier. Enkelte regulatorer utvisker grensen mellom planlegger og regulator og justerer aktivt den planlagte banen under bevegelsesutføring. Adaptive reguleringsstrategier justerer eksempelvis modeller og parametre basert på en oppfatning av hvor vellykket reguleringen er [21].

De fleste typene planleggingsalgoritmer nevnt i foregående avsnitt innebærer betydelig prosesseringskompleksitet. I et dynamisk miljø kan det skje endringer som fører til at man må forkaste en plan, i tillegg til at hurtig sensoravlesning er nødvendig for å oppdage endringen. Sensoravlesningsfrekvensen kan dermed overstige den gjennomførbare planleggingsfrekvensen og forårsake at mange planleggingsalgoritmer ikke vil gi en plan som reflekterer oppfattelsen av miljøets nåværende tilstand. For å bøte på prosesseringsproblemet har det i de senere årene blitt utviklet algoritmer som baserer seg på sampling av robotens konfigurasjonsrom. En tilfeldig eller deterministisk *sampel* av konfigurasjonsrommet blir av en kollisjonsdetekteringsalgoritme bestemt å ligge enten i den gyldige eller ugyldige delen av konfigurasjonsrommet. Etter tilstrekkelig antall samplinger dannes det en vei med høy samplingstetthet mellom en start- og målkonfigurasjon. Konfigurasjonsromssampling kan resultere i en effektiv planleggingsprosess, og en plan det er billig å forkaste hvis det oppstår endringer som ugyldiggjør planen. «Rapidly exploring Random Trees» [15, kap. 5.5] konstruerer for eksempel en tredatastruktur av konfigurasjoner basert

på en tilfeldig sampling av konfigurasjonsrommet. Andre algoritmer tar sikte på å vedlikeholde ett eller flere *samplingstrær* for å gi en vedvarende representasjon av omgivelsene. *Veikartsalgoritmer* tar også sikte på å gi en vedvarende minimalistisk representasjon av omgivelsene, ofte med samplingsalgoritmer som en underplanlegger. Det resulterende veikartet som algoritmene gir er en graf med noder og kanter som beskriver konfigurasjonsrommet. Blant veikartsklassen av algoritmer finner vi den *elastiske veikartsalgoritmen* («Elastic Roadmaps» [23]) som tar sikte på effektiv planlegging og tilfredsstilling av krav og begrensninger av den typen som den nevnte kaffebringeroboten er underlagt. Dette algoritmesystemet omtales i større detalj i kapittel 2.7.

1.3. Bidrag

Denne oppgaven gir en innføring i planleggingsalgoritmer for roboter, med fokus på samplings- og tilbakekoblingsbasert bevegelsesplanlegging. Spesielt er algoritmer tiltenkt bruk på roboter i et dynamisk miljø vektlagt. Et slikt algoritmerammeverk, «Elastic Roadmaps» [23] blir ingående beskrevet. Rammeverket beskriver hvordan ulike prioriterte hensyn kan kombineres til å gi et helhetlig pådrag mot roboten slik at den forflyttes mot en målkonfigurasjon. I motsetning til algoritmer som definerer én vei eller kurve beskriver rammeverket et uendelig antall veier gjennom konfigurasjonsrommet ved hjelp av en veikartstilnærming. Det designes en implementasjon av rammeverket tiltenkt fremtidig bruk på Sintefs Seekur Jr. robotplattform med påmontert Schunk LWA 3 manipulatorarm. Designet utgjør et system av algoritmekomponenter, støttekomponenter, integrasjon mot robotens operativsystem, simulering og visualisering. Upresise og mangelfulle deler av rammeverksbeskrivelsen utfylles i designet. Det implementerte designet simuleres gjennom en rekke testscenarioer og vurderes som et grunnlag for videre utvikling. Resultatet av implementasjonen er et system for bevegelsesplanlegging som kan ta hensyn til de mange utfordringer som oppstår når en robot med et betydelig antall frihetsgrader befinner seg i et dynamisk miljø. Systemets modularitet gjør det blant annet mulig å samtidig tilfredsstille ulike krav til oppdateringsfrekvenser på ulike sensorer. Et utvidbart design basert på velprøvde

designmetodikker og velutviklede tredjepartsbibliotek skal sørge for at oppgaveresultatet kan gjennomgå fremtidig bruk og utvikling.

2. Litteraturstudie

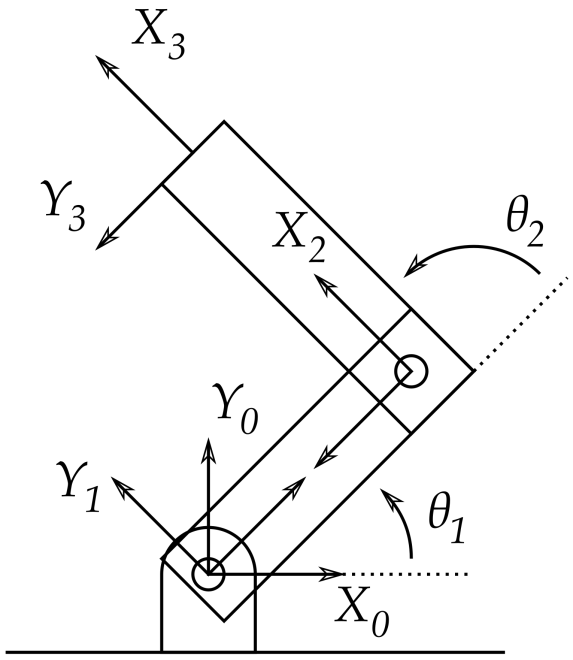
Bevegelsesplanlegging er et bredt begrep som har ulik mening innenfor ulike fagfelt. Dette kapittelet tar først og fremst for seg bevegelsesplanlegging i et robotikkperspektiv, men vil også påpeke eventuelt ulike tolkninger der det er hensiktsmessig. Innledningsvis i kapittelet introduseres stoff som er felles for mye av teorien resten av kapittelet omtaler. Det gis det et innblikk i planlegging fra et overordnet nivå og ned til mer konkret styring og kontroll av et robotsystem. Den elastiske veikartsalgoritmen presenteres inngående i kapittelets siste del.

2.1. Robotspesifikt planleggingsgrunnlag

Store deler av litteraturen rundt planlegging for roboter er bygd opp rundt en felles beskrivelse av robotens egenskaper. Ved å bruke et felles grunnlag i form av konfigurasjonsrom og arbeidsrom kan mange robotspesifikke detaljer abstraheres vekk og algoritmer kan gjøres allsidige. Her presenteres konfigurasjonsrommet og arbeidsrommet, som er grunnleggende konsept for bevegelsesplanlegging av roboter. I tillegg gis det et innblikk i regulering i «joint space» og «operational space», som er relevant blant annet innen den elastiske veikartsalgoritmen.

2.1.1. Konfigurasjonsrom og arbeidsrom

Enhver fysisk robot befinner seg i et *arbeidsrom* $\mathcal{W}$, der $\mathcal{W} = \mathbb{R}^2$ eller $\mathcal{W} = \mathbb{R}^3$. I $\mathcal{W}$ er det, i tillegg til roboten, hindringer og objekt roboten samhandler med. Settet med hindringer benevnes som $\mathcal{O}$, der $\mathcal{O} \subset \mathcal{W}$. Robotens konfigurasjonsrom, $\mathcal{C}$, er settet med robotens gyldige konfigurasjoner [15, kap. 4]. En konfigurasjon er samlingen variabler som

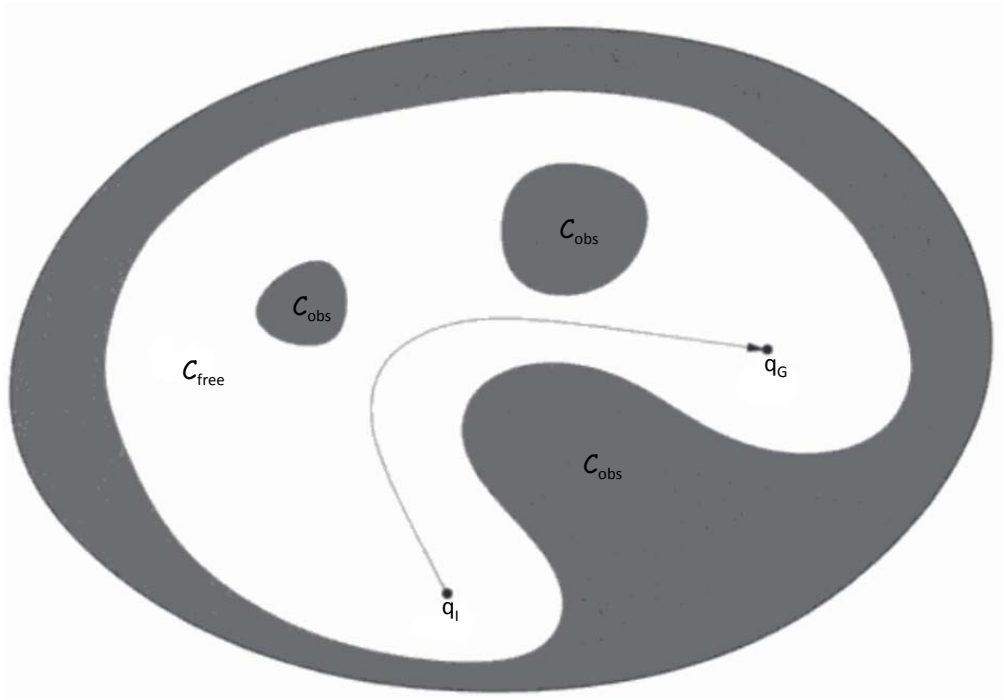


Figur 2.1.: Robotarm med to rotasjonsledd. Hentet fra [15].

angir posisjonen til ethvert punkt på roboten. Robotarmen vist i figur 2.1 vil eksempelvis ha et konfigurasjonsrom som inneholder alle gyldige kombinasjoner av θ_1 og θ_2, som er vinkelutslagene til armens to rotasjonsledd. I motsetning til robotens *tilstandsrom* inneholder konfigurasjonsrommet ingen informasjon om robotens bevegelse eller dynamikk. $\mathcal{C}$ er et n-dimensjonalt rom, der n er antall frihetsgrader, og tar vanligvis form av et n-dimensjonalt *manifold*. Et manifold er en «fin» topologi i den forstand at alle punkter på et manifold kan forstås intuitivt som en flate i Euklidsk geometri. Mer presist: for «nabolaget» $\mathcal{A}$ til ethvert punkt x i manifoldet $\mathcal{M} \subseteq \mathbb{R}^m$ finnes det en homeomorfisme til $\mathbb{R}^n$, der n er konstant for en gitt x og angir manifoldets dimensjon [15, s.111].

Når man beskriver robotens kinematikk ved hjelp av et konfigurasjonsrom oppnår man en abstrahering fra robotens fysiske detaljer. Dermed kan planleggingsalgoritmer brukes til å løse en lang rekke problemer som lar seg abstrahere til et slikt topologisk rom. Skal en løsning på planleggingsproblemet i konfigurasjonsrommet finnes, er det viktig at konfigurasjonsrommets topologi forstås og kan brukes av planleggingsalgoritmen.

Hindringer i arbeidsrommet $\mathcal{W}$ gir opphav til en todeling av konfigurasjonsrommet $\mathcal{C}$. $\mathcal{C}_{free}$ er den åpne delen av $\mathcal{C}$ der roboten fritt kan bevege seg, mens $\mathcal{C}_{obs}$ er den utilgjengelige delen av $\mathcal{C}$. Følgelig er $\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{obs}$. Figur 2.2 illustrerer et todimensjonalt konfigurasjonsrom med hindringer. Ettersom en robot kan ha et ubegrenset antall frihetsgrader blir konfigurasjonsrommets dimensjon tilsvarende ubegrenset. Det finnes dermed ingen enkel sammenheng mellom $\mathcal{W}$ og $\mathcal{C}_{obs}$, og fullstendig kartlegging av $\mathcal{C}_{obs}$ er svært vanskelig. De fleste, om ikke alle planleggingsalgoritmer tiltenkt bruk i sanntid unngår dermed en fullstendig kartlegging av $\mathcal{C}$. I enkelte tilfeller med enkle roboter og få frihetsgrader kan $\mathcal{C}_{obs}$ betraktes som en folding mellom hindringene $\mathcal{O}$ og konfigurasjonsrommet $\mathcal{C}$.



Figur 2.2.: Illustrasjon av planleggingsproblemet der målet er å finne en vei mellom q_I og q_G i $\mathcal{C}_{free}$ [15].

2.1.2. Joint space og operational space

Ved planlegging og styring av roboter, spesielt robotmanipulatorer[1], involverer ofte oppgaven bruk av robotens verktøyledd[2]. Robotens øvrige ledd og frihetsgrader muliggjør bruken av verktøyleddet, men er i seg selv ikke interessante i forhold til robotens oppgave. Det er dermed en fordel å kunne handle innenfor robotens «operational space», eller *operasjonsrom*, som behandler kun robotens verktøyledd. Operasjonsrommet er dermed et subsett av robotens konfigurasjonsrom. I motsetning til operasjonsrommet omhandler robotens «joint space», eller *leddrom*, samtlige av robotens ledd. Khatib[10] formulerer en metode for kontroll av posisjon og krefter innen operasjonsrommet.

Den dynamiske beskrivelsen av en robotmanipulator i leddrommet er gitt av Lagrange-ligningen:

$$A(q)\ddot{q} + b(q,\dot{q}) + g(q) = \Gamma \tag{2.1}$$

Der $A(q)$ er robotens kinetiske energimatrise, mens $b(q,\dot{q})$, $g(q)$ og Γ er henholdsvis Coriolis-, tyngdekraft- og kraftvektorer. Γ representerer den kraften som påtrykkes robotens ledd av aktuatorer. Khatib viser hvordan man kan koble Γ, som er i leddrommet, fra en spesifisering av verktøyleddets bevegelse ved hjelp av «ulineær dynamisk frakobling»[10, s.5]. En kraftvektor F_m^* angir en kraft mot verktøyleddet som en funksjon av ønsket posisjon x_d og ønsket hastighet av verktøyleddet, $\dot{x}_d$.

$$F_m^* = -k_v(\dot{x} - v \cdot \dot{x}_d) \tag{2.2}$$

Khatib viser hvordan Γ kan omskrives som en funksjon av F_m^*:

$$\Gamma = J^T(q) \cdot \Lambda(q) \cdot F_m^* + \tilde{b}(q,\dot{q}) + g(q) \tag{2.3}$$

Metoden beskrives i detalj i [10].

[1] Også kalt robot*arm*

[2] Verktøyledd, eller «end effector», er som regel det ytterste leddet i en robotmanipulator. Dette leddet står for robotens interaksjon mot en oppgave.

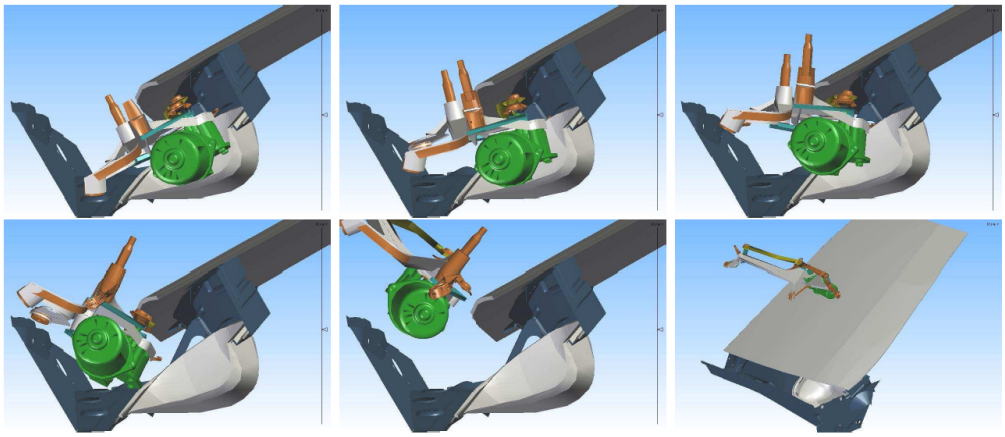
2.2. Med en plan som løsning

En problemstilling som illustrerer planleggingsproblemet er «Pianoflytterens problem» [15, s.131]. Gitt et sett rom i et hus der det i ett av rommene befinner seg et piano. Oppgaven består av å planlegge et sett bevegelser, eller en *rute*, som forflytter pianoet fra utgangsposisjonen til en målposisjon i et annet rom, uten å kollidere med eventuelle hindringer langs veien. Pianoproblemet kan videre utvides til å beskrive problem i ulike fagfelt. Lar man for eksempel pianoet byttes ut med en robot kan oppgaven betraktes som et reguleringsproblem. Deler man opp pianoets bevegelser i diskrete steg tar det form av mer datavitenskapelig problemløsing. Tilsvarende kan en lang rekke problemer som krever et sett med planlagte steg for å løses betraktes som bevegelsesplanlegging. Bevegelsesplanlegging legger seg dermed under den «klassen» av problemer som krever en *plan* for å løses. En plan enten i form av et sett diskrete steg, eller som en kontinuerlig planleggingsfunksjon. LaValle [15] nevner blant annet *proteinfoldingsproblemet*, som er problemet med å avgjøre om et molekyl kan plasseres i en kompleks proteinstruktur. Problemet kan betraktes som et et geometrisk «puslespill» på lik linje med problemet vist i figur 2.3. Med fokus på planlegging for roboter betraktes en *kurve* i denne oppgaven som en kontinuerlig geometrisk representasjon av en plan. En planleggingsprosess resulterer ofte i en geometrisk vei eller kurve, som kan for eksempel være en parametrisert kurve («spline») i et matematisk rom.

2.3. Planlegging og veiplanlegging

På et overordnet nivå handler planlegging om å finne en uhindret vei fra start til mål i et arbeidsrom eller konfigurasjonsrom. Betrakter man planleggingsproblemet på et slikt overordnet nivå kan man benytte planleggingsalgoritmer til en rekke svært ulike problemer. Da tas det ikke hensyn til dynamikken til den som skal følge den resulterende veien. Planlegges det i arbeidsrommet tar man nødvendigvis heller ikke hensyn til de kinematiske begrensninger som skyldes robotens konstruksjon.

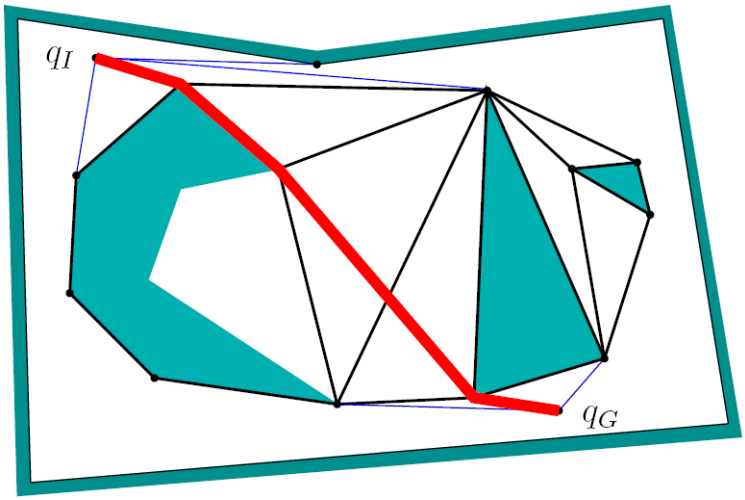
Kombinatoriske algoritmer danner seg et fullstendig bilde av $\mathcal{C}_{obs}$ ved



Figur 2.3.: Problem fra bilindustrien med å montere eller fjerne en vindusviskermotor fra et hulrom i bilchassiset. Hentet fra [15, s. 7].

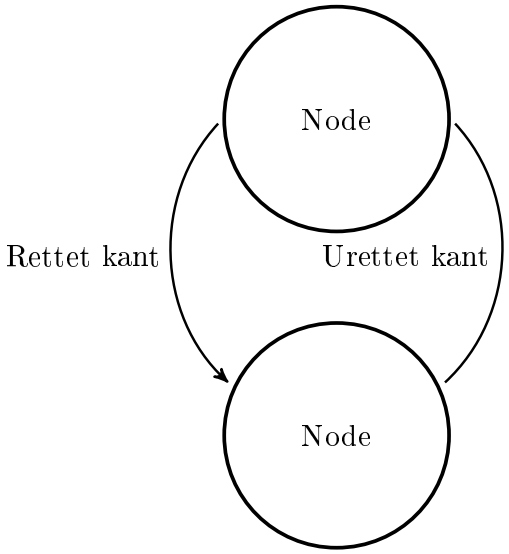
for eksempel «vertical cell decomposition»[15, s.209]. Her defineres $\mathcal{C}_{obs}$ ved hjelp av konvekse «celler» som bygges opp ved en gradvis «utforsking» av $\mathcal{C}_{obs}$. Prosessen resulterer i et endelig sett med celler som representerer det uendelige settet $\mathcal{C}_{obs}$. Basert på settet med celler kan man konstruere et *veikart*, som er en graf med et sett *milepæler*(noder) og kanter mellom milepæler. Figur 2.5 illustrerer datastrukturen. Kantene i veikartet representerer gyldige bevegelser mellom milepæler i $\mathcal{C}_{free}$. Problemet med å finne en vei fra start til mål i $\mathcal{C}_{free}$ er dermed redusert til et søk i en graf, der eksempelvis Dijkstras algoritme eller dybde/bredde først-søk kan benyttes [8]. Veikarttankegangen benyttes ikke bare innen kombinatoriske algoritmer, men kan representere ethvert arbeids- eller konfigurasjonsrom. *Kombinatoriske* algoritmer refererer dermed til selve kartleggingen av $\mathcal{C}_{obs}$, som i motsetning til andre metoder og algoritmer er fullstendig. Den komplette kartleggingen krever riktignok at alle hindringer i $\mathcal{W}$ kan representeres som definerte geometriske størrelser. Algoritmer som kartlegger $\mathcal{C}_{obs}$ fullstendig er algoritmisk komplette, som innebærer at de kan innen endelig tid finne løsningen på planleggingsproblemet hvis løsningen eksisterer, og vil innen endelig tid rapportere at det ikke finnes noen løsning hvis det

ikke finnes en løsning på problemet. Figur 2.4 illustrerer en plan gjennom et konfigurasjonsrom der $\mathcal{C}_{obs}$ er fullstendig modellert med polygoner.



Figur 2.4.: Kombinatorisk planlegging. Hindringer er modellert som polygoner og den korteste veien mellom q_I og q_G i konfigurasjonsrommet er illustrert med en tykk strek. Hentet fra [15].

Hvis $\mathcal{W}$ og $\mathcal{C}_{obs}$ kan diskretiseres vil problemet reduseres til et «enkelt» søk mellom gyldige og ugyldige diskrete deler av settet. Søk i et slikt diskretisert sett beneves ofte som «Grid search» [15]. Diskretisering av et kontinuerlig konfigurasjons- og arbeidsrom til en robot er ofte problematisk. For grov inndeling fører lett til at løsninger forsvinner, og for fin inndelig gir ineffektiv søking. En løsning er å velge en stadig finere inndeling av $\mathcal{C}$ til en løsning er funnet. Problemet «grov og effektiv mot fin og ineffektiv» er noe som er aktuelt i mange sammenhenger, deriblant innen sampling-basert planlegging som omtales i del 2.6.1.



Figur 2.5.: En generisk grafdatastruktur med noder og kanter.

2.4. Kurveplanlegging

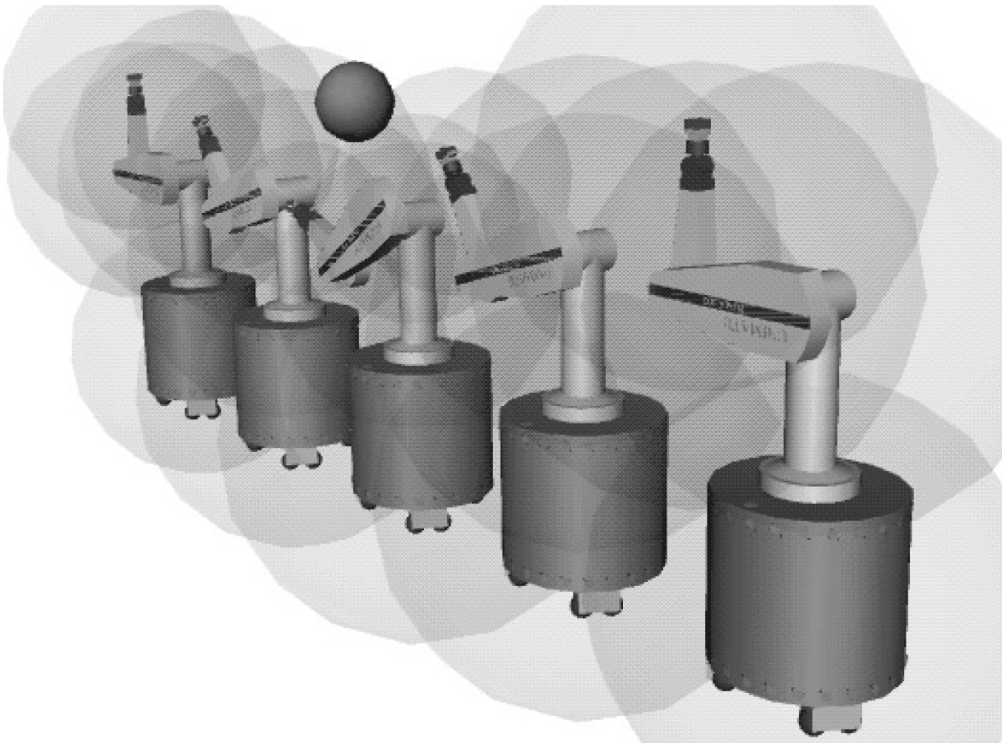
Når veiplanleggeren har gitt en mer eller mindre fast definert vei i $\mathcal{W}$ eller $\mathcal{C}$ er det opp til kurveplanleggeren å finne en mer nøyaktig spesifisering av den geometriske kurven roboten skal følge. Robotens muligheter til å følge kurven vil avhenge av robotens kinematiske og dynamiske egenskaper, og det er noe kurveplanleggeren tar hensyn til når den «finjusterer» planen. I tillegg kan kurveplanleggeren være underlagt ulike optimalitetskriterer som skal sørge for optimal utnyttelse av moment og tid, sørge for en jevn bevegelse eller lignende [21]. Man kan grovt sett dele denne klassen av planleggingsalgoritmer inn i «online» og «offline»-grupper, der «offline»-algoritmer kjøres på forhånd mens «online»-algoritmer kjører etter hvert som roboten beveger seg. «Offline»-algoritmer er dermed avhengig av god modellering av robotens kinematikk og dynamikk, mens «online»-algoritmer kan i større grad kompensere for unøyaktigheter og uforutsette hendelser underveis [11]. Det er verdt å påpeke at kurveplanleggere ofte antar dynamiske, *men kjente* omgivelser [23].

Kröger omtaler i større detalj utviklingen rundt kurveplanlegging [11, s.17]. Dette avsnittet er et kortfattet sammendrag av Krögers omtale. Roth og Kahn utviklet de første konseptene og algoritmene for regulering av lineariserte robotmanipulatorer i 1971, og oppnådde tidsoptimal jevn regulering av slike. I løpet av 1980-tallet fortsatte utviklingen i retning ulineære systemer og spesifisering av robotens dynamikk gjennom parametriserte geometriske kurver. Ethvert punkt på kurven kan så angi maksimal akselerasjon og hastighet for robotens aktuatorer som kan benyttes til å gi en tidsoptimal bane. Tankegangen rundt parametrisering av kurver har blitt utvidet til å inkludere ulike geometriske konsept samt ulike optimalitetskriterier. Eksempelvis bruk av «B-spliner» og inkluderingen av akselerasjonsbegrensninger for å gi en «optimalt jevn» bevegelse.

Med utgangspunkt i «offline»-kurveplanlegging tar «online»-klassen av algoritmer hensyn til uforutsette hendelser som oppstår under kjøring samt unøyaktigheter i modelleringen av robotens kinematikk, dynamikk, omgivelser og lignende. Hvis det ikke er samsvar mellom den antatte og faktiske virkeligheten «offline»-planlagte kurver forholder seg til, så vil eventuelle optimalitetsvurderinger være av liten verdi. Tilsvarende vil forhåndsplanlagte baner være av liten verdi hvis det er usikkerhet rundt arbeidsrommet eller konfigurasjonsrommets form. En «online»-algoritme, «Elastic strips» [18], tilpasser eksempelvis en kurve beregnet «offline» i sanntid under bevegelsesutførsel for å unngå hindringer underveis. Algoritmen betrakter området rundt en planlagt rute som en «elastisk tunnel» som den planlagte kurven kan endres innenfor. Figur 2.6 viser hvordan en robot unngår en hindring innenfor tunnelen.

2.5. Regulering

Når en plan er formulert av en planleggingsalgoritme er det regulatorens oppgave å styre roboten i henhold til denne. Det innebærer en minimering av avviket mellom robotens faktiske tilstand og tilstanden planen beskriver. *Tilstand* omfatter her de parametre som en planleggingsalgoritme har spesifisert, for eksempel leddvinkler eller vinkelhastigheter. Planen kan være en kontinuerlig geometrisk kurve, eller en diskret samling av konfigurasjoner som beskriver en plan-



Figur 2.6.: Illustrasjon av «Elastic Strips»-algoritmerammeverket [18]. Et område rundt en tidligere planlagt kurve betraktes som en «elastisk tunnel» som kurven kan tøyes innenfor. Kurven er illustrert som et sett konfigurasjoner, og den elastiske tunnelen er det gråfargede området rundt kurven. Roboten unngår en hindring (en ball) innenfor tunnelen.

messig bevegelse (se samplingbasert planlegging på side 19). Det kan dermed være hensiktsmessig å skille mellom *kurvefølging* og tradisjonell *regulering* [21, s.135]. Med tradisjonell regulering menes punkt-til-punkt-regulering der regulatoren skal styre robotens tilstand mot et gitt *settpunkt*, helst uten forstyrrelser eller svingninger. Kurvefølging derimot, innebærer en mer eller mindre rigid begrensning av posisjon, hastighet eller andre parametre kurven spesifiserer. Valget av regulator vil dermed være avhengig blant annet av planens form, og hvor nøyaktig planen er spesifisert. Hvis planen innebærer regulering av kraft eller moment mellom robot og omgivelser blir problemet ganske annerledes enn kurve- eller settpunksfølging. Dette kapittelet tar ikke stilling til kraftreguleringsproblemet, men Villani og Schutter gir en introduksjon i [21, kap. 7].

En rekke ulike reguleringskonsept kan benyttes innen robotstyring. Feltet er bredt og dette avsnittet gjengir et knapt utdrag fra bakgrunnsstoffet presentert i [11], [21]. Av de enklere konseptene er PID-regulering, som kan benyttes uten å kjenne til systemets dynamikk. Bruk av for eksempel Ziegler-Nichols metode for parameterinnstilling kan gi tilstrekkelig regulering, spesielt når det reguleres mot et fast settpunkt (i motsetning til kurvefølging). For å oppnå optimal (i en eller annen forstand) regulering må man derimot ta hensyn til systemets dynamikk. Inversdynamisk regulering og «feedback linearization» henholdsvis motvirker og «skjuler» systemets ulinearitet, og tilrettelegger for eksempelvis PD-regulering av det resulterende systemet. «Feedback linearization» transformerer det ulineærer systemet til en «indre løkke», mens en «ytre løkke» sørger for at det transformerte systemet oppfattes som lineært når man betrakter systemets inn- og utganger. Systemets dynamikk er dermed beholdt og tatt hensyn til, i motsetning ved inversdynamisk regulering der målet er å kansellere alle ulinearitet. En klasse reguleringsalgoritmer basert på «feedback linearization» kalt «computed-torque control» forsøker også å transformere systemet til en mer lineær og håndterlig form, gjerne der man ikke trenger nøyaktig kunnskap om robotens nyttelast. Videre finnes det adaptive strategier for alle de nevnte reguleringsalgoritmene. Disse tilpasser parametrene til reguleringsalgoritmen basert på en oppfatning av algoritmens vellykkethet i forhold til et eller flere optimalitetskriterer. Chung, Fu og Hsu beskriver de nevnte algoritmene i større detalj [21, s.133].

2.6. Bevegelsesplanlegging

Klassen av algoritmer som går under samlebetegnelsen bevegelsesplanlegging forsøker å kombinere veiplanlegging og kurveplanlegging til èn operasjon. Hvis man på et tidlig tidspunkt tar stilling til robotens kinematikk kan man forsikre seg om at roboten kan følge den resulterende planen. Bruk av robotens konfigurasjonsrom, som står sentralt i denne klassen av algoritmer, sørger for at robotens kinematikk blir en sentral del av planutformingen. Denne klassen algoritmer kan grovt sett deles inn i to deler: *samplingbaserte planleggere*, som utforsker konfigurasjonsrommet $\mathcal{C}$ ved å *sample* konfigurasjoner, og *potensialtfelt* som unngår kartlegging fullstendig.

2.6.1. Samplingbasert planlegging

Samplingbaserte bevegelsesplanleggere utforsker konfigurasjonsrommet ved å *sample* konfigurasjoner og ved hjelp av kollisjonsdetektering avgjøre om den samplede konfigurasjonen befinner seg i $\mathcal{C}_{obs}$ eller $\mathcal{C}_{free}$. Etter hvert vil settet med samplinger danne en forståelse av $\mathcal{C}_{free}$, og en gyldig bane mellom q_I og q_G kan formuleres. «Etter hvert» innebærer at en løsning vil finnes i endelig tid hvis det eksisterer en gyldig vei mellom q_I og q_G. Hvis det derimot *ikke* finnes en gyldig vei mellom q_I og q_G står algoritmen i fare for å aldri avslutte. Samplingbaserte bevegelsesplanleggere kan dermed ses på som et kompromiss mellom effektivitet og algoritmisk «kompletthet»[3] [15, kap. 5].

Ved å unngå eksplisitt modellering av $\mathcal{C}_{free}$ mister man den algoritmiske komplettheten fra kombinatoriske algoritmer. I stedet må man se til andre, løsere formuleringer av kompletthet. Samplingstetthet står her sentralt. Ved kontinuerlig sampling av $\mathcal{C}$ vil man nærme seg en vilkårlig konfigurasjon i $\mathcal{C}$ når tiden går mot uendelig. Enten man sampler $\mathcal{C}$ tilfeldig eller deterministisk vil man over tid få en høy nok samplingstetthet til at en gyldig vei vil finnes. Finnes det derimot ingen gyldige veier gjennom $\mathcal{C}_{free}$ vil algoritmen aldri kunne fastslå dette. LaValle [15] omtaler denne formen for algoritmisk kompletthet som

[3] En algoritme er komplett hvis den finner en løsning når løsningen eksisterer, eller korrekt angir at en løsning ikke eksisterer

«sannsylighetsmessig kompletthet», som innebærer at sannsynligheten for å finne en gyldig løsning konvergerer mot èn når samplingstettheten blir stor nok.

2.6.2. Probabilistic Roadmaps

«Probabilistic Roadmaps» (PRM) er en klasse med bevegelsesplanleggere som danner seg *veikart* over konfigurasjonsrommet. Ved å legge inn en innsats i opprettelsen av et slikt veikart kan man på en effektiv måte løse fremtidlige planleggingsspørringer innen det kartlagte konfigurasjonsrommet. PRM, også kalt «sampling-based roadmaps» eller bare «roadmaps» (veikart), oppretter en topologisk graf over $\mathcal{C}_{free}$ med noder som representerer konfigurasjoner og kanter mellom noder som indikerer kollisjonsfrie veier mellom nodene. Figur 2.5 på side 15 illustrerer en generisk grafstruktur.

Denne kartleggingen av $\mathcal{C}_{free}$ ble opprinnelig beskrevet av Kavraki et al [12] og var tiltenkt bruk i statiske omgivelser av holonomiske roboter. Kantene mellom nodene i grafen angir gyldige baner roboten kan følge. Banene utformes av en lokal planleggingsalgoritme. Ulike måter kan benyttes for å bestemme om to noder i grafen er direkte sammenkoblet. Yang og Brock [23] bruker eksempelvis en «synlighetstest» mellom noder i sin «Elastic Roadmap»-algoritme (se del 2.7). Hvis en rett linje kan uhindret trekkes mellom hvert ledd i to robotkonfigurasjoner i arbeidsrommet er konfigurasjonene (nodene) sammenkoblet. Jaillet og Siméon [13] har en mer nøyaktig bestemmelse av sammenkobling mellom noder i veikartet. En lokal «RRT» planleggingsalgoritme (omtales i del 2.6.3) brukes til å finne en vei mellom nodene.

Ved å beskrive konfigurasjonsrommet som et veikart reduseres altså bevegelsesplanleggingsproblemet til et søk i en topologisk graf.

2.6.3. Rapidly exploring Dense Trees

En måte å organisere samplinger av konfigurasjonsrommet og samtidig sørge for en høy nok samplingstetthet langs en gyldig vei er å benytte en tredatastruktur. Samplingsbaserte planleggingsalgoritmer

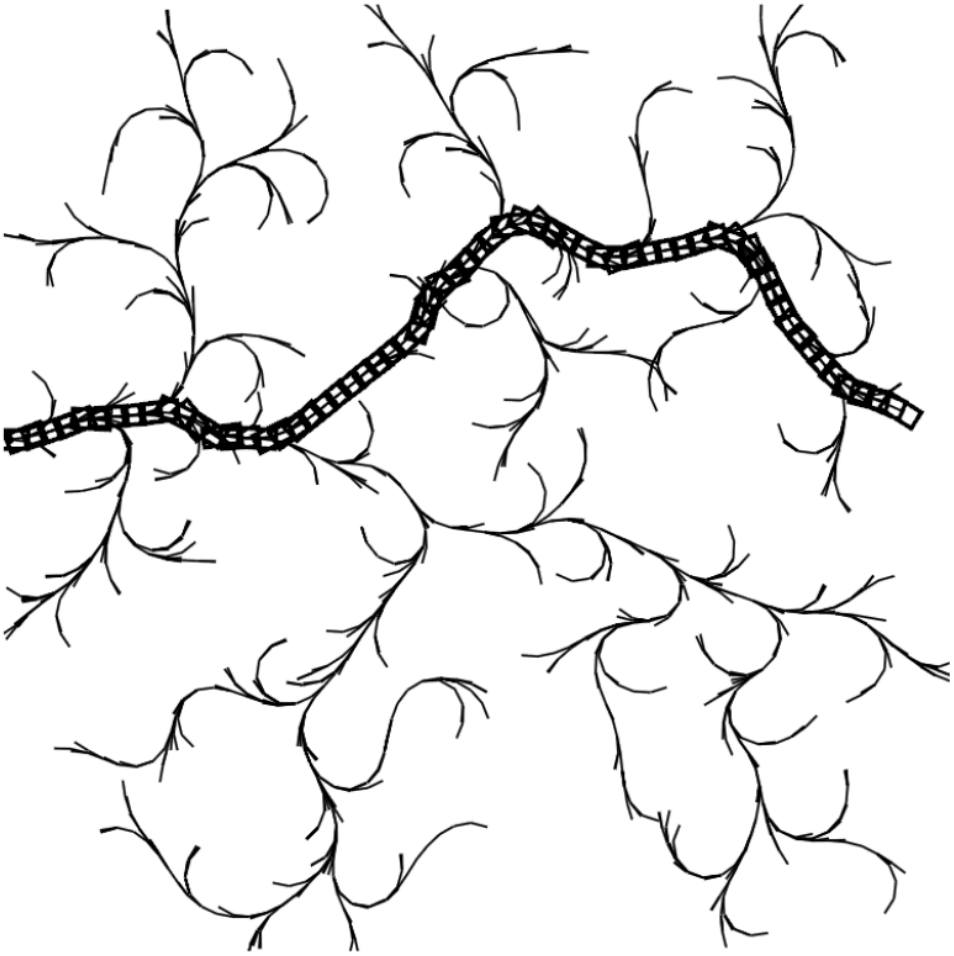
organisert under en slik datastruktur omtales ofte som «Rapidly Exploring Dense Trees» (RDT). Kortere samplingsrekker kombineres til å forme en trestruktur, der «trestammen» har en høy samplingstetthet, mens kortere «greiner» av samplinger vokser utover $\mathcal{C}_{free}$. Samplingstreet vokser (nye samplinger legges til) helt til en tett vei, eller «trestamme», går fra q_i til q_g. Prinsippet illustreres i figur 2.7. Algoritmen beskrives i større detalj i [15, kap. 5.5].

Som navnet antyder vil det gi en relativt rask utforsking av $\mathcal{C}$. Ulike strategier kan benyttes til å styre samplingen av nye samplingsgreiner i treet, der den enkleste er en tilfeldighetsstyrt strategi. Algoritmer som benytter en tilfeldig tilnærming omtales som «Rapidly exploring Random Trees» (RRT). Burns og Brock [7] omtaler ulike RRT-tilnærminger, deriblant bruken av «Voronoi-vektet bølgefrontsekspansjon» som skal sørge for en raskere sampling mot uutforskede deler av $\mathcal{C}$. I [7] beskrives det hvordan man kan bruke tidligere RDT-søk og et statistisk optimalitetskriterium om søkets vellykkethet til å styre nye søk. Kunnskap om konfigurasjonsrommets topologi tilegnes dermed gjennom erfaring. Har man a priori-kunnskap om konfigurasjonsrommets topologi kan denne bidra til et en mer effektiv samplings- og planleggingsprosess.

Dynamic Rapidly Exploring Random Trees

RDT og RRT er i hovedsak enkeltsøksalgoritmer. For et gitt sett med en utgangskonfigurasjon og målkonfigurasjon (q_i og q_g) konstrueres det ett samplingstre. Når det gis en ny målkonfigurasjon forkastes treet og algoritmen starter på nytt. Tilsvarende vil treet forkastet hvis det oppstår endringer i konfigurasjonsrommet, eksempelvis når en ny hindring oppdages. Det kan i mange tilfeller være hensiktsmessig å beholde hele, eller deler av treet gjennom flere søk. Spesielt hvis det kun oppstår små endringer i konfigurasjonsrommet kan store deler av treet beholdes uten kostnadskrevende endringer.

Som en direkte videreføring av RDT-klassen av algoritmer finner vi blant annet «Dynamic Rapidly Exploring Random Trees» (DRRT) og «Reconfigurable Random Forest» (RRF) [20]. Når deler av konfigurasjonsrommet ugyldiggjøres vil DRRT-algoritmen beskjære den delen av treet som er ugyldiggjort. Deretter vil treet igjen vokse med



Figur 2.7.: Illustrasjon over et tre av konfigurasjoner i $\mathcal{C}_{free}$. «Greiner» vokser utover og utforsker konfigurasjonsrommet. De kombineres til en «trestamme» med høy samplingstetthet som vil utgjøre den endelige bevegelsen gjennom $\mathcal{C}_{free}$. Hentet fra [15].

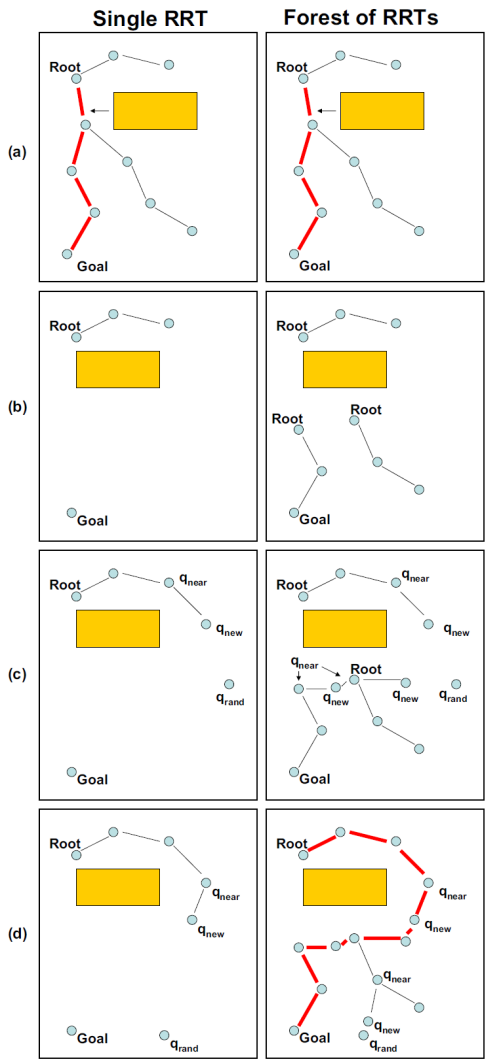
nye samplinger rettet mot den ugyldiggjorte delen av konfigurasjonsrommet. DRRT beholder og vedlikeholder dermed ett enkelt tre av konfigurasjoner gjennom hele dens levetid. «Random Reconfigurable Forest» tar vedlikeholdskonseptet videre og innfører muligheten for å opprettholde en hel skog av RRT-trær. Når deler av et tre ugyldiggjøres av endringer i $\mathcal{C}$ deles treet i to eller flere trær, der hvert tre har hver sin rotkonfigurasjon. Hvert tre vokser med nye samplinger og vil, hvis det finnes en gyldig vei, vokse sammen og igjen danne en kontinuerlig vei i $\mathcal{C}_{free}$. Figur 2.8 illustrerer hvordan et konfigurasjonstre kan endre seg når deler av konfigurasjonsrommet ugyldiggjøres.

Det finnes mange ulike algoritmer for «generelt vedlikehold», balansering og kombinering av ulike trestrukturer [8]. En algoritme som tar for seg eksplisitt vedlikehold av RRT-baserte skoger er «Lazy Reconfigurable Forest» (LRF) [20]. Den forsøker å minimere vedlikeholdskostnaden ved en «lat tilnærming» til treets gyldighet. I stedet for å verifisere hele skogen når konfigurasjonsrommet endres verifiseres kun konfigurasjonsrekken som utgjør veien mellom q_i og q_g. Kombinert med en tilbakeholden beskjæringsalgoritme forsøker LRF å unngå «overbeskjæring» av skogen, som gir stor vedlikeholdskostnad.

Verdien de persistente flersøksalgoritmene gir i form av «beholdt kunnskap» må dog vurderes opp mot vedlikeholdskostnadene av datastrukturene. Hvis endringene i konfigurasjonsrommet er av en slik art at mye vedlikehold kreves kan det tenkes at enkeltsøksalgoritmer er mer effektive.

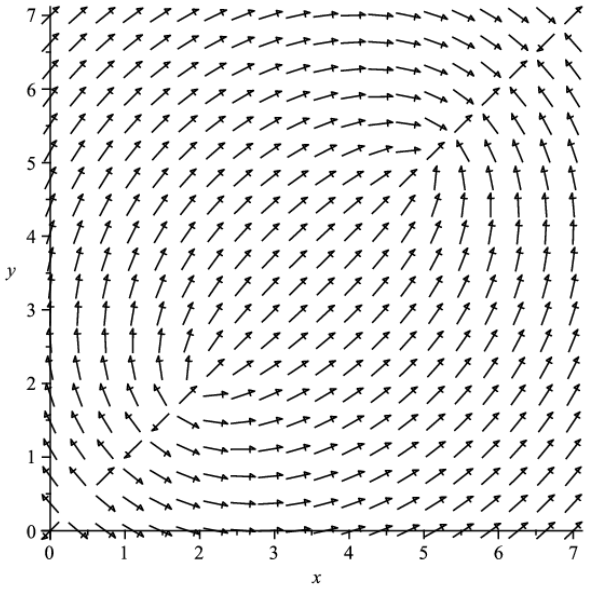
2.6.4. Potensialfelt

En måte å unngå modellering av $\mathcal{C}_{obs}$ er å opprette kunstige potensialfelt [17]. Roboten kan betraktes som en samling partikler under påvirkning av et kraftfelt $\mathcal{U}$ som har et globalt minimum i målkonfigurasjonen. Kraftfeltet kan konstrueres slik at roboten trekkes mot målkonfigurasjonen q_G samtidig som den frastøtes eventuelle hindringer. Ved å opprette $\mathcal{U}$ i robotens arbeidsrom unngår man kompliserte interaksjoner med konfigurasjonsrommet. Etter hvert som nye hindringer oppdages kan disse unngås ved stadig oppdatering av $\mathcal{U}$. Planleggingsproblemet kan dermed betraktes som et optimaliseringsproblem,



Figur 2.8.: Vedlikehold av enkelttrær og en skog av konfigurasjoner i $\mathcal{C}$. Rektangelet illustrerer en hindring som ugyldiggjør deler av konfigurasjonstreet. Kolonnen til venstre viser hvordan et enkelttre reduseres, mens i kolonnen til høyre deles treet i tre enkelttrær som etter hvert vokser sammen igjen. Hentet fra [20].

der målet er å finne det globale minimum i $\mathcal{U}$. Roboten trekkes mot q_G ved at for eksempel gradienten i potensialfeltet oversettes til en kraft eller moment som virker på robotens ledd. Figur 2.9 illustrerer et enkelt potensialtfelt med et globalt minimum.



Figur 2.9.: Illustrasjon av et potensialtfelt. Feltet har et tiltrekningspunkt (5,5) og et frastøtningspunkt (2,2).

Hovedproblemet med metoden er at man ikke kan garantere at $\mathcal{U}$ er fritt for lokale minimum. Ved «uheldig» plassering av hindringer (frastøtende felt) risikeres roboten å sette seg fast i et lokalt minimum uten å nå målkonfigurasjonen. Andre problemer med metoden inkluderer vanskeligheter med å passere mellom hindringer og oscillasjoner som kan oppstå rundt og mellom hindringer [22]. Problemet med lokale minimum er forsøkt løst blant annet ved å innføre «random walks» [17]. Hvis det mistenkes at roboten har satt seg fast i et lokalt minimum kan det foretas en «random walk», en bevegelse i en tilfeldig retning, og håpe at roboten kommer seg videre.

2.7. Det elastiske veikartet

Det elastiske veikartet som presenteres i [23] er et forholdsvis åpent definert algoritmesystem. Få detaljer bestemmes av definisjonen på systemet som angis i artikkelen [23], og definisjonen er som sådan mer en idé. I artikkelen beskrives det derimot en implementasjon av den generelle definisjonen som demonstrerer bruken av algoritmesystemet i form av et mer spesifikt system. I denne delen presenteres det elastiske veikartet på en tilsvarende måte, der først den generelle «idéen» beskrives og deretter den mer spesifikke implementasjonen.

2.7.1. Generell definisjon

«Elastic Roadmaps» [23], eller det «elastiske veikart», er en algoritme som viderefører PRM-tankegangen fra [12]. I motsetning til den opprinnelige PRM-algoritmen der grafen representerer konfigurasjonsrommet, så er det elastiske veikartet en representasjon av arbeidsrommet. En graf med kollisjonsfrie konfigurasjoner som noder representerer robotens arbeidsrom. Nodene, som kalles «milestones» eller milepæler, befinner seg dermed i $\mathcal{C}_{free}$. $\mathcal{C}_{free}$ beskrives altså indirekte ved hjelp av frie konfigurasjoner i arbeidsrommet. En milepæl kan betraktes som en virtuell robot, med en gitt konfigurasjon, som befinner seg i en modell av det virkelige arbeidsrommet. Kanter i grafen indikerer om roboten kan forflyttes mellom to milepæler. Når algoritmen får en planleggingsforespørsel kan mål- og startkonfigurasjonene q_g og q_i opprettes som noder i grafen. Planleggingsproblemet reduseres slik til et søk i en graf. Ønsker man å finne korteste vei mellom q_g og q_i kan man eksempelvis benytte Dijkstras korteste vei-algoritme[8].

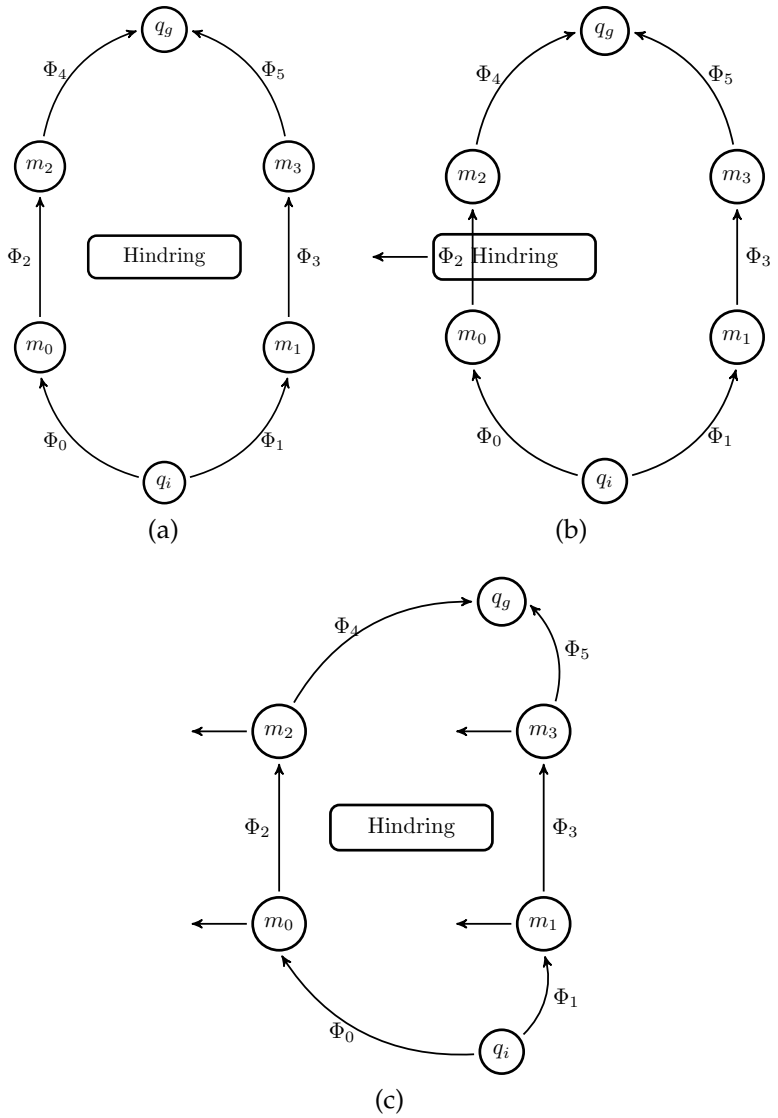
Den topologiske grafen består av et sett milepæler $M = \{m_1, \ldots, m_n\}$ og en ordnet relasjon $C : \Phi \times M \times M \rightarrow \{sann, usann\}$ som erstatter de tradisjonelle kantene i en graf. Relasjonen C er sann for to milepæler, m_i og m_j hvis en *kontroller* Φ er i stand til å bevege roboten mellom $m_i \rightarrow m_j$. Kontrolleren er i utgangspunktet ikke av en fastsatt type, men Yang og Brock [23] bruker «Task-level controllers» i sin implementasjon (se del 2.7.2). Styringen av roboten fra q_i til q_g skjer ved bruk av et sett kontrollere, der hver individuelle kontroller styrer roboten mellom sine tilknyttede milepæler. Det utformes dermed ingen konkret

plan eller konfigurasjonsrekke, men et uendelig antall potensielle veier. Dette er i sterk kontrast til de fleste andre planleggingsalgoritmer, som på en eller annen måte utformer en plan eller kurve roboten skal følge. Fremgangsmåten gjør at man unngår å bruke tid på å utforme en detaljert plan som kan bli ugyldiggjort av de dynamiske omgivelsene roboten befinner seg i. Frigjøring av planleggingskostnad muliggjør for eksempel bedre kraftregulering[4], som kan kreve oppdateringsrater opp mot 1000 Hz [23]. Uten en fast plan har man dog ingen mulighet til å optimalisere ruten roboten skal følge. Roboten kan risikere å ikke finne «åpenbare» veier til målet ved uheldig eller mangelfull plassering av milepæler. Algoritmen spesifiserer hverken Φ eller noen prosedyre for oppretting av kanter i grafen, og er i den forstand mer en idé enn en konkret algoritme.

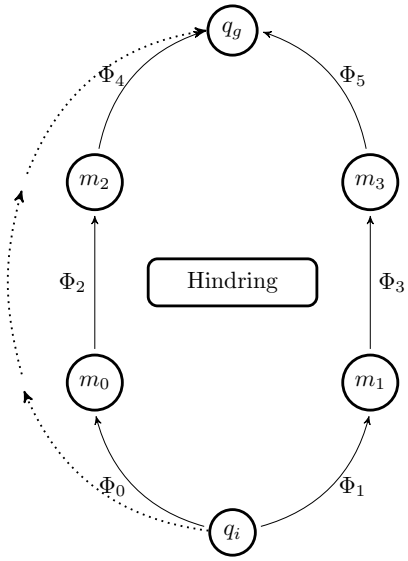
Den opprinnelige PRM-algoritmen [12] er tiltenkt bruk i statiske omgivelser. Når en milepæl eller kant legges til i grafen vil disse i utgangspunktet ikke endres. En milepæl er per definisjon kollisjonsfri, og i et konstant miljø må den aldri fjernes eller flyttes på for å tilpasse seg endringer i omgivelsene. Det elastiske veikartet er derimot tiltenkt bruk i et dynamisk miljø, som krever at milepælene endrer seg i takt med omgivelsene. Den generelle algoritmespesifikasjonen bestemmer riktignok ikke hvordan milepælene skal vedlikeholdes. Figur 2.10 viser et tenkt veikart over et arbeidsrom med en initiell konfigurasjon q_i, en målkonfigurasjon q_g og en hindring som er omringet av milepæler $M = \{m_0, \ldots, m_3\}$. Etter som hindringen flytter seg oppdateres milepælene slik at de fremdeles beskriver $\mathcal{C}_{free}$ og kantene mellom milepælene beholdes. Basert på denne virkelighetsrepresentasjonen kan planleggingsproblemet løses som vist i figur 2.11.

Den «sannsynlighetsmessige komplettheten» (se side 20), som beskriver andre samplingsbaserte algoritmers garanti for å finne en løsning på planleggingsproblemet, er i denne algoritmen forkastet. Det gis ingen garanti for at planleggingsproblemet kan løses hvis det eksisterer en løsning. Tanken er at enhver garanti som involverer et dynamisk miljø utenfor systemets kontroll uansett vil være av liten verdi. Algoritmen hevder å gjøre en bevisst nedprioritering av dette aspektet, til fordel for prosesseringskostnad og responstid.

[4]Kontroll av krefter som roboten påtrykker omgivelsene



Figur 2.10.: Illustrasjon av et elastisk veikart over arbeidsrommet. Milepæler er markert som sirkler. (a) viser en statisk hindring omringet av mileæler. I (b) flyttes hindringen mot venstre, mens milepælene oppdateres i (c).



Figur 2.11.: En planlagt rute mellom q_i og q_g demonstreres ved stiplede linjer mellom milepæler. Kontrollere $\{\Phi_0, \Phi_2, \Phi_4\}$ skal sørge for å styre roboten langs den planlagte ruten.

2.7.2. Spesifikk implementasjon

Sentralt i implementasjonen beskrevet av Yang og Brock [23] står bruken av «Task-level controllers» [19]. Dette er et i praksis en måte å kombinere ulike prioriterte hensyn til ett pådrag som kan påtrykkes roboten. De ulike hensynene som ofte kalles «tasks», oppgaver eller oppførsler, gis i form av en kunstig kraft F eller et moment Γ. Eksempelvis som en kraft rettet bort fra en nærliggende hindring eller et moment som genereres rundt et av robotens ledd når leddet nærmer seg sitt kinematiske maksimumsutslag. En kraft F_{task} som påtrykkes robotens verktøyledd og et underordnet moment Γ_0 kan kombineres slik [23]:

$$\Gamma = J_{task}(q) \cdot F^T + N_{task}^T(q) \cdot \Gamma_0 \tag{2.4}$$

der J_{task} er verktøyleddets Jacobian-matrise[5] og N_{task}^T er en projiseringsmatrise inn mot nullrommet til J_{task}. Nullromsprojiseringen skal sørge for at en underordnet kraft ikke påvirker høyere prioriterte krefter. Tilsvarende beskrives det hvordan prinsippet kan utvides til å gjelde et vilkårlig antall krefter:

$$\Gamma = \Gamma_1 + N_1^T(q) \cdot (\Gamma_2 + N_2^T(q) \cdot (\Gamma_3 + N_3^T(q) \cdot (\ldots))) \tag{2.5}$$

Prinsippet beskrives i større detalj av Sentis og Khatib [14]. Bruken og kombinasjonen av ulike hensyn på en slik måte er i tråd med oppførselsparadigmet i robotikken [6]. Etter en slik tankegang vil mange små og store *oppførsler* utgjøre en helhetlig styring av et system. De ulike kreftene som påvirker roboten kan dermed betraktes som oppførsler. Yang og Brock benytter følgende notasjon (beskrevet i [16]) for oppførselsprioritering: $\phi_j \triangleleft \phi_i$, der oppførselen ϕ_i er høyere prioritert enn ϕ_j. Med en slik notasjon kan ligning 2.5 omskrives til

$$\phi_3 \triangleleft \phi_2 \triangleleft \phi_1 \tag{2.6}$$

ϕ_n er her forbundet med Γ_n og utføres i nullrommet av ϕ_{n-1}.

For å bestemme om en milepæl kan nås fra en annen benytter Yang og Brock en enkel synlighetstest i sin implementasjon. Hvis en milepæl

[5] Beskrivelse av sammenhengende mellom hastigheter til de ulike deler av en kinematisk rekke

er «direkte synlig» fra en annen, opprettes det en kant mellom nodene i grafen. Med synlighet menes det her at en rett linje kan trekkes mellom hvert ledd i de to milepælskonfigurasjonene uten at linjene kolliderer med en registrert hindring. Hvis milepælene har direkte sikt mellom hverandre er det sannsynlig at relasjonen C for milepælene er sann. Kontrolleren som utfører bevegelsen trenger derimot ikke å følge siktlinjen mellom milepælene, men står fritt til å holde seg på avstand fra hindringer eller ta andre hensyn. Dermed kan det sies at det eksisterer et uendelig antall veier bare mellom to av milepælene i veikartet. Bruken av en slik kontroller sentralt i algoritmen utvisker i stor grad den «tradisjonelle» grensen mellom planlegging og regulering.

Opprettelsen av milepæler skjer i forbindelse med oppdagelsen av en hindring. Når en hindring er oppdaget modelleres denne som et konvekst geometrisk objekt i grafen, og denne omringes av milepæler. For å sikre at milepælene er i en gunstig avstand til den modellerte hindringen og samtidig tilfredsstiller eventuelle orienteringsbegrensninger benyttes en egen kontroller for å «dra» milepælene inn mot en god konfigurasjon. Yang og Brock benytter følgende kontroller til oppgaven:

$$\phi_{posture} \lhd \phi_{avoidance} \lhd \phi_{task} \lhd \phi_{feature} \lhd \phi_{kinematic} \lhd \phi_{collision} \tag{2.7}$$

Den høyest prioriterte oppførselen er $\phi_{collision}$, som skal unngå direkte kollisjoner med hindringer. $\phi_{kinematic}$ skal sørge for at roboten ikke overskrider sine kinematiske begrensninger, mens $\phi_{feature}$ skal holde roboten på en passende avstand fra sin tilknyttede hindring. ϕ_{task} skal overholde orienteringsbegrensninger pålagt av robotens oppgave (hvis for eksempel oppgaven er å holde et glass vann, kan ikke glasset roteres vilkårlig). $\phi_{avoidance}$ skal holde roboten unna andre hindringer og $\phi_{posture}$ bedriver «kinematisk kondisjonering»[6].

2.7.3. Utfordringer

Ser man bort fra alle utfordringer knyttet til oppdagelsen og modelleringen av arbeidsrommet og hindringer i dette er denne algoritmens vellykkethet i stor grad beroende på en god plassering av milepæler.

[6]Oppmuntrer til en god holdning.

Milepælene skal beskrive det frie konfigurasjonsrommets topologi slik at en plan kan formuleres. Med et lite antall milepæler risikerer man at det ikke finnes noen vei mellom q_i og q_g, mens et for stort antall vil være kostnadskrevende. Yang og Brock løser problemet ved å omringe alle oppdagede hindringer med milepæler (vist i figur 2.10). Løsningen er på ingen måte optimal, men man vil kunne komme seg rundt hindringen. Med en uheldig plassering av hindringer vil man kunne få oppfatningen av at roboten sniker seg langs hindringer i stedet for å gå rett mot målet. En løsning kan være å jevnlig forsøke å redusere antallet milepæler basert på et optimalitetskriterium. Eksempelvis forsøke å fjerne en tilfeldig milepæl, for deretter å vurdere det resulterende minimale spenntreet [8] over grafen eller vurdere om lengden på den planlagte ruten blir kortere.

Bestemmelsen av relasjonen $C : \Phi \times M \times M \rightarrow \{sann, usann\}$ som utgjør kantene i grafen spesifiseres ikke av algoritmen. Yang og Brock bruker i sin implementasjon en «synlighetstest» som innebærer at hvis to milepæler $\{m_0, m1\}$ har direkte sikt mellom seg så er *sannsyligvis* $C(\Phi, m_0, m_1) = sann$. Finner man ut underveis at det ikke er tilfellet planlegger man en ny rute, som i seg selv ikke er spesielt kostnadskrevende. Problemet er at roboten kan ha satt seg fast i en konfigurasjon det er kinkig å komme seg ut av, spesielt med en såpass grov bestemmelse av sammenkoblingene i grafen. Synlighetstesting er riktignok billig beregningsmessig, men ytterligere raffinering kan gi stor uttelling. En RDT-algoritme (se side 20) med en relativt grov samplingstetthet kan brukes til å bestemme $C(\Phi, m, m)$. Eventuelt i kombinasjon med en synlighetstest: $C(\Phi, m, m) = sann$ hvis både synlighetstesten og RDT-algoritmen finner en gyldig vei. Fordelen ved å bruke en RDT-algoritme vil være at den tar hensyn til robotens kinematikk.

Generelt har algoritmen preg av å være en idé, i motsetning til en tydelig spesifisert algoritme. Svært få implementasjonsdetaljer beskrives i artikkelen [23] og den beskrevne implementasjonen er gjort mot en holonom robot. Det gir stor frihet til å tilpasse algoritmen til egne behov, samt en utfordring i form av å finne gode løsninger på mangelfulle deler av algoritmebeskrivelsen.

2.8. Robot Operating System

For å tilrettelegge utvikling rundt en robotplattform finnes det en rekke ulike programvarerammeverk. Rammeverk som tar seg av kommunikasjon mellom ulike komponenter i systemet og abstraherer bort maskin- og programvaredetaljer «Robot Operating System» (ROS) [5] er et slikt rammeverk, som i tillegg til å være et kommunikasjonsrammeverk inkluderer et omfattende programvarebibliotek for «typisk» robotfunksjonalitet.

Rammeverket er i bruk på Sintefs mobile robot Seekur Jr., som denne oppgaven er rettet mot. Implementasjon av en planleggingsalgoritme på roboten må dermed forholde seg til rammeverket og rammeverkets utviklingskonvensjoner. Den mest grunnleggende integrasjonen mot rammeverket innebærer *abonnering* på tilstandsinformasjon fra roboten og *publisering* av pådrag mot robotens aktuatorer. Som nevnt i forrige avsnitt inkluderer rammeverket en del programvarebibliotek. Disse kan en algoritmeimplementasjon dra nytte av for å lette utviklingsprosessen.

2.9. Sammendrag

Kapittelet har tatt for seg problemstillinger og metodikk rundt bevegelsesplanlegging for roboter. Hovedtrekkene i kapittelet kan oppsummeres som følgende punkter:

- Overordnet planlegging: løsning av generelle problem.
- Veiplanlegging: «grov» planlegging for roboter.
- Kurveplanlegging: raffinering og nøyaktig spesifisering av veiplanleggeres resultat.
- Bevegelsesplanlegging: kombinasjon av veiplanlegging og kurveplanlegging.
- Regulering: styring av roboten i henhold til en plan.
- Det elastiske veikartet: en annerledes bevegelsesplanlegger.

3. Systemdesign

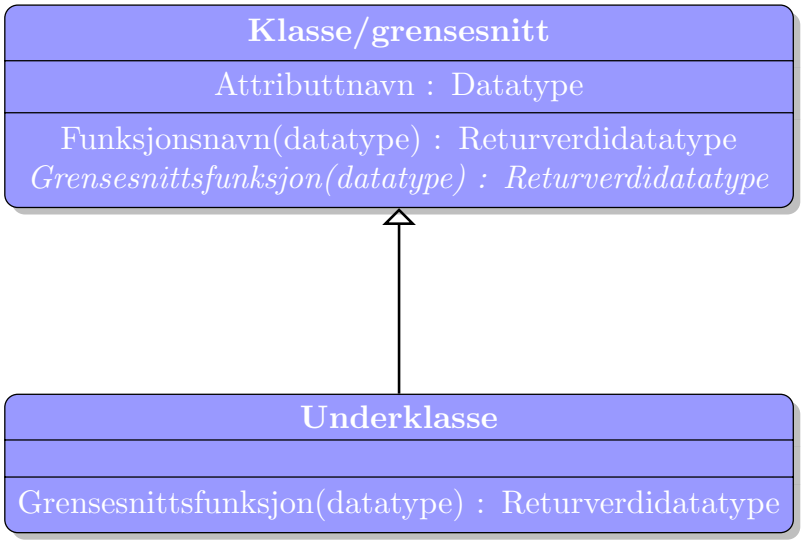
Dette kapittelet presenterer et design for bruk av «Elastic Roadmap»-algoritmerammeverket som er beskrevet i kapittel 2.7. Rammeverket formulerer en plan basert på en infrastruktur, et veikart, som vedlikeholdes for å reflektere den kjente oppfattelsen av virkeligheten. Planleggingsprosessen, veikartsvedlikeholdet samt avhengigheter som dras inn av funksjonalitetskrav fører til at designet som presenteres er et systemdesign mer enn et konsist algoritmedesign. Vedlikehold av veikartet behøver eksempelvis kollisjonsdetektering for å bestemme sammenkoblingen mellom milepæler. Kinematikken til robotplattformen Seekur Jr. med tilknyttet manipulatorarm må tas hensyn til. Visualisering av veikartet inkluderes også i designet, som et nyttig verktøy til feilsøking og testing.

Systemet designes for en objektorientert C++-implementasjon. Dette er en velprøvd designmetodikk som gir et enkelt dokumenterbart og oversiktlig system, forutsatt at metodikken brukes hensiktsmessig. Programmeringsspråket C++ tilbyr en middelvei mellom prosesseringsmessig effektivitet og enkelhet i bruk. Dessuten gir bruken av C++ en fordel når det gjelder utvalg av tredjepartsbibliotek.

3.1. Figurforklaring

Som en del av designet foreligger det flere spesifikasjoner av klasser og grensesnitt. Figur 3.1 er et eksempel på en slik spesifikasjon. Fremstillingen følger UML-standarden for klassediagram og tolkes som følgende: Hver boks angir en dataklasse eller et grensesnitt. I den øverste kolonnen er klassenavnet, i den midterste spesifiseres dataattributter, mens den nederste kolonnen angir funksjoner. Attributter følger formatet «navn : datatype», mens funksjoner inkluderer datatyper for funksjonsargument samt datatype for returverdi. Datatypen er en løs

spesifisering og representerer ikke nødvendigvis en konkret implementerbar datatype. Funksjoner i kursiv er *grensesnittfunksjoner* som ikke er implementert i klassen som angir funksjonen, men som implementeres i underklasser. Forholdet mellom grensesnitt og underklasser angis som «underklasse $-\triangleright$ grensesnitt».



Figur 3.1.: Eksempel på et UML-klassediagram.

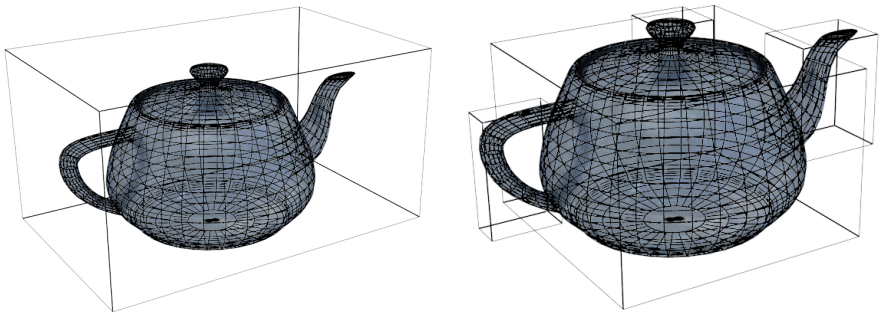
3.2. Hoveddeler

Med «hoveddeler» menes de delene av systemet som er sentrale og ikke minst interessante i forhold til systemets totale funksjonalitet. Det innebærer systemets interne verdensrepresentasjon og hvordan denne relateres til veikartsgrafen, kobling mot robotens kinematikk, samt kontroller og tilknyttede kontrolloppførsler.

3.2.1. Verdensbildet

Sentralt i systemet står algoritmens verdensrepresentasjon. Dette er en «minimumsrepresentasjon» av virkeligheten. Kraftig forenklet, men

likevel detaljert nok til at roboten kan utføre sin oppgave. Hindringer i robotens arbeidsrom modelleres fortrinnsvis som konvekse størrelser, der det enkleste er å omslutte hindringen av en kube (ofte kalt «bounding box»). Konvekse størrelser er enklere å modellere, i tillegg til at kollisjonsdetektering mellom slike er mindre kostnadskrevende enn mellom ikkekonvekse objekt. Figur 3.2 illustrerer prinsippet. En slik forenkling er et enkelt første steg for å demonstrere algoritmens prinsipielle funksjonalitet. I praksis kan en slik grov forenkling være veldig begrensende i forhold til robotens oppgave. Videre inneholder verdensrepresentasjonen alle milepæler som opprettes. Milepælene tar form av en konfigurasjon, en virtuell robot, i verdensbildet. Opprettelsen av milepæler skjer i forbindelse med inkludering av nye hindringer. Systemet tar derimot ikke stilling til hvordan hindringer oppdages. Oppdagelsen av hindringer involverer tolkning av data fra ulike sensorer og er i seg selv ingen enkel oppgave.

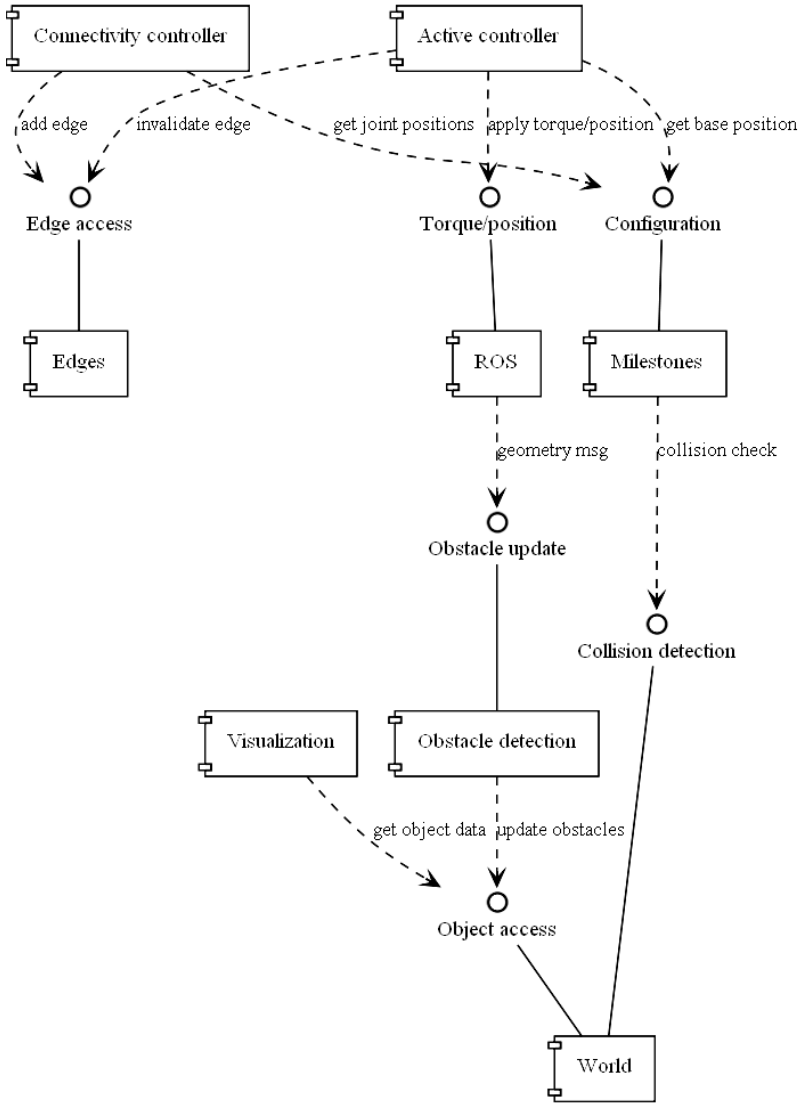


(a) (b)

Figur 3.2.: Konveks modellering av en ikkekonveks geometrisk størrelse. En ikkekonveks tekanne omsluttes i (a) av én kube, mens (b) viser en mer nøyaktig tilnærming.

3.2.2. Grafskomponenter

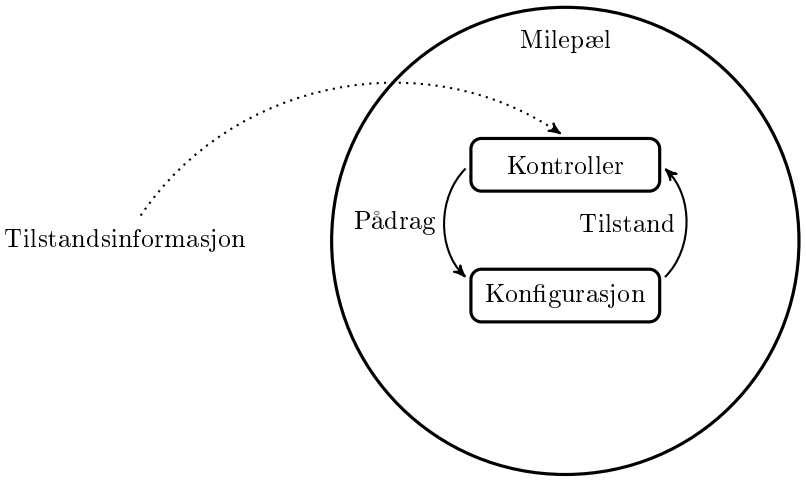
Grafen er datastrukturen som milepæler og kanter organiseres i, som beskrevet i kapittel 2.7. Datastrukturen opprettes ikke eksplisitt som



Figur 3.3.: Systemets hovedkomponenter og informasjonsflyt mellom disse. Firkantene representerer komponenter og sirklene er grensesnitt mot komponentene som sirklene har en heltrukken linje mot. Stiplede linjer angir koblinger mellom komponenter via et grensesnitt.

en del av designet, men vil være en del av totalsystemet i den forstand at systemet inneholder alle grafskomponenter. Den viktigste komponenten i grafen er milepælen. Hver milepæl inneholder en konfigurasjon, samt en kontroller som kontinuerlig oppdaterer konfigurasjonen. Milepælene er først og fremst tilknyttet hindringer som legges til i verdensbildet. Kontrollerens oppgave er å holde konfigurasjonen på en passe avstand til hindringen, samt sørge for at eventuelle andre krav overholdes. Kontrolleren omtales i større detalj i del 3.2.4. Det grunnleggende oppsettet innad i en milepæl er illustrert i figur 3.4.

Kantene i grafen er en enkel datastruktur som inneholder referanser til milepælene den forbinder, samt til kontrolleren som brukes når roboten beveger seg langs kanten. I tillegg har kanten en statusfelt som angir om den er gyldig eller ikke. En gyldig kant ugyldiggjøres hvis en hindring blokkerer sikten mellom milepælene som kanten forbinder.



Figur 3.4.: Informasjonsflyt innad i en milepæl. Kontrolleren oppdaterer milepælens konfigurasjon basert på nåværende konfigurasjon samt tilstandsinformasjon fra systemet. Eksempelvis informasjon om hvor hindringer befinner seg.

3.2.3. Robot og kinematikk

For å gi en abstrahering fra detaljer rundt roboten defineres det er generisk grensesnitt for robotspesifikk funksjonalitet. «RobotModel», som vist i figur 3.12, definerer funksjoner for oppsett av Jacobian-matriser [17, kap. 4] og for å påtrykke robotmodellen en konfigurasjonsvektor. Robotmodellen vil være robotens «ansikt» i visualisering av den virtuelle verdensrepresentasjonen. Figuren viser videre hvordan klassen «SeekurJrModel» implementerer grensesnittet og tilbyr en implementasjon spesifikk for Seekur Jr. med tilknyttet robotarm. Roboten er illustrert i figur 3.5.

Roboten med tilknyttet manipulatorarm har en rekke ulike koordinatsystem som tillegges de ulike leddene i systemet. For å beskrive sammenhengen mellom de ulike koordinatsystemene brukes homogene transformasjonsmatriser av formen

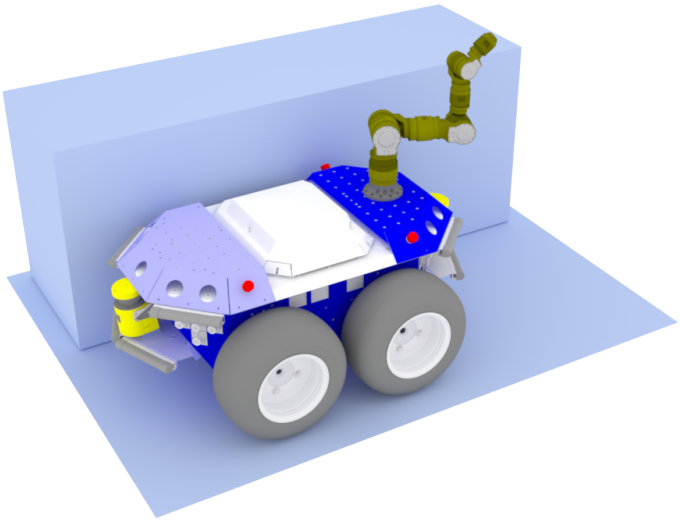
$$H_0^1 = \begin{bmatrix} R_0^1 & d \\ 0 & 1 \end{bmatrix}$$

der R_0^1 er en matrise som angir rotasjonen mellom koordinatsystemene $0 \rightarrow 1$ og d er en vektor mellom origo i koordinatsystemene [17, kap. 2.7]. For Seekur Jr. med manipulatorarm er det åtte ulike koordinatsystem. Ett lagt til basens rotasjonssentrum, og sju lagt til de sju leddene i manipulatorarmen som vist i 3.7. Sammenhengen mellom robotens basekoordinatsystem, «b», og dens verktøyleddsystem, «e», er gitt av følgende transformasjonsmatriser:

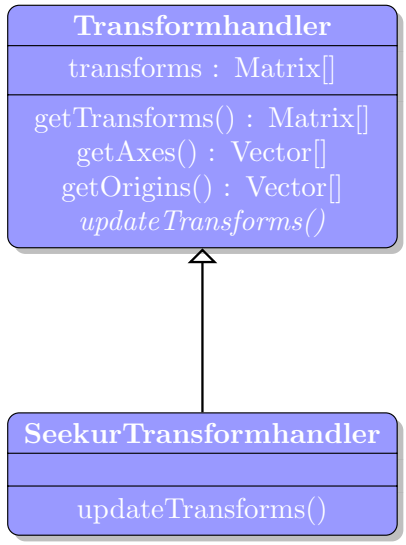
$$H_b^e = H_b^{j_0} \cdot H_{j_0}^{j_1} \ldots H_{j_4}^{j_5} \cdot H_{j_5}^e \tag{3.1}$$

Koordinatsystemene til armens sju ledd er navngitt $j_0, \ldots, j_5, j_e$. Transformasjonene er et resultat av robotens konfigurasjonsvektor q:

$$q = \begin{bmatrix} x \\ z \\ \theta_b \\ \theta_0 \\ \theta_1 \\ \vdots \\ \theta_e \end{bmatrix} \tag{3.2}$$



Figur 3.5.: Illustrasjon av Seekur Jr. robotplattform med påmontert Schunk LWA3 manipulatorarm. 3d-modellene tilhører Schunk GmbH (manipulator) og MobileRobots (plattform).



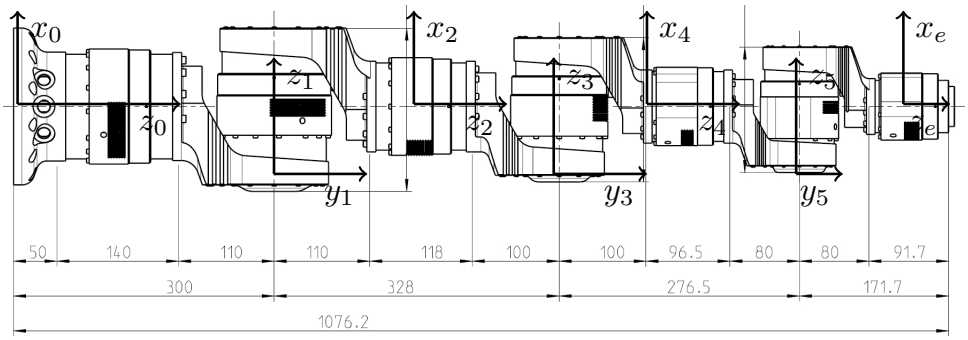
Figur 3.6.: Grensesnitt for håndtering av robotspesifikke transformasjonsmatriser.

$\theta_0, \ldots, \theta_5, \theta_e$ er rotasjonsvinkler[1] rundt manipulatorarmens tilhørende z-akse, som vist i figur 3.7. Figurens nedre del viser robotplattformen og dens koordinatsystem. Merk at z-aksen peker mot robotens fremre del.

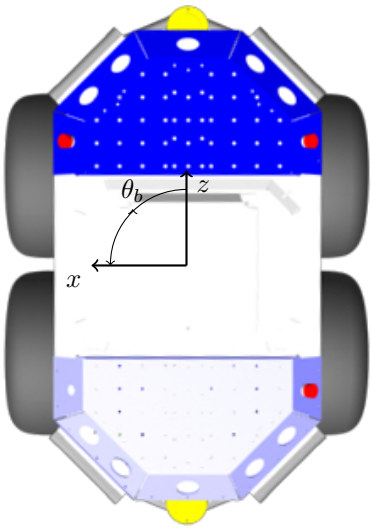
For å forenklere utregningen av transformasjonsmatrisene defineres det et grensesnitt, «Transformhandler», som vist i figur 3.6. «SeekurJrTransformhandler» implementerer grensesnittet og funksjonen *updateTransforms()* som står for utregningen av transformasjonsmatrisene basert på en konfigurasjonsvektor.

Roboten er et ikkeholonomisk system, som innebærer at systemets tilstand beskrives av både q og $\dot{q}$ [21, kap. 1.3.6]. Roboten kan rotere om basens midtpunkt, men manipulatorarmen er ikke plassert langs basens rotasjonssentrum. Ikkeholonomien og plasseringen av manipulatorarmen er noe dette designet ikke tar hensyn til. Krefter og moment rundt basens koordinatsystem og rundt manipulatorarmens første ledd oversettes direkte til et rotasjonsmoment og translasjonskraft mot robotens

[1]Rotasjon og akseplassering følger høyrehåndsregelen.



(a)



(b)

Figur 3.7.: LWA3 7-ledds manipulatorarm (a) og Seekur Jr. sett ovenifra (b). Bilder tilhører henholdsvis Schunk GmbH og MobileRobots.

base. I praksis medfører det at roboten ikke nødvendigvis klarer å nå en *fullstendig* målkonfigurasjon ($q_{robot} = q_G$), men kun oppfyller verktøyleddets mål. Figur 3.8 viser hvordan en kraft påtrykt armens første ledd oversettes til bevegelse av robotbasen. Rotasjonskraft og translasjonskraft blir henholdsvis $F_{rot} = F \cdot \sin(\theta_F)$ og $F_{trans} = F \cdot \cos(\theta_F)$.

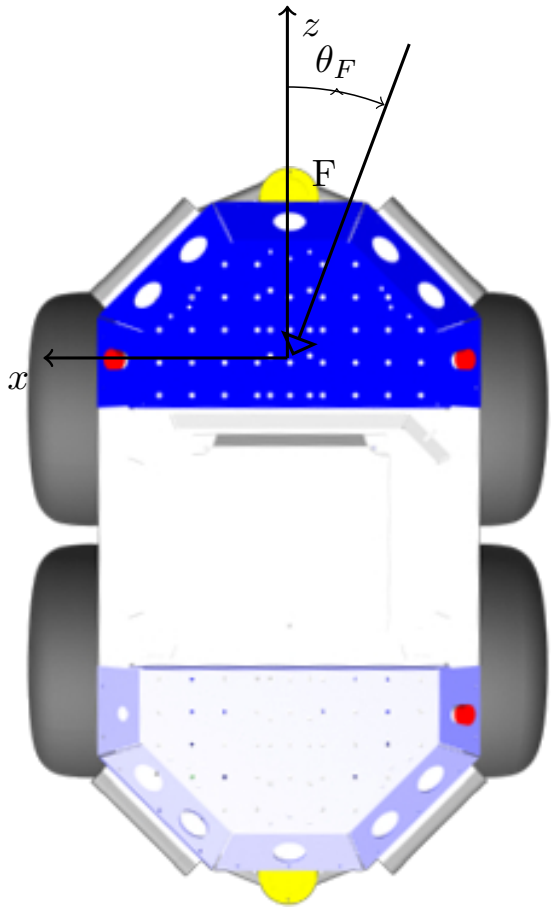
Det viser seg dessuten at robotens rotasjonssentrum ikke er konsistent. Med påmontert manipulatorarm flyttes rotasjonssentrum fra basens midtpunkt fremover i retning manipulatorarmen. Tilsvarende vil rotasjonssentrum endre seg når armen endrer konfigurasjon og hvis noe løftes opp av armen. Dette aspektet vil være gjenstand for fremtidig arbeid og tas ikke hensyn til i dette designet.

3.2.4. Kontroller

Kontrollerens oppgave er å gi et pådrag mot en konfigurasjon. Pådraget kan være en kraft eller moment, og eventuelt integreres opp mot en hastighet eller en posisjon. Som et moment vil pådraget u kunne skrives som

$$u = \tau = \ddot{q} \tag{3.3}$$

der q er konfigurasjonsvektoren som beskriver robotens posisjon og vinkelutslag. Som beskrevet i del 2.7 bestemmes pådraget basert på et sett med prioriterte underoppførsler. En underordnet oppførsel projiseres inn mot nullrommet til en høyere prioritert oppførsel, som beskrevet i del 2.1.2 og [10]. Kontrollobjektets oppgave er å kombinere pådragene fra et sett med oppførsler underlagt kontrolleren. Dette gjøres ved å definere et felles grensesnitt for alle typer oppførsler, der hver oppførsel gis fritt spillerom til å implementeres på en hensiktsmessig måte. Figur 3.9 viser grensesnittet «Behaviour» med en funksjonsspesifikasjon *calculateForce()*, som **må** implementeres av alle underklasser. Underklassene i figuren er basert på kontrolloppsettet beskrevet i Yang og Brocks implementasjon (se del 2.7) og illustrerer hvordan grensesnittet kan brukes. Underklassene krever ulik informasjon for å beregne sitt pådrag, og vil dermed variere i implementasjon. «CollisionBehaviour» vil for eksempel trenge informasjon om hvor eventuelle hindringer befinner seg, mens «KinematicBehaviour» må vite hva som er



Figur 3.8.: En kraft F påtrykt manipulatorarmens første ledd oversettes til et rotasjonsmoment og en translasjonskraft. Robotbildet tilhører MobileRobots.

robotens kinematiske begrensninger og hvor langt unna disse roboten befinner seg. Underoppførslene kan konfigureres til å kalkulere en kraft etter en gitt frekvens, eller etter en tilknyttet kontrollers forespørsel. Kontrolleren konfigureres til å «samle inn» krefter fra underoppførsler og kalkulere pådragsverdier etter en gitt frekvens. Ideelt sett bør hver oppførsel kjøre på en frekvens som tilfredsstiller oppførselens oppgave, men etter dette designet har ikke oppførselen mulighet til å påvirke kontrollerens oppdateringsfrekvens. Kontrolleren må derfor ha en frekvens som tilfredsstiller oppførselens med høyest frekvensbehov.

3.2.5. Konkret oppførselsdesign

Oppførselsgrensesnittet som er vist i figur 3.9 demonstrerer hvordan ulike oppførsler kan enes under et felles grensesnitt. For å demonstrere algoritmen designes to av oppførslene for implementasjon: *GlobalBehaviour* og *AvoidanceBehaviour*.

GlobalBehaviour skal sørge for å påtrykke roboten en kraft som forflytter den mot en målkonfigurasjon. Basert på et enkelt potensialtfelt gir den en kraftvektor som funksjon av avstanden til en målkonfigurasjon. Potensialfeltet U består av ett tiltrekningspunkt:

$$U_i(q) = \frac{1}{2}\zeta_i \|o_i(q) - o_i(q_g)\|^2 \tag{3.4}$$

der $o_i(q)$ angir posisjonene til robotens ledd basert på robotens nåværende konfigurasjon q, mens $o_i(q_g)$ er tilsvarende posisjoner for målkonfigurasjonen q_g. Parameteren ζ justerer feltets styrke. U_i er dermed feltet som påvirker koordinatsystem i på roboten. Kraften F_i er gitt av

$$F_i(q) = -\nabla U_i(q) = -\zeta_i(o_i(q) - o_i(q_g)) \tag{3.5}$$

Fordi potensialtfeltet kun består av ett tiltrekningspunkt (og ingen frastøtningspunkter) oppstår aldri det «lokale minimumsproblemet» som er beskrevet i del 2.6.4.

AvoidanceBehaviour gir en kraftvektor mot roboten som funksjon av avstanden til *nærmeste* hindring. Den består av et enkelt potensialtfelt

U med ett frastøtningspunkt:

$$U_i = \begin{cases} \frac{1}{2}\eta_i(\frac{1}{\rho(o_i(q))} - \frac{1}{\rho_0})^2 & ,\rho(o_i(q)) \leq \rho_0 \\ 0 & ,\rho(o_i(q)) > \rho_0 \end{cases} \tag{3.6}$$

der $\rho(o_i(q))$ er avstanden mellom robotledd i og nærmeste hindring. ρ_0 angir minimumsavstanden for kraftpåvirkning fra en hindring og η er en justeringsparameter. Kraften F blir dermed

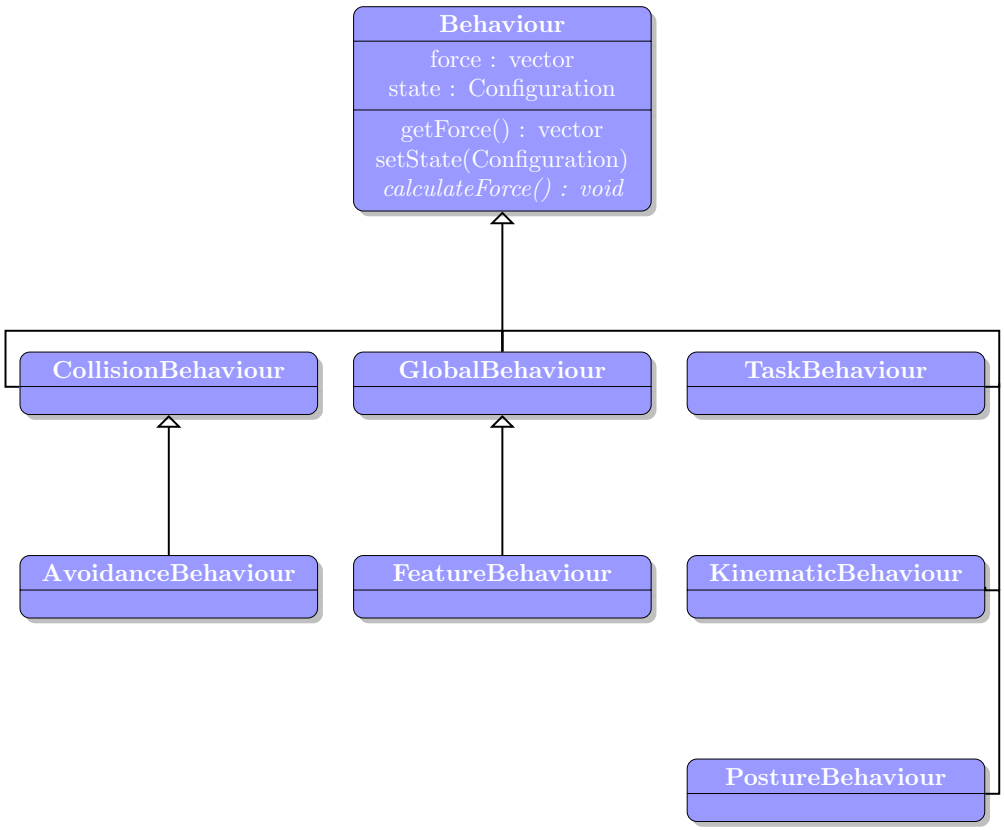
$$F_i = \begin{cases} \eta_i(\frac{1}{\rho(o_i(q))} - \frac{1}{\rho_0})\frac{1}{\rho^2(o_i(q))} & ,\rho(o_i(q)) \leq \rho_0 \\ 0 & ,\rho(o_i(q)) > \rho_0 \end{cases} \tag{3.7}$$

Oppsett av potensialfunksjoner er beskrevet av Spong et al. [17, kap. 5.2].

3.2.6. Hovedkomponenter

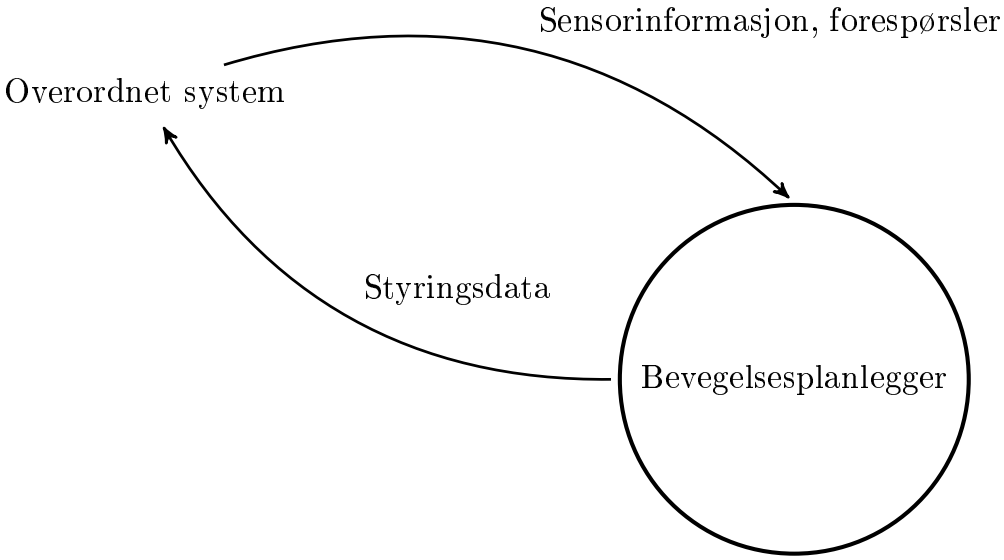
Systemet deles inn i hovedkomponenter som vist i figur 3.3. Nederst i figuren vises verdensrepresentasjonen «World» som tilbyr grensesnitt for kollisjonsdetektering og håndtering av objektmodeller. Øverst i figuren vises «Connectivity controller»-komponenten. Denne er ansvarlig for å opprette og vedlikeholde kanter i grafen. Når en ny milepæl legges til i grafen kan denne ha direkte sikt til andre milepæler. Hvis det er tilfellet legges det til kanter mellom den nyopprettede milepælen og de andre milepælene. Tilsvarende kan opprettelse eller endring av en hindring føre til at eksisterende siktlinjer mellom milepæler blir blokkert. «Connectivity controller» skal da sørge for å ugyldiggjøre kanter som nå er brutt. Komponenten utfører denne oppgaven med en frekvens på rundt $0,5 - 2Hz$ (eller høyere om nødvendig), avhengig av hvor endringsutsatt miljøet rundt roboten er.

Når en planleggingsforespørsel gis til algoritmen vil den resultere i et sett med kanter som angir en plan. Settet med kanter og tilhørende kontrollere skal sørge for å flytte roboten fra dens opprinnelige konfigurasjon og til målkonfigurasjonen via milepælkonfigurasjoner langs veien. «Active controller»-komponenten i figur 3.3 refererer til den kontrolleren som fører roboten mot neste milepæl langs veien. Kontrolleren gir et pådrag mot «ROS»-komponenten. «ROS» kan betraktes som den fysiske roboten, selv om komponenten kan innbefatte et



Figur 3.9.: Klassediagram som viser grensesnittet «Behaviour» med implementasjonsklasser. Alle funksjoner skrevet i kursiv **må** implementeres av alle klasser som implementerer grensesnittet.

vilkårlig antall robotrelaterte systemer. Informasjon om hindringer og eventuelt relevante hendelser skjer via denne komponenten.



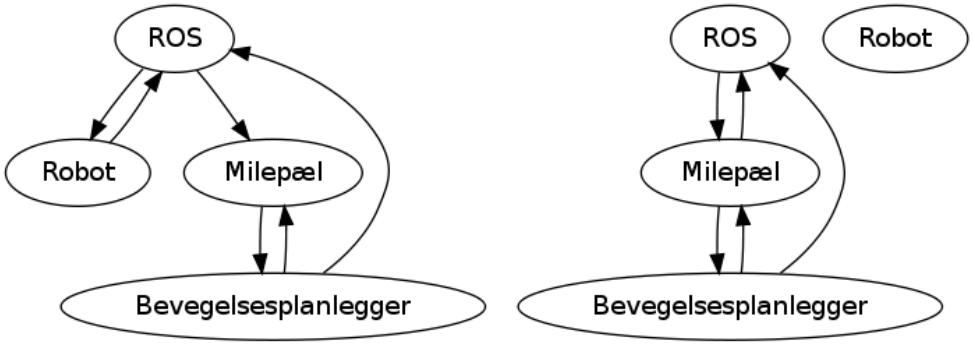
Figur 3.10.: Planleggingsalgoritmen sett utenifra. Sensorinformasjon bidrar til å gi algoritmen et grunnleggende verdensbilde. Med sensorinformasjon menes «sanset virkelighetsinntrykk», ikke rådata fra en sensor. Planleggingsforespørsler fra et overordnet system resulterer i styringsdata sendt fra algoritmen til roboten.

Figur 3.10 viser den grunnleggende informasjonsflyten mellom bevegelsesplanleggeren og et overordnet system. Planleggingsforespørsler kan gis av for eksempel en menneskelig operatør eller en annen algoritme som er tettere knyttet mot robotens oppgave. Når algoritmen får beskjed om at en hindring er oppdaget vil denne modelleres som en «bounding box» og milepæler opprettes rundt hindringen.

3.2.7. Simulering og visualisering

Den fysiske roboten representeres som en milepæl i grafen. Milepælens konfigurasjon skal direkte representere roboten ved hjelp av tilstandsin-

formasjon mottatt via Ros. Enkel simulering gjøres ved at milepælen som representerer roboten sender sin tilstandsinformasjon over Ros og simulerer dermed den fysiske roboten. Simuleringen er dermed en direkte kobling mellom pådrag og tilstand, en antakelse om «perfekt regulering». Verdien av en så enkel simulering er først og fremst i form av visualisering. Figur 3.11 viser hvordan en milepæl erstatter roboten i en simulering.

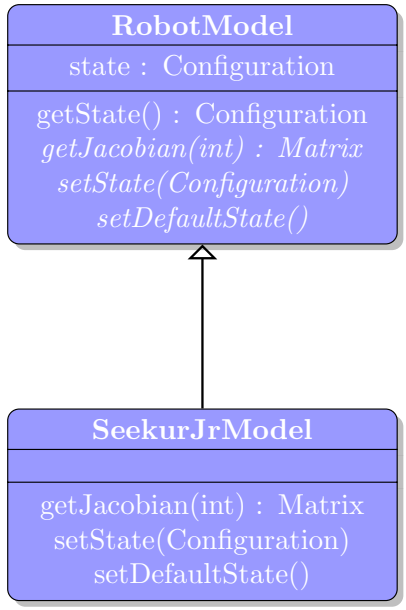


Figur 3.11.: Informasjonsflyt blant utvalgte komponenter. Bevegelsesplanleggeren gir pådragsinformasjon til roboten via Ros. Milepælen representerer roboten og mottar tilstandsinformasjon fra roboten via Ros. I (b) simuleres roboten, og milepælen som representerer roboten sender tilstandsinformasjon mot Ros.

3.3. Planleggingsdelen

Skillet mellom algoritme og system er til en viss grad utvisket gjennom designprosessen. Basert på et persistent veikart, en *graf* med milepæler, kanter og kontrollere kan man betrakte algoritmen som prosessen med å finne og følge en plan gjennom veikartet. Systemet blir i det perspektivet infrastrukturen som algoritmen benytter.

For å formulere en plan benyttes Dijkstras korteste vei-algoritme [8] som finner den korteste veien mellom to noder i en graf. Hver kant i grafen gis et kostnadsestimat som gjenspeiler kantens verdi. Enten i form av avstand mellom milepælene som kanten sammenkobler, eller basert på et annet optimalitetskriterium. Kostnaden til enhver kant settes i utgangspunktet til «1», slik at søkealgoritmen vil foretrekke en plan med færrest mulig kanter. Hvis kanten roboten følger blir ugyldiggjort må en ny plan formuleres. Tilsvarende må en ny plan formuleres hvis roboten har utilstrekkelig fremgang mot milepælen kanten er rettet mot. Fordi det er et relativt usikkert kriterium for sammenkoblinger i grafen er det sannsynlig at roboten ikke alltid klarer å følge en kant og planen må omformuleres.



Figur 3.12.: Grensesnittet «RobotModel». Grensesnittet definerer nødvendige operasjoner, men forholder seg ikke til hvordan funksjonaliteten implementeres. Funksjoner i kursiv **må** implementeres av alle underklasser. «SeekurJrModel» er implementasjonen av en Seekur Jr. med tilknyttet LWA3 manipulatorarm.

3.4. Avvik fra algoritmespesifikasjon

Relasjonen $C : \Phi \times M \times M \rightarrow \{sann, usann\}$ som erstatter de tradisjonelle kantene i grafen har i algoritmespesifikasjonen (se del 2.7) en unik kontroller Φ tilknyttet hver kant i grafen. For to milepæler, m_i og m_j, vil tilhørende Φ styre bevegelsen mellom milepælene. I praksis vil de ulike kontrollerene skilles kun av målkonfigurasjonen de styrer mot. Hindringsunngåelse, kinematiske begrensninger, orienteringsbegrensninger og tilsvarende vil være likt for alle Φ. I et effektivitetsøyemed gir det dermed mening å benytte samme Φ for alle $C(M, M)$ og kun endre på kontrollerens målkonfigurasjon. Kontrolleren Φ kan dermed betraktes som en funksjon, der $\Phi(m_j)$ gir et pådrag som flytter roboten fra dens nåværende konfigurasjon mot konfigurasjonen til milepælen m_j.

Den virkelige roboten representeres i grafen som en milepæl. Dette innebærer at enhver kant $C(\Phi, m_j, m_i)$ har en kontroller som ikke fører roboten mellom $m_j \rightarrow m_i$, men heller mellom $m_{robot} \rightarrow m_i$.

3.5. Kobling mot Ros

Den fysiske roboten er tilknyttet planleggingssystemet via Ros og kommunikasjonstjenesten som Ros tilbyr. Robotens nåværende konfigurasjon kommuniseres til planleggingsystemet, som sender pådragsdata for motorhastigheter og vinkelutslag tilbake til roboten. Figur 3.11 viser den grunnleggende koblingen mot Ros. Dette er «minimumsintegrasjonen» mot Ros og den fysiske roboten. For tettere integrasjon, som kan på lang sikt være arbeidsbesparende, kan man blant annet benytte det omfattende tredjepartsbiblioteket eller kraftige verktøy for skriptbasert konfigurasjon som Ros tilbyr. For eksempel kan man se for seg at den fysiske roboten beskrives av en generell skriptfil som benyttes av algoritmen i dens utregninger. Da vil det ikke bare være enklere å oppdatere planleggeren etter som det fysiske oppsettet endres, men også enklere å konfigurere systemet for bruk mot en ny robot. Tettere integrasjon mot Ros vil bety at algoritmeimplementasjonen vil være utsatt for endringer i Ros-rammeverket. Dessuten vil det være mer arbeidskrevende å benytte algoritmen til andre formål uten-

for rammeverket.

3.6. Sammendrag

Kapittelet har presentert et systemdesign for implementasjon av «Elastic Roadmaps» med følgende hovedtrekk:

- En intern verdensrepresentasjon.
- En grafdatastruktur med milepæler og kanter.
- Milepæler som virtuelle roboter.
- Den virtuelle roboten beskrives kinematisk.
- En samling kontrollere står for styringen av virtuelle og fysiske roboter.
- Flere underoppførsler utgjør en kontroller.
- Planlegging gjøres som et korteste-vei-søk blant milepæler og kanter.
- Kommunikasjon med den fysiske roboten og eventuelt andre systemer skjer via Ros.

4. Systemimplementasjon

I dette kapittelet omtales implementasjonen av designet utformet i kapittel 3. Tredjepartsbibliotek som bygger opp «kjernefunksjonaliteten» er vektlagt i kapittelet, samt mangler i forhold til designet.

Figurene 5.1, 5.2, 5.3 og 5.4 (i kapittel 5) viser systemets grafiske ansikt.

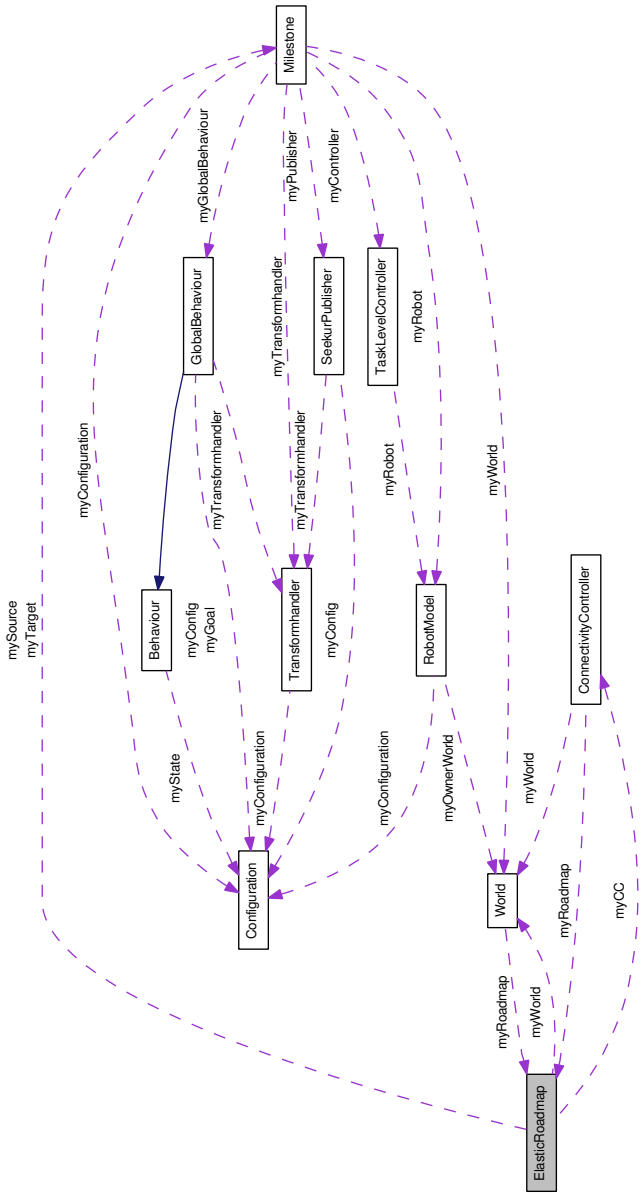
4.1. Tredjepartsbibliotek

Flere tredjepartsbibliotek er benyttet under implementasjonen av systemdesignet. Bruken av disse gir betydelig arbeidsbesparing, men det kreves en innsats for å sette seg inn i bruken av bibliotekene. Sentralt i både visualisering og kollisjonsdetektering står fysikkrammeverket «Bullet physics», som også har vært det mest arbeidskrevende biblioteket benyttet under implementasjonen.

4.1.1. Kollisjonsdetektering

Opprettelse av kanter i grafen skjer hvis to milepæler har direkte sikt mellom seg. Rette linjer trekkes mellom hvert ledd i de to konfigurasjonene og det kontrolleres om linjene skjærer en registrert hindring. En slik kontroll omtales som kollisjonsdetektering og kan utføres av ulike programvareimplementasjoner. Deriblant det åpne fysikkrammeverket «Bullet physics» [1], som er lisensiert under den frie programvarelisensen «zlib». Rammeverket er først og fremst en fysikksimulator, men inkluderer en essensiell og høyst effektiv kollisjonsdetekteringsmodul.

Sentralt i Bullet står verdensobjektet som har referanser til alle geometriske objekt. Hindringer og milepæler legges til som geometri i



Figur 4.1.: Klasseoversikt generert av dokumenteringssystemet Doxygen. Sammenhengen mellom klasser er angitt med stiplede linjer. En klasse har en eller flere referanser til objekt av klasser som det går piler mot.

verdensobjektet. Verdensobjektet i Bullet utgjør dermed «World» komponenten vist i figur 3.3 på side 37.

Milepæler representeres i verdensobjektet som en virtuell robot. En klasse som implementerer «RobotModel»-grensesnittet (vist i figur 3.12) er derfor nødt til å konstruere en geometrisk modell av roboten den representerer og legge denne til i verdensobjektet.

Hindringer representeres i denne implementasjonen som enkle kuber («bounding box»). Bullet har riktignok mulighet til å behandle kompleks geometri, noe som gir utvidbarhet mot fremtidig funksjonalitet.

Fordi Bullet er et fysikkrammeverk kan simulering på sikt tilføres mye funksjonalitet i form av dynamikk. En umiddelbar fordel med Bullet er at kollisjoner kan inngå i simulering av systemet. Når en simulert robot eksempelvis kolliderer med en hindring vil roboten visuelt kollidere, i motsetning til at den går i ett med hindringen.

4.1.2. Visualisering

Visualisering av verdensrepresentasjonen gjøres via verdensobjektet i Bullet og grafikkbiblioteket OpenGL [4]. Inkludert i Bullet er et rammeverk for enkel visualisering ved hjelp av OpenGL. Dette benyttes for å gi en visuell oversikt over milepæler og hindringer i grafen.

4.1.3. Lineær algebra

Ulike deler av systemet har behov for å utføre matriseberegninger. Spesielt kontrolleren og tilhørende oppførselskomponenter har et omfattende matematisk grunnlag og bør kunne utføre beregninger effektivt. Matrisebiblioteket «Eigen» er et fritt tilgjengelig programvarebibliotek for lineær algebra som er lisensiert under «LGPL3+». Eigen hevdes å være raskt, allsidig, pålitelig og elegant [3].

4.2. Tilknytning til Ros

Programvareimplementasjonen organiseres som en *pakke* under Roshierarkiet. I praksis betyr det at kildekoden følges av en manifestfil

skrevet i et XML-format som beskriver pakkens funksjon samt avhengigheter til andre Ros-pakker. Manifestfilen brukes først og fremst til kompilering av pakken og lenking opp mot andre pakker. Kompilering av Ros-pakker fasiliteres av byggesystemet CMake, som også benyttes til å kompilere det elastiske veikartssystemet. Kombinasjonen av manifestbeskrivelse og CMake er et kraftig verktøy som gjør det svært enkelt å benytte funksjonalitet som finnes i eksisterende Ros-pakker. En oversikt over tilgjengelige Ros-pakker finnes i [5]. Blant annet er Bullet og Eigen implementert som Ros-pakker og bruk av bibliotekene skjer via Ros.

4.3. Avvik fra design

4.3.1. Mangler

I del 3.2.4 omtales designet av kontrolleren som setter pådrag mot de ulike aktuatorene på roboten. Implementasjonen av denne er mangelfull. I henhold til designet skal den kombinere kraftbidrag fra et sett underoppførsler og gi ett samlet pådrag som reflekterer prioriteten til de ulike underoppførslene. Arbeidet med implementasjon av denne komponenten var mer omfattende enn forventet og av tidsmessige hensyn kunne ikke komponenten ferdigstilles. Kontrollerkomponenten er altså mangelfull og er kun i stand til å uttrykke pådrag fra én underoppførsel. Kontrolleren betraktes som kjernefunksjonalitet i selve algoritmeimplementasjonen og ferdigstilling av komponenten vil være en viktig del av det fremtidige arbeidet med systemet. Testing og simulering av nåværende kontroller skjer med enkeltvise oppførsler.

4.3.2. Avvik fra robotoppsett

Implementasjonen benytter en modell av Seekur Jr. med tilknyttet manipulatorarm. De implementerte størrelsene av robot og arm er samstemte med størrelsene beskrevet i dokumentasjonen til hvert av systemene. Den homogene transformasjonsmatrisen som angir sammenhengen mellom robotbasen (Seekur Jr.) og manipulatorarmens første

ledd avviker derimot fra den nåværende fysiske monteringen av manipulatorarmen på roboten. Dette har liten betydning i forhold til simulering av systemet, men vil kreve et mer nøyaktig oppsett av transformasjonsmatrisen når planleggingsystemet skal brukes på den fysiske roboten. Transformasjonsmatrisen det er snakk om er $H_b^{j_0}$, som er vist i ligning 3.1 på side 39.

4.4. Detaljert implementasjonsdokumentasjon

Kodebasen som implementasjonsprosessen har resultert i er dokumentert gjennom dokumentasjonssystemet Doxygen [2]. Doxygen genererer tekstlig og billedlig dokumentasjon som angir sammenhenger mellom klasser, attributter og funksjoner basert på definisjoner og kommentarer i koden. Dette er et kraftig hjelpemiddel som kan benyttes for å forstå og videreutvikle systemet. Et eksempel på et Doxygen-generert diagram er figur 4.1, som gir en oversikt over klassene som utgjør systemet. Både dokumentasjon og kode er å finne i vedlegg A.

4.5. Erfaringer

Bruken av Bullet som kollisjonsdetekteringssystem har en del fordeler og ulemper. Bullet er et relativt omfattende programvarebibliotek, og krever av den grunn betydelig innsats for å kunne benyttes hensiktsmessig. Biblioteket har en del funksjonalitet som ikke er til umiddelbar nytte i implementasjonen av den elastiske veikartsalgoritmen. Derimot gir bruken av biblioteket muligheter for fremtidig utnyttelse og utvidelse av veikartssystemet. I dette tilfellet betyr også bredere funksjonalitet et bredere publikum, som gir økt vedlikehold av et åpent system som Bullet.

Matrisebiblioteket «Eigen» har etter implementasjonsprosessen vist seg å leve opp til sitt løfte om å være allsidig, enkel og elegant [3].

5. Testoppsett

Dette kapittelet tar for seg testmetodikk og de konkrete mulighetene implementasjonen har for oppsett av testscenarioer. Test av implementasjonen består av et sett simulerte testscenarioer som beskrives i del 5.2. Resultat fra simuleringen presenteres i kapittel 6.

5.1. Simulering

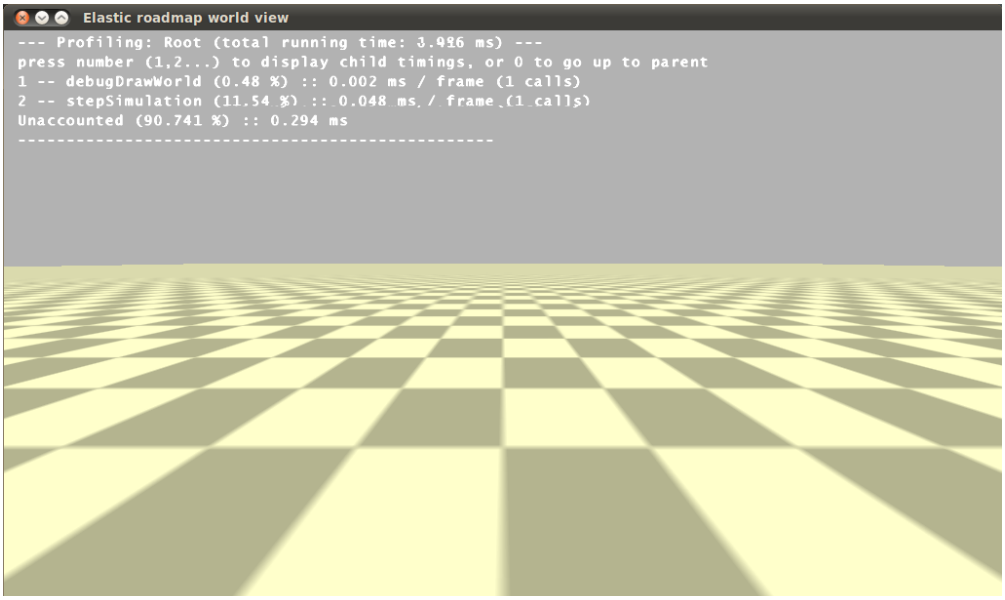
Simulering av systemet skjer ved hjelp av følgende operasjoner:

- Opprettelse av en robotsimulerende milepæl.
- Spesifisering av en målkonfigurasjon (q_G) for roboten.
- Opprettelse av «bounding box»-type hindringer.
- Flytting/fjerning av hindringer.

Den grafiske fremstillingen av systemet og grunnlaget for testscenarioene er vist i figurene 5.1, 5.2, 5.3 og 5.4. Figur 5.1 viser en tom virtuell verden med et bakkeplan. Merk at dette bakkeplanet kun er ment som en visuell «bakgrunn», og er ingen hindring for å representere mer kompleks arbeidsromsgeometri. I figur 5.2 vises en enkelt milepæl. Mileælen kan enten være en vilkårlig milepæl i veikartet, eller den kan representere den fysiske robotens konfigurasjon. Figur 5.3 viser en «bounding box» som omslutter en tenkt, vilkårlig kompleks hindring, og et sett med milepæler som omringer hindringen. Hvis hindringens posisjon endres er det de fire milepælenes oppgave å «følge etter» hindringen og opprettholde en gyldig konfigurasjon. I figur 5.4 er det i tillegg til en hindring med tilknyttede milepæler en simulert robot og en målkonfigurasjon. Målkonfigurasjonen opprettes

i likhet med den simulerte roboten som en milepæl. Målet med planleggingsprosessen er altså at robotmilepælen skal tilsynelatende bli ett med målmilepælen.

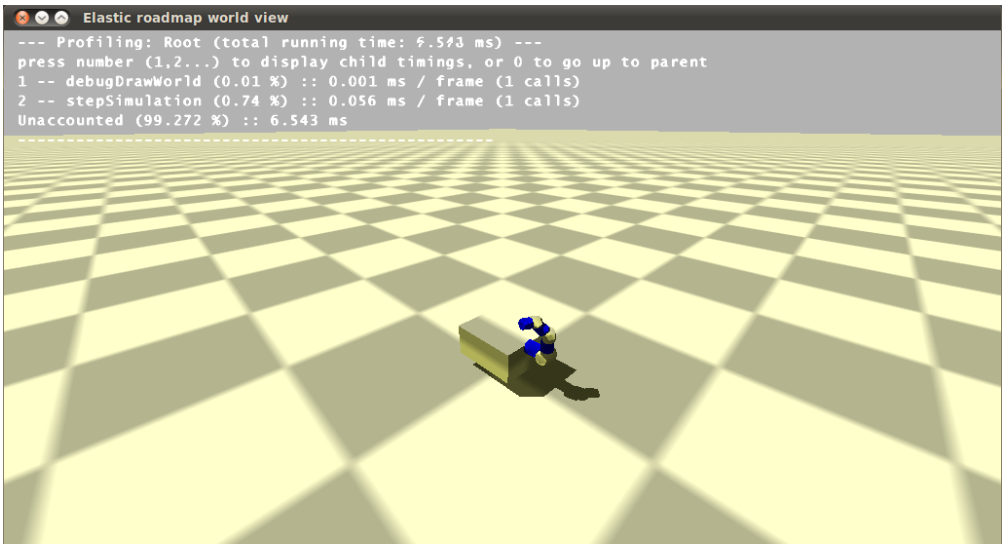
Som nevnt i kapittel 3.2 simuleres roboten ved en direkte kobling mellom pådrag og tilstand. Simuleringen er altså en antakelse om «perfekt regulering» av roboten, og må ikke tolkes som en simulering av robotens dynamikk.



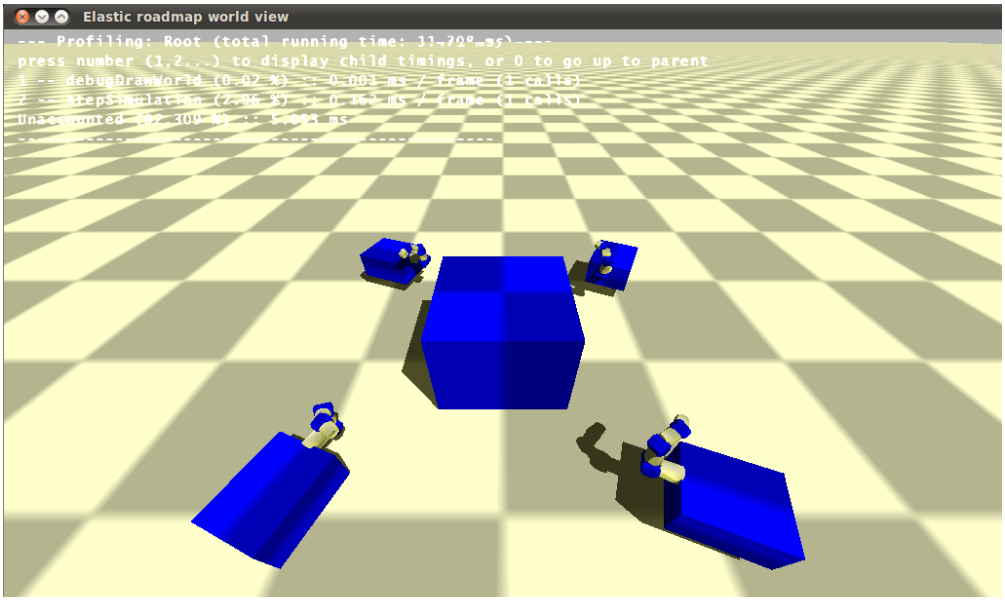
Figur 5.1.: Visualisering av en tom verdensrepresentasjon.

5.2. Testscenarioer

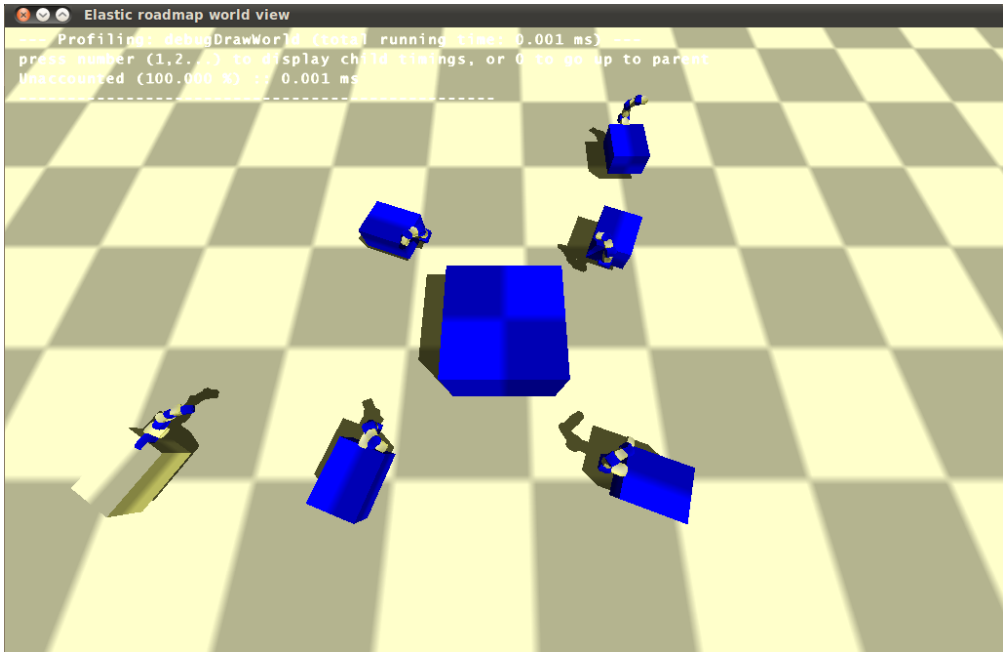
Testing av systemet er inndelt i et sett testscenarioer. Hvert scenario skal illustrere ulike deler av algoritmen og fremheve implementasjonens styrker og svakheter. Fordi implementasjonen av kontrolleren er mangelfull vil de to implementerte oppførslene testes separat. «GlobalBehaviour» er oppførselen som skal forflytte roboten mot et mål, mens «AvoidanceBehaviour» skal sørge for at roboten styrer unna nærmeste



Figur 5.2.: Visualisering av en milepæl. Milepælen fremstår som en virtuell robot i verdensrepresentasjonen.



Figur 5.3.: Visualisering av en kubisk hindring omringet av milepæler.



Figur 5.4.: Visualisering av planlegging og bevegelsesutførelse. Milepælen nederst til venstre på figuren simulerer roboten mens milepælen øverst til høyre representerer robotens målkonfigurasjon (q_G). En hindring hindrer direkte sikt mellom robot og mål.

hindring. «GlobalBehaviour» inngår i alle scenarioer der roboten skal styres mot et mål.

Hvert testscenario beskrives i delkapitlene som følger. En figur viser oppsettet av milepæler og hindringer grafisk. Scenariofigurene er et todimensjonalt koordinatsystem som viser veikartet, det virtuelle arbeidsrommet, sett ovenifra. Det er dermed ingen høydeinformasjon som vises i verken scenariooppsett eller dataplottene i kapittel 6. Aksene i koordinatsystemet følger bakkeplanet slik det er definert i implementasjonen. Målet for planleggingssystemet er i form av en posisjon for verktøyleddet, markert som et diamanttegn. Innad i systemet representeres riktignok målet som en milepæl med en fullstendig konfigurasjon, ikke bare en verktøyleddsposisjon. I illustrasjonsøyemed er det likevel hensiktsmessig å vise et «forenklet» mål der verktøyleddets målposisjon er det mest interessante. Den simulerte roboten vises i form av basens rotasjonsakse og verktøyleddets posisjon. Robotens øvrige koordinatsystem og fysiske dimensjoner inngår ikke i figurene. En oversiktlig presentasjon av de mest interessante delene av roboten er prioritert fremfor eventuell merverdi i form av ekstra informasjon.

5.2.1. Fellesoppsett

Alle oppførsler og kontrollere kjører med en oppdateringsfrekvens på 10 Hz. Synlighet mellom alle milepæler kontrolleres med en frekvens på 0,5 Hz.

5.2.2. Scenario 1

Scenarioet skal demonstrere systemets evne til å nå et mål som befinner seg ute av syne. Den simulerte roboten har en utgangskonfigurasjon

$$q = \begin{bmatrix} x \\ z \\ \theta_b \\ \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_e \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{\pi}{2} \\ \frac{\pi}{2} \\ \frac{\pi}{4} \\ \vdots \\ \frac{\pi}{4} \end{bmatrix}$$

En kubisk hindring med sentrum i $(x, z) = (-5, 5)$ befinner seg mellom roboten og målet, som er $(x, z) = (-10.22, 10.67)$. Scenarioet er illustrert i figur 5.5.

5.2.3. Scenario 2

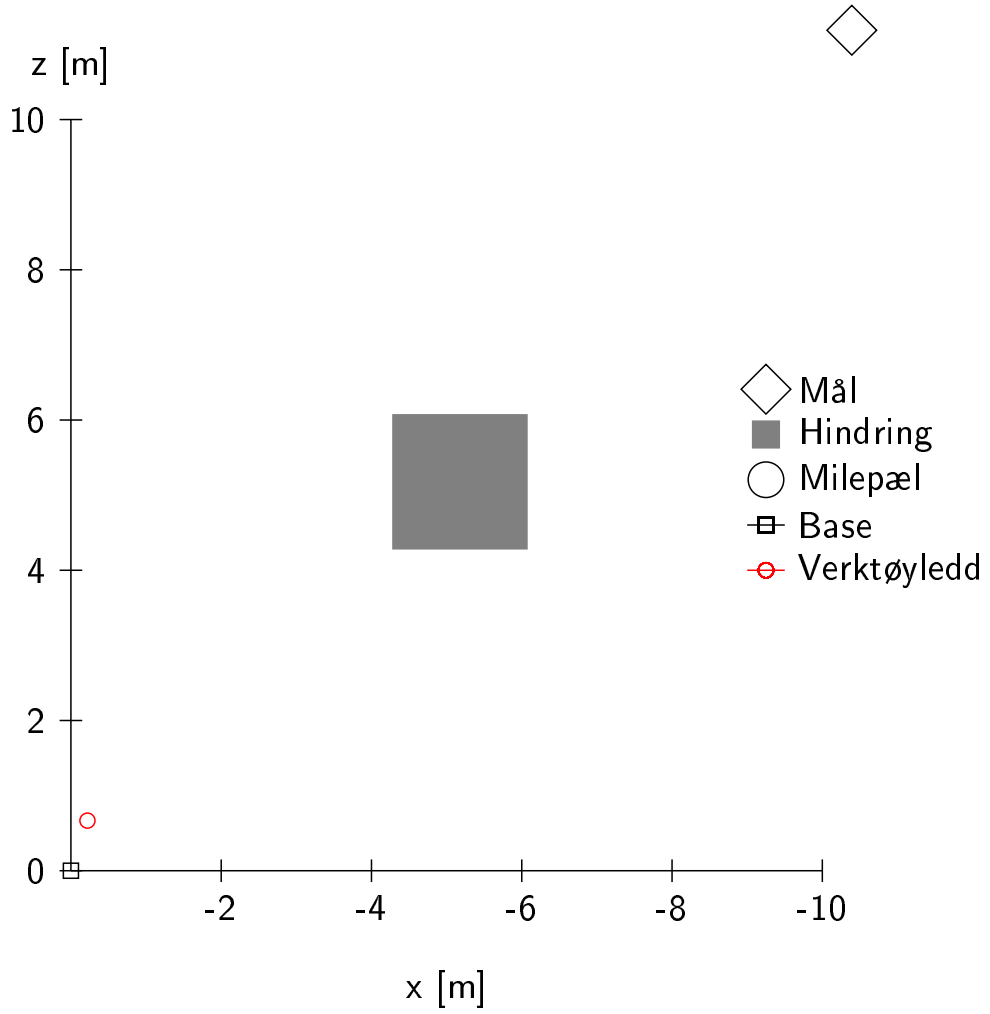
Scenarioet skal demonstrere systemets evne til å tilpasse seg endringer i verdensbildet. Scenarioet er i utgangspunktet identisk med det første scenarioet, men 5 sekunder inn i simuleringen flyttes hindringen i negativ x-retning slik at det blir fri sikt mellom roboten og målet. Forflytningen av hindringen er diskret, slik at dens posisjon umiddelbart endres etter 5 sekunder slik figuren antyder. Scenarioet er illustrert i figur 5.6.

5.2.4. Scenario 3

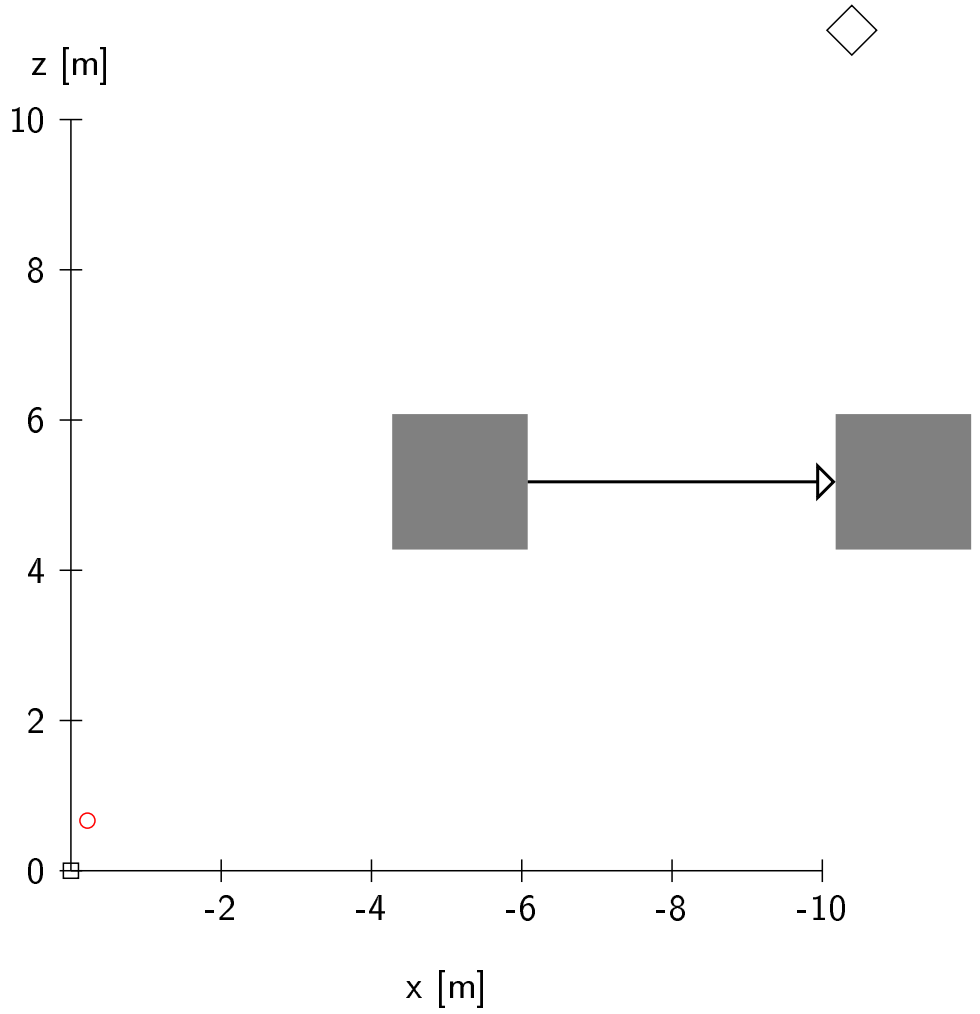
Scenarioet skal demonstrere systemets evne til å tilpasse seg endringer i verdensbildet. Scenarioet er svært likt det foregående scenarioet, men nå flyttes hindringen i positiv x-retning slik at den i utgangspunktet frie siktlinjen mellom robot og mål blokkeres. Som i scenario 2 forflyttes hindringen 5 sekunder inn i simuleringen. Scenarioet er illustrert i figur 5.7.

5.2.5. Scenario 4

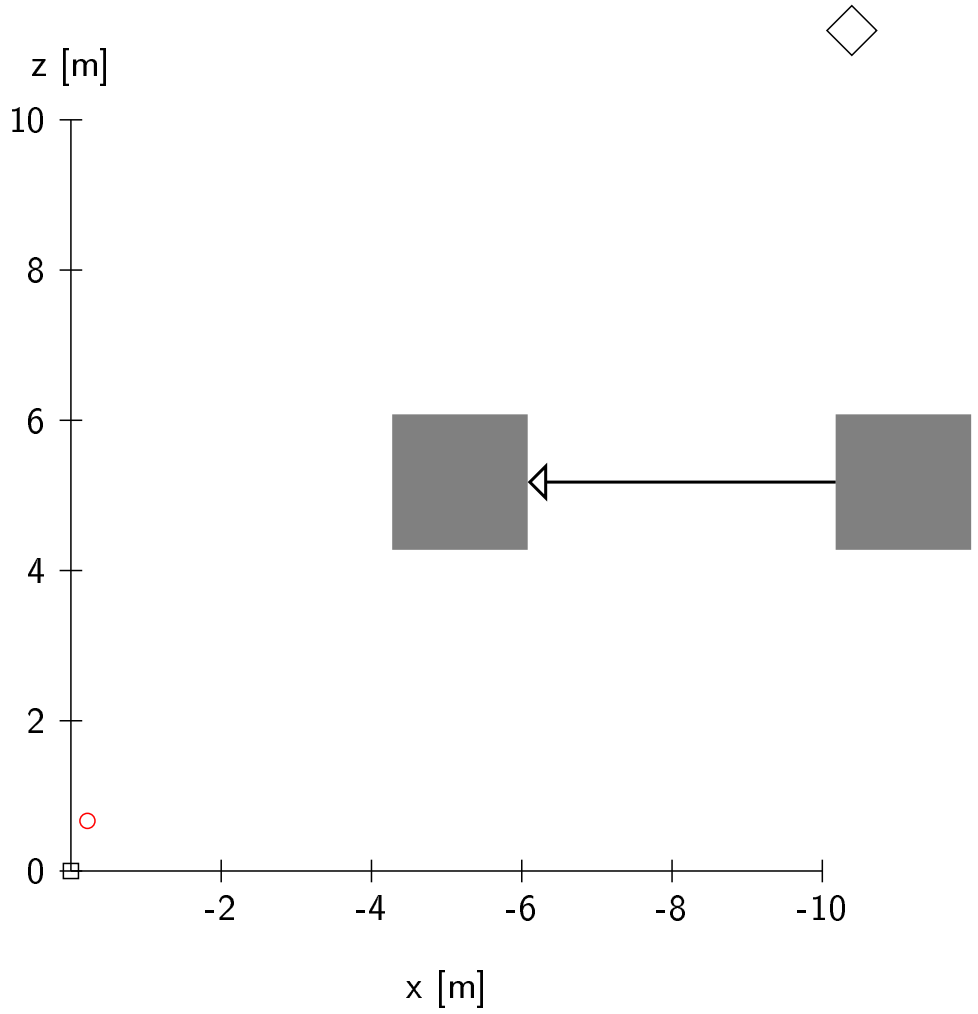
Scenarioet illustrerer problemet med en grov bestemmelse av sammenkoblinger mellom milepæler. Utgangs- og målposisjon er identisk



Figur 5.5.: Testscenario 1: Statisk hindring mellom utgangsposisjon og mål.



Figur 5.6.: Testscenario 2: Dynamisk hindring mellom utgangsposisjon og mål.



Figur 5.7.: Testscenario 3: Dynamisk hindring blokkerer planlagt rute.

fra de foregående scenarioer, men en hindring er nå plassert nært siktlinjen mellom utgangs- og målposisjonene. Scenarioet er illustrert i figur 5.8.

5.2.6. Scenario 5

Scenarioet illustrerer «AvoidanceBehaviour», som skal sørge for å styre roboten unna nærliggende hindringer. Den simulerte roboten har en initiell konfigurasjon

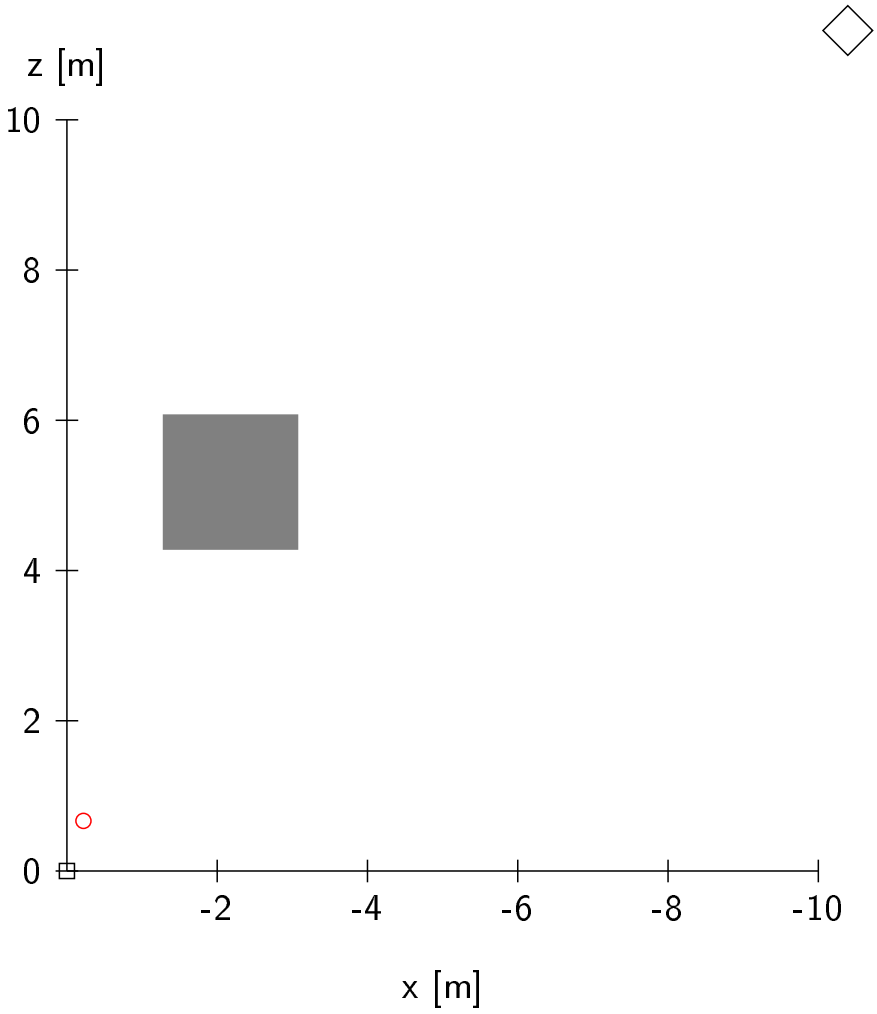
$$q = \begin{bmatrix} x \\ z \\ \theta_b \\ \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_e \end{bmatrix} = \begin{bmatrix} -5 \\ 5 \\ 0 \\ \frac{\pi}{2} \\ \frac{\pi}{2} \\ \frac{\pi}{4} \\ \vdots \\ \frac{\pi}{4} \end{bmatrix}$$

En hindring er plassert nær roboten med sentrum $(x, z) = (-7, 5)$. Scenarioet er illustrert i figur 5.9.

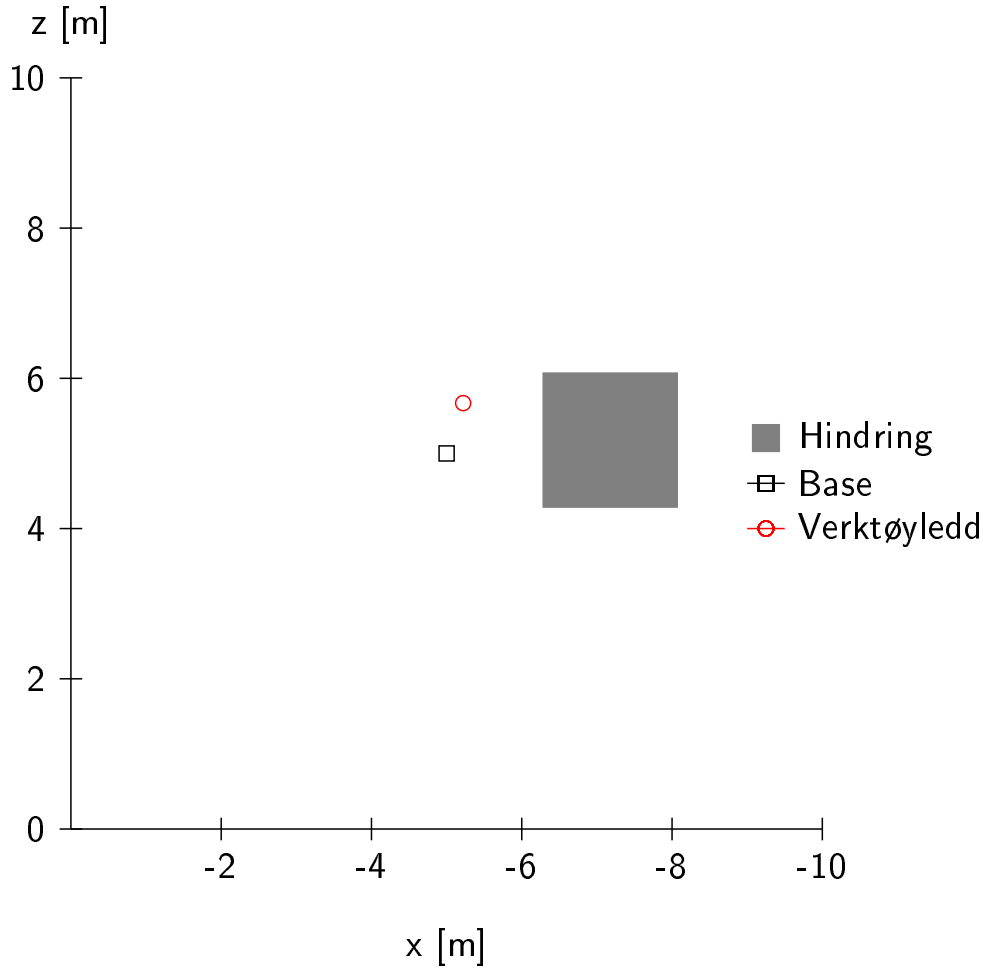
5.3. Sammendrag

Dette kapittelet har presentert et sett med simuleringsscenarioer som skal demonstrere systemets evner og forbedringspotensiale:

- Scenario 1: Statisk hindring mellom utgangsposisjon og mål.
- Scenario 2: Dynamisk hindring mellom utgangsposisjon og mål.
- Scenario 3: Dynamisk hindring blokkerer planlagt rute.
- Scenario 4: Statisk hindring og tilsynelatende klar bane.
- Scenario 5: Hindringsunngåelse.



Figur 5.8.: Testscenario 4: Tilsynelatende klar bane.



Figur 5.9.: Testscenario 5: Hindringsunngåelse

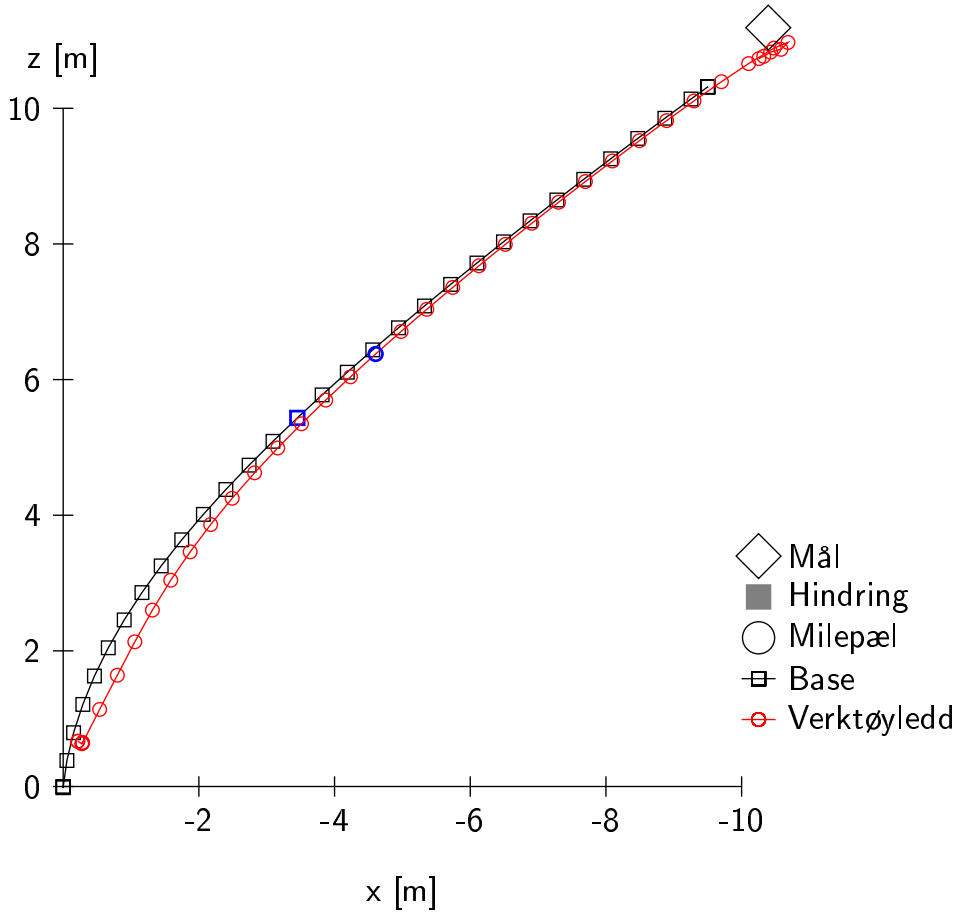
6. Resultat

Resultatene etter simulering av testscenarioene beskrevet i kapittel 5 presenteres i dette kapittelet. Resultatet presenteres på samme format som testscenarioene, i form av et koordinatsystemplott. Figurene som utgjør resultatplottene følges av en tekstlig beskrivelse. Vurdering av resultatene gis i kapittel 7.

6.1. Figurforklaring

Resultatsfigurene er gitt som et todimensjonalt koordinatsystem lagt til systemets bakkeplan, tilsvarende som i foregående kapittel. Den simulerte robotens rotasjonsakse og verktøyleddsposisjon vises som distinkte punkter med rette linjer trukket mellom punktene. Hvert punkt langs disse posisjonslinjene representerer ett sekunds avstand i tid. Dette gir en viss følelse av robotens tidsbruk og hastighet under simuleringen. I scenarioer der det inngår hindringer vil det ved hvert hjørne av hindringen være en milepæl. Figur 5.3 på side 60 illustrerer oppsettet av milepæler og hindringer. Milepælene illustreres som en sirkel med sentrum lagt til milepælens verktøyledd. I scenarioene der en hindring flyttes illustreres milepælenes bevegelse i form av overlappende sirkler, der det er ett sekund mellom hver forflytning av milepælsirkelen. For å gjøre figurene mer oversiktlig er enkelte «irrelevante» milepæler ikke vist, selv om hver hindring har fire tilhørende milepæler. I et forsøk på å gi en bedre illustrasjon av bevegelse under simuleringen er enkelte posisjonsmarkeringer tegnet i blått. Blåfargen illustrerer posisjonen til markeringen når det har gått 20 sekund av simuleringen. Alle blåmalte posisjoner er dermed på samme sted i tid. Figur 6.1 viser hvordan den simulerte robotens bevegelser angis og illustrerer «dynamikken» i simuleringen. Roboten i figuren starter med fartsretning langs z-aksen i figuren.

Spesifikke milepæler omtales etter milepælens kompassretning i forhold til hindringen den er tilknyttet. For eksempel den «nordvestre» milepælen befinner seg i positiv z-retning og positiv x-retning i forhold til hindringen den er tilknyttet. Positiv z-retning er altså «nord», mens negativ z-retning er «sør».



Figur 6.1.: Demonstrasjon av posisjonsplott.

6.2. Scenario 1

Figur 6.2 viser resultatet av simuleringen. Roboten, som starter i koordinatsystemets origo, beveger seg mot og innom milepælen nordvest for hindringen før den fortsetter mot målet.

6.3. Scenario 2

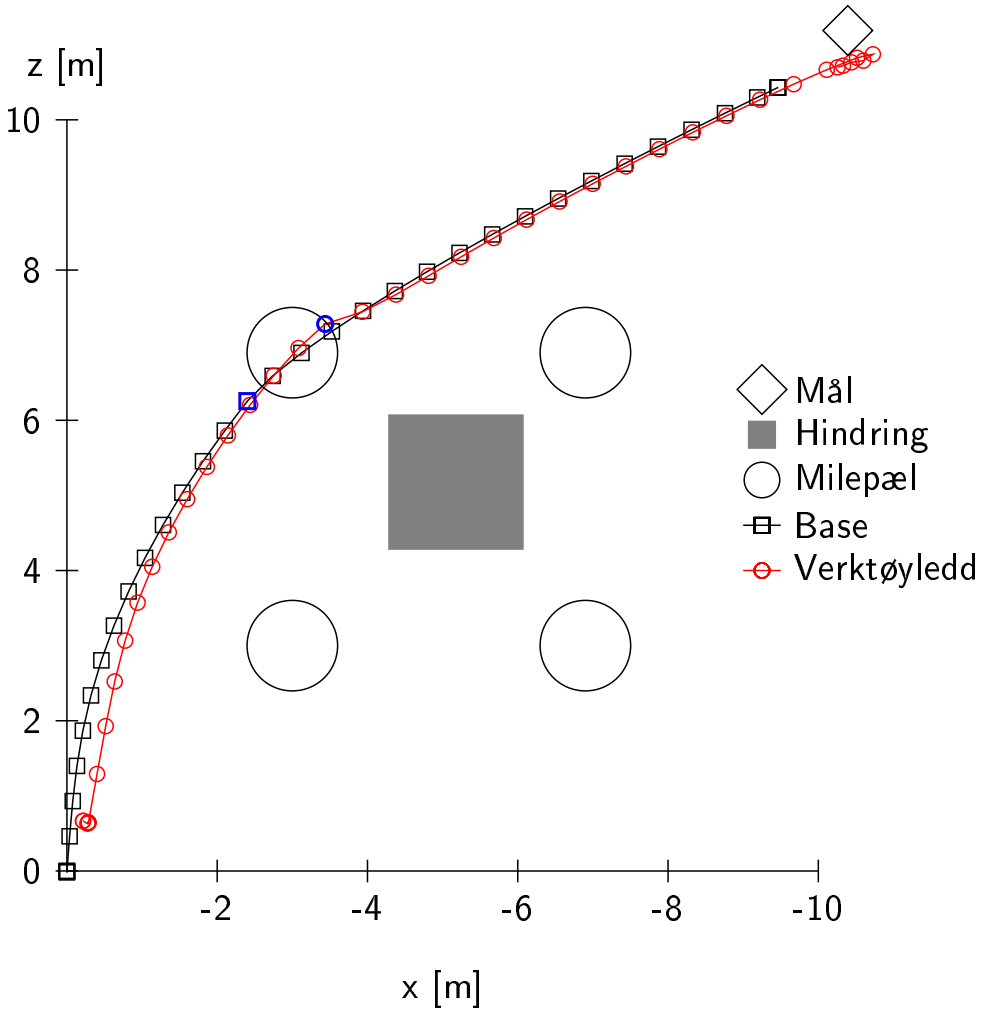
Figur 6.3 viser resultatet av simuleringen. De fem første sekundene av simuleringen er tilnærmet identisk det første scenarioet. Etter fem sekunder flytter hindringen seg og de tilknyttede milepælene flytter etter hindringen. Roboten har som i scenario 1 en planlagt rute som går innom den nordvestre milepælen, som den følger etter og innom før den fortsetter mot målet.

6.4. Scenario 3

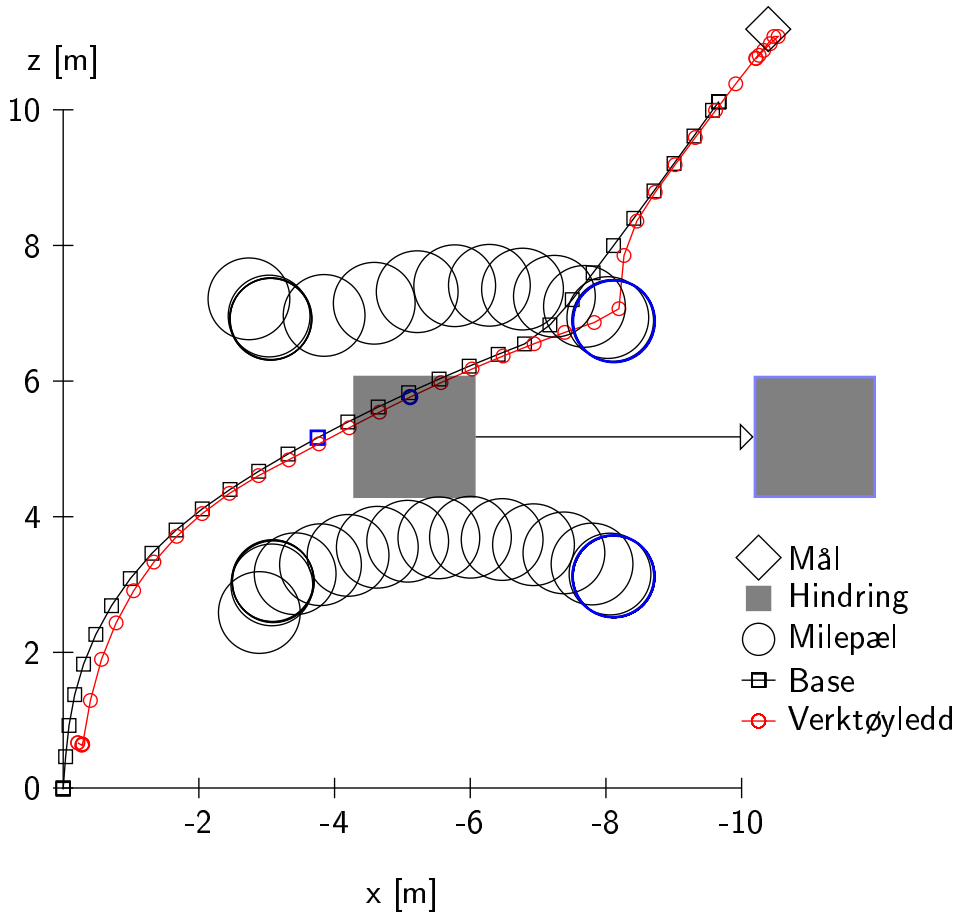
Figur 6.4 viser resultatet av simuleringen. Roboten har i de fem første sekundene av simuleringen klar bane mot målet. Etter fem sekund flyttes hindringen og blokkerer veien mot målet. En ny plan formuleres som sender roboten innom den sørøstre milepælen. Det fremkommer ikke tydelig av figuren om den sørvestre milepælen er en del av planen. Etter 20 sekund har milepælene tilpasset seg den nye hindringsplasseringen og man kan se at roboten passerer den sørøstre milepælen noen sekunder senere, før den går mot målposisjonen.

6.5. Scenario 4

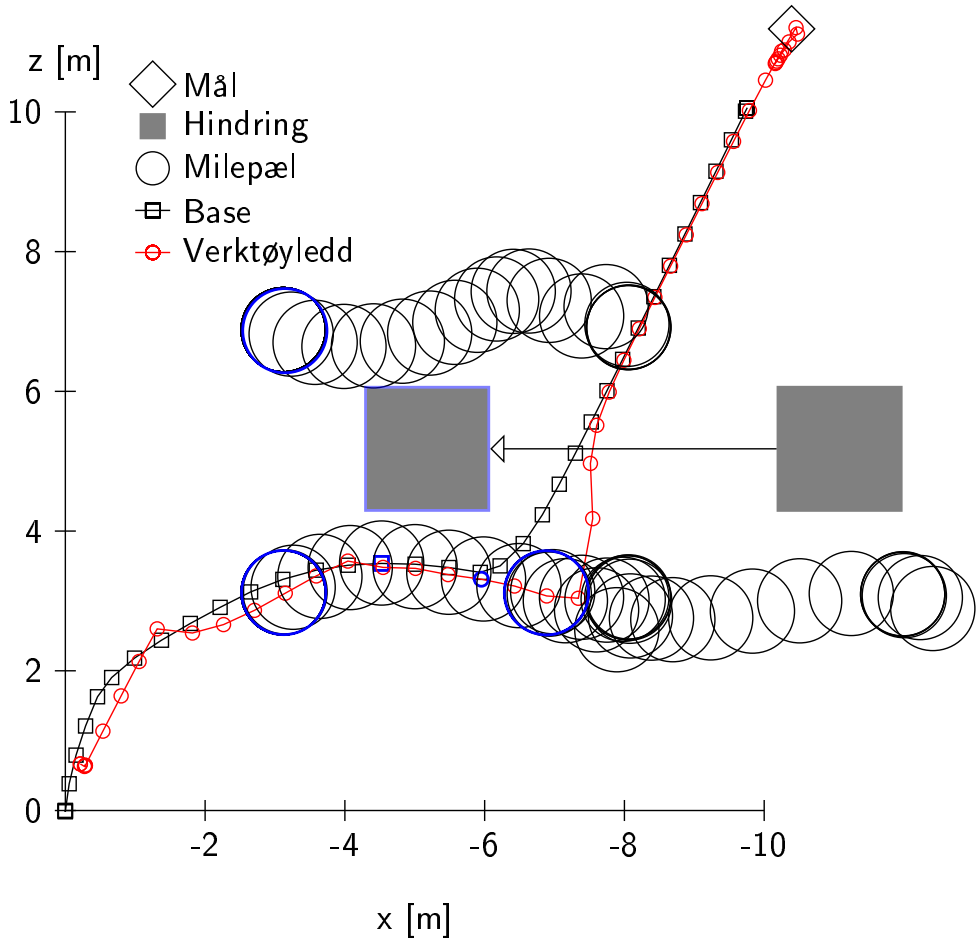
Figur 6.5 viser resultatet av simuleringen. Det er i utgangspunktet fri sikt mellom roboten og målet, og roboten setter avgårde i retning målposisjonen. Etter ca. 15 sekunder oppdateres sammenkoblingene mellom milepæler og veien roboten fulgte blir ugyldiggjort. En ny rute planlegges, og man kan se at basen svinger i retning av den nordvestre milepælen. Veien mot den nordvestre milepælen ugyldiggjøres raskt, da hindringen kommer mellom manipulatorarmen og



Figur 6.2.: Posisjonsplott for testscenario 1.

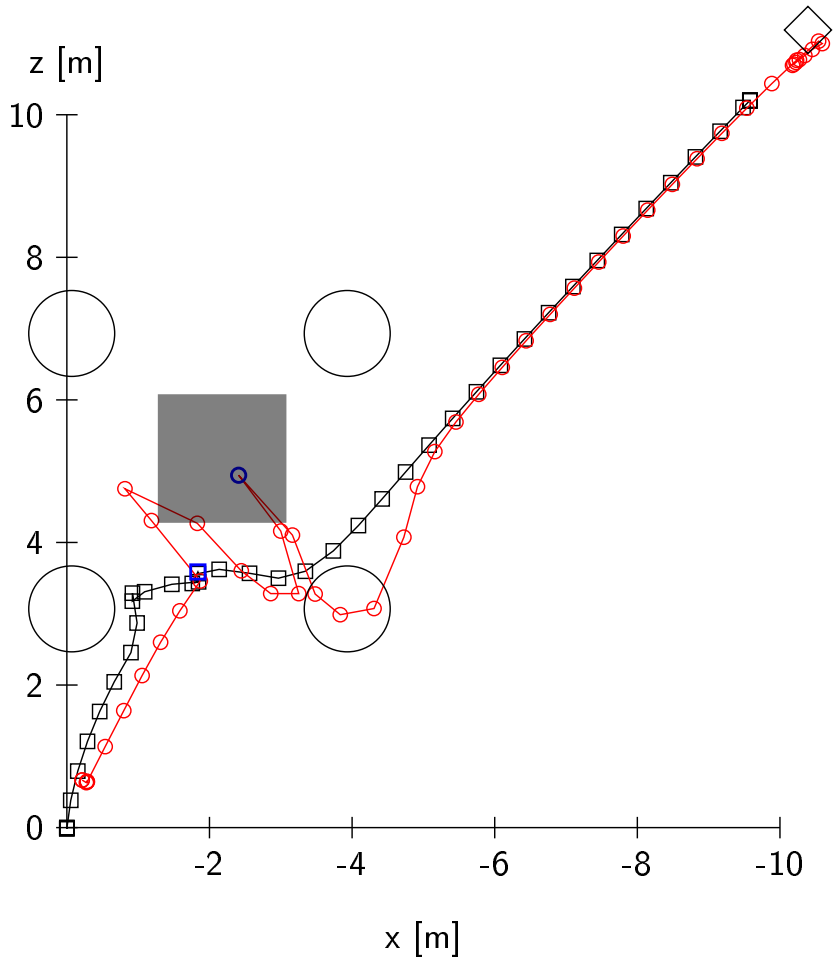


Figur 6.3.: Posisjonsplott for testscenario 2.



Figur 6.4.: Posisjonsplott for testscenario 3.

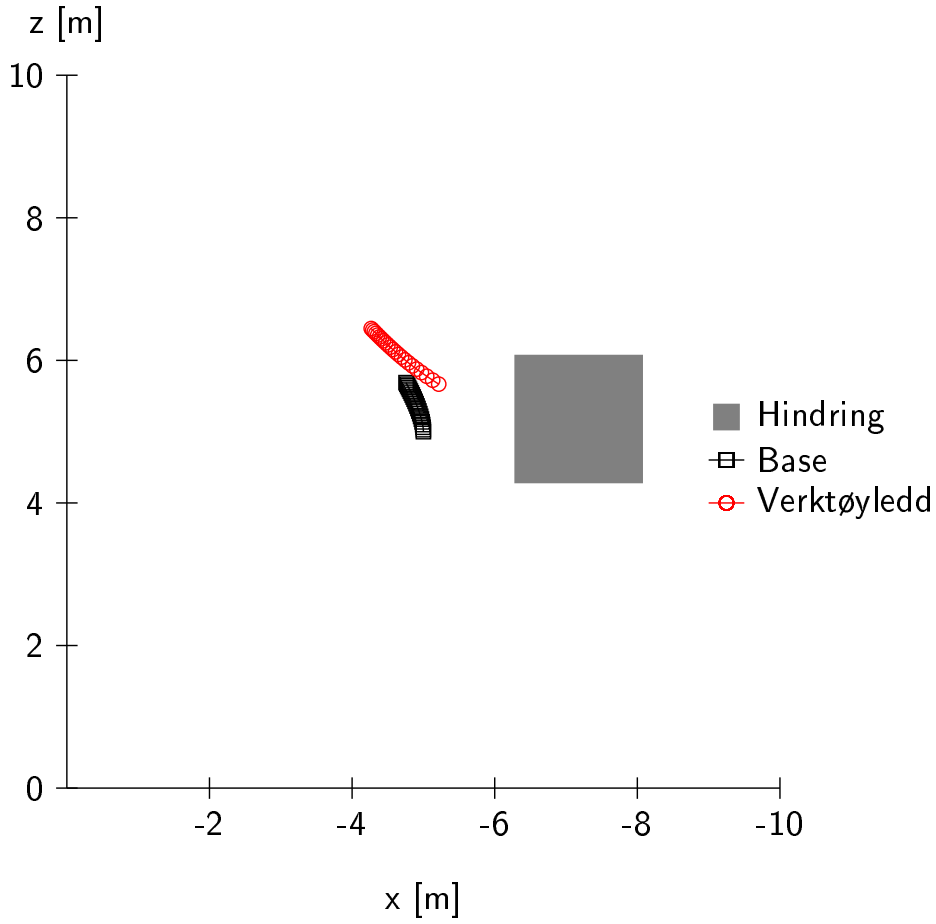
milepælen. Det oppstår noen oscillasjoner før roboten fortsetter mot den sørøstre milepælen og til slutt mot målposisjonen. Overlappende verktøyleddsposisjon og hindring antyder en kollisjon.



Figur 6.5.: Posisjonsplott for testscenario 4.

6.6. Scenario 5

Figur 6.6 viser resultatet av simuleringen. Roboten har her ikke en målposisjon og all bevegelse skyldes nærhet til en hindring. Roboten fjerner seg fra hindringen med stadig synkende hastighet.



Figur 6.6.: Posisjonsplott for testscenario 5.

6.7. Sammendrag

Dette kapittelet har vist og beskrevet resultatet etter simulering av scenarioene som ble beskrevet i forrige kapittel:

- Scenario 1: Statisk hindring mellom utgangsposisjon og mål.
- Scenario 2: Dynamisk hindring mellom utgangsposisjon og mål.
- Scenario 3: Dynamisk hindring blokkerer planlagt rute.
- Scenario 4: Statisk hindring og tilsynelatende klar bane.
- Scenario 5: Hindringsunngåelse.

Resultatene er gitt som et posisjonsplott i et koordinatsystem lagt til verdensrepresentasjonens bakkeplan. En tekstlig beskrivelse følger hvert av plottene.

7. Analyse og diskusjon

Kapittelet tar for seg resultatene etter simuleringen presentert i foregående kapittel. Hvert scenario blir først gjennomgått hver for seg, deretter diskuteres de gjennomgående resultatene etter simuleringen. Til slutt i kapittelet diskuteres rapporten i sin helhet.

7.1. Analyse av testscenarioer

7.1.1. Scenario 1

Scenarioet er, i et planleggingsperspektiv, et relativt enkelt oppsett med en enkelt hindring mellom en utgangs- og målposisjon. Scenarioet skal illustrere hvordan systemet tenker når det finner en rute fram til målposisjonen. Det er flere veier fram til målet, og algoritmen velger en av de to korteste veiene. Den andre korteste veien ville vært innom den sørøstre milepælen vist i figur 6.2. Fordi roboten starter med fronten mot positiv z-retning er det hensiktsmessig å velge ruten som ble valgt, da det innebærer en mindre svingradius. Grunnen til at den nordvestre, og ikke den sørøstre milepælen ble valgt er riktignok at rekkefølgen milepælene opprettes i har betydning for søkealgoritmen som bestemmer veien (Dijkstras algoritme). Når to veier har lik kostnad velges veien med milepæler som tidligst ble lagt til i datastrukturen.

Kanskje det mest påfallende med scenarioet er at verktøyleddet aldri når målposisjonen. Den nærmer seg med stadig synkende hastighet, men kommer aldri helt frem før simuleringen avsluttes. Det skyldes at oppførselen som styrer bevegelsen, «GlobalBehaviour», gir et moment basert på et potensialtfelt. Feltet har en tiltrekkende komponent lagt til målposisjonen, der det resulterende momentet minker etter som roboten nærmer seg målposisjonen. Kraften i feltet er et resultat av ligning 3.5 på side 45. Oppførselen gir likevel et pådrag som flytter

roboten i henhold til planen slik at den ender opp svært nærme målet.

7.1.2. Scenario 2

Dette scenarioet introduserer dynamikk til verdensrepresentasjonen og skal illustrere hvordan spesielt milepælene forholder seg til endringer. Endringen er i seg selv ikke spesielt realistisk. En «oppdaget» hindring som flyttes 6 meter i ett steg ville i et praktisk situasjon kunne vært et tegn på betydelig usikkerhet i sansingen av virkeligheten. Like fullt flytter milepælene etter hindringen og opprettholder den frie konfigurasjonen de representerer.

I simuleringen er det ingen dynamisk eller kinematisk forskjell på «roboten» og andre milepæler. Milepælene som følger etter hindringen er underlagt den samme «trege» reguleringen som roboten, selv om de ikke representerer en virkelig robot på samme måte. Det ville vært fordelaktig å kunne fjerne reguleringsbegrensningen for milepælene slik at de har fulgt hindringen tettere. Reguleringen av robot og milepæler består i scenarioet av en «GlobalBehaviour»-oppførsel med helt likt parameteroppsett. «Hissig» parametertuning av milepælene kan bidra til tettere følging av hindringen. «Halvbuen» milepælene beveger seg i er et resultat av reguleringen og er ikke essensiell for å opprettholde en gyldig milepæl.

Når hindringen etter 5 sekunder flyttes åpner det seg en direkte vei mellom roboten og målposisjonen. Likevel går roboten innom milepælen den gikk mot før hindringen ble flyttet. Algoritmen beregner ikke en ny rute med mindre kanten som følges blir ugyldiggjort eller det angis et nytt mål. Alternativt kunne den planlagte ruten ha blitt reevaluert med jevne mellomrom, eller når endringer har oppstått. Korteste-vei-søket som bestemmer planen er utregningsmessig relativt billig og en reevaluering ville i dette scenarioet gitt et bedre resultat.

7.1.3. Scenario 3

Som i foregående scenario er det her dynamikk og endringer i verdensrepresentasjonen. Nå endres oppsettet slik at en allerede planlagt rute blir ugyldiggjort og roboten må tilpasse seg endringene. Systemet op-

pdager at det ikke lenger er fri sikt mellom roboten og målposisjonen få sekunder etter at endringen har oppstått. Merk at det ikke er selve endringen som utløser oppdagelsesprosessen, men «ConnectivityController»-komponentens[1] jevnlige sjekk av sammenkoblingene i grafen. Når systemet oppdager at kanten som følges ikke lenger er gyldig planlegges det en ny vei mot målet.

Etter 5 sekunder er hindringens plassering den samme som i scenario 1, man kunne dermed antatt at den samme ruten som i scenario 1 ble valgt. Den nordvestre milepælen befinner seg derimot mellom hindringen og målposisjonen, skjult fra roboten, så en lengre rute blir valgt. Det fremkommer ikke tydelig i resultatfiguren om roboten er innom den sørvestre milepælen eller ikke.

Som i scenario 2 er milepælene som følger hindringen relativt saktegående. Det går opp mot 15 sekund fra hindringen flyttes og til milepælenes posisjon stabiliserer seg.

7.1.4. Scenario 4

Scenarioet viser hvor usikker bestemmelsen av sammenkoblinger mellom milepælene er. Selv om det er fri sikt mellom roboten og målet sørger kinematikken og dynamikken i systemet for at målet ikke kan nås direkte. Når roboten nærmer seg hindringen blokkeres sikten til målet og en ny plan formuleres. Roboten går så i retning den nordvestre milepælen. Det oppstår en oscillasjon der roboten veksler mellom å styre mot den nordvestre og den sørøstre milepælen. I oscillasjonens ytterpunkter ligger hindringen mellom verktøyleddet og en av milepælene og det ser ut til at kantene mellom roboten og milepælene veksler mellom å være gyldige og ugyldige. Uansett oppstår det en kollisjon mellom robot og hindring, noe som er uheldig. Spesielt når systemet ikke har noen formening om hva hindringen representerer er det ikke godt å si om noen eller noe har blitt skadet.

Det kan spekuleres i om kollisjonen kunne ha vært unngått hvis designet hadde vært fullstendig implementert. Simuleringen er med én enkelt oppførsel som ikke tar hensyn til nærliggende hindringer. Hvis «AvoidanceBehaviour» hadde vært inkludert som en høyere prioritert

[1] Systemkomponent vist i figur 3.3 på side 37.

oppførsel kunne forhåpentligvis kollisjonen ha blitt unngått. Oppførselen gir en kraft med retning bort fra den hindringen som er nærmest roboten. Roboten kunne i det tilfellet blitt styrt rundt hindringen på trygg avstand, eller blitt holdt på avstand til en ny plan er formulert.

Robotbasen er aldri innom den sørøstre milepælen som roboten styrer innom. Kun verktøyleddet passerer milepælen før roboten går videre mot målet. Det er gunstig i den forstand at basen ikke er like manøvrerbar som manipulatorarmen. Riktignok kan det være at man ønsker å holde basen mer innenfor området milepælene dekker, eller at man vil holde armen så sammenfoldet som mulig for å minimere kollisjonsfaren.

7.1.5. Scenario 5

Dette er det eneste scenarioet der unngåelsesoppførselen demonstreres. Roboten starter inntil hindringen og beveger seg med synkende hastighet bort fra hindringen. Sammenliknet med robotens bevegelse i tidligere scenarioer er hastigheten roboten beveger seg med relativt lav. Det vil være en tuningsoppgave å tilpasse størrelsen på kraftvektoren som «dytter» roboten unna i tillegg til å bestemme kraftens rekkevidde.

7.2. Diskusjon av simuleringsresultater

Selv om implementasjonen mangler det som har blitt karakterisert som kjernefunksjonalitet viser resultatene etter simulering at systemet er i stand til å planlegge og utføre en bevegelse som forflytter en simulert robot fra en startkonfigurasjon og til en målkonfigurasjon. Den grunnleggende oppgaven, bevegelsesplanlegging, er dermed vist å være fungerende, men på ingen måte perfekt.

Ytelsesmessig viser systemet seg å håndtere endringer i sanntid. Scenarioene representerer likevel ingen ytelsestest der systemets grenser utforskes. Utvilsomt representerer vedlikehold av milepælene med sine kontrollere den største prosesseringskostnaden, noe som kan legge begrensninger på bruken av systemet.

7.3. Diskusjon av rapport

Rapportens første del, litteraturstudiet, gir en relativt bred fremstilling av teori rundt planlegging. Feltet er stort, og det er sannsynlig at viktig og relevant teori er utelatt. Bredden i feltet gjør at mye av det som presenteres er svært overfladisk, som kan gi en unyansert fremstilling av stoffet. «Elastic Roadmap» gis som den mest inngående og detaljerte beskrivelsen av stoff i litteraturstudiet og danner grunnlaget for resten av rapporten. Litteraturstudiet kunne blitt gjort mer detaljert og tettere knyttet opp mot teorien som ligger til grunn for det elastiske veikartet. Det ville riktignok gått på bekostning av bredden i litteraturstudiet, og leseren ville ikke fått samme mulighet til å vurdere algoritmen opp mot eksisterende problematikk og løsninger.

Rapportens andre del, som omhandler design av algoritmesystemet, gir en relativt detaljert presentasjon av systemets viktigste deler. Fordi systemet er såpass «spredt» kan det være vanskelig å få oversikt over den helhetlige funksjonaliteten. Et mer overordnet design kan gi en bedre forståelse av flyt og funksjonalitet, men lett utelate interessante og viktige detaljer.

Systemimplementasjonskapittelet er det mest kortfattede kapittelet. Det er svært få detaljer rundt det «ferdige» systemet som gjennomgås. Større innsikt i implementasjonen kunne vært interessant, men med fare for å kjede leseren med «uviktige» detaljer. Omfattende dokumentasjon er dessuten å finne på vedlagt CD (se sysdoc.pdf i vedlegg A). Den vedlagte dokumentasjonen gir en strukturert og detaljert innsikt i systemets kodebase. Den har riktignok et format og detaljnivå som gjør den mest nyttig for fremtidig utvikling av systemet. Ytterligere detaljer rundt tredjepartsbibliotek som er benyttet kan gi en bedre forståelse av både systemet og eventuelle muligheter biblioteket gir på lengre sikt. Igjen med fare for å gi «uviktig eller unyttig» informasjon, som uansett er tilgjengelig og referert til. Kapittelet gir derimot en konsis forklaring som gir en forståelse av bibliotekets nytteverdi og bakgrunnen for valget av biblioteket. Kapittelet gir derimot ikke noe sammenlikningsgrunnlag som kan rettferdiggjøre valget av tredjepartsbiblioteket fremfor andre bibliotek.

Gjennom simulering av en rekke testscenarioer og analysering av resultatene vises det at systemet er i stand til å planlegge og utføre

en bevegelse i et dynamisk miljø. Oppgaven er i den forstand utført, på tross av at systemets potensial, som er beskrevet i del 2.7 og gjennom designkapittelet, ikke er utnyttet til det fulle. Det implementerte systemets styrker og svakheter er fremhevet og diskutert. Systemets svakhet i form av uutnyttet potensiale er dessuten fremhevet, og til en viss grad demonstrert gjennom simulering (scenario 4).

8. Konklusjon

En lang rekke problemer har en løsning som innebærer en eller annen for form *plan*. En optimal løsning på Rubiks kube er eksempelvis i form at et sett diskrete steg, rotasjoner av kubens deler. Forflytning av en robot fra en utgangsposisjon frem til en målposisjon innebærer ofte en mer kontinuerlig plan i form av en geometrisk kurve. Hvis forflytningen innebærer at roboten må styres unna hindringer eller opprettholde begrensninger ulik art kompliseres raskt oppgaven. Oppgavens første del, litteraturstudiet, omhandler bevegelsesplanlegging for roboter. Det gis et innblikk i ulik metodikk som behandler problemet fra ulike sider. Fra planlegging på et høyt abstraksjonsnivå som ikke tar hensyn til detaljer rundt planutførelsen og ned til planleggingsalgoritmer som nøyaktig spesifiserer en geometrisk kurve i rommet. Bevegelsesplanlegging betraktes ofte som et flerstegsproblem, der en *veiplanlegger* gir en overordnet plan, en *kurveplanlegger* raffinerer planen med tanke på kinematikk og dynamikk, mens en *regulator* tar seg av styringen av roboten i henhold til planen gitt av kurveplanleggeren. *Bevegelsesplanlegging* er en samlende betegnelse som er tenkt å kombinere vei- og kurveplanlegging til én effektiv planleggingsprosess. «Elastic Roadmap» er en slik bevegelsesplanlegger, som i motsetning til andre planleggingsalgoritmer bevisst ofrer optimalitet til fordel for effektivitet. Resultatet er et algoritmesystem som formulerer en svært løs og ikke nødvendigvis optimal plan, men som klarer å ta hensyn til utfordringene en mobil robot i et dynamisk miljø er stilt overfor.

Resten av oppgaven etter litteraturstudiet omhandler design, implementasjon og testing av et «Elastic Roadmap»-inspirert algoritmesystem. Systemet designes etter det rammeverket nedlagt av Yang og Brock [23], men med noen endringer der det er hensiktsmessig. Deler av implementasjonsprosessen omtales der det er hensiktsmessig. Spesielt er bruken av tredjepartsbibliotek til kollisjonsdetektering relativt sentralt i implementasjonen. I tillegg omtales manglende imple-

mentasjon av komponenter som inngår i systemdesignet. Det implementerte systemet testes gjennom et sett med fem simulerte testscenarioer. Disse skal demonstrere systemets evne til bevegelsesplanlegging under ulike omstendigheter. Manglende funksjonalitet illustreres både i form av fravær av testscenario for den manglende funksjonaliteten, og gjennom et testscenario der den simulerte roboten kolliderer med en hindring.

Rapporten har gjennom det beskrevne arbeidet dekket oppgavens hoveddeler som omhandler et litteraturstudie av emnet, et design og implementasjon av et algoritmesystem for bevegelsesplanlegging for mobile roboter, samt analysering og vurdering av dette systemet.

8.1. Videre arbeid

Fremtidig utvikling av systemet foreslås både i et kort- og langsiktig perspektiv. På kort sikt vil fullføring av det planlagte designet kunne gi et bedre vurderingsgrunnlag for utvikling på lang sikt. Spesielt gjelder dette kontrolleren som inngår i det planlagte designet. Kontrolleren skal kombinerer pådrag fra ulike underoppførsler, men er som nevnt tidligere mangelfullt implementert og kan kun håndtere én underoppførsel. Videre kan det være hensiktsmessig å implementere ytterligere oppførsler, avhengig av hvilke bruksområder som tillegges systemet. Eksempelvis kan en oppførsel sørge for at orienteringsbegrensninger overholdes (à la kaffeoppgaven brukt som eksempel innledningsvis i rapporten). Med en fullstendig kontroller kan forhåpentligvis situasjoner slik den som oppstod i simuleringsscenario 4 unngås. Eksisterende oppførsler kan forbedres og tilpasses bruk på roboten, for eksempel sørge for at kraftpådraget er i en passende størrelsesorden. I simuleringsscenarioene viste det seg at verktøyleddet aldri helt nådde målposisjonen, noe som kan behjelpes med eksempelvis en form for integrasjonseffekt.

Når planleggingsystemet skal tas i praktisk bruk må systemet ytterligere tilpasses den fysiske roboten det skal planlegges for. Det innebærer først og fremst å sørge for at det er samsvar mellom de fysiske og modellerte størrelser, og gi et pådrag fra kontrolleren på et format som er håndterlig for roboten. Ved posisjonsregulering av roboten er

Figur 8.1.: Tenkt grensesnitt for enkel samhandling mellom ROS og planleggingssystemet.

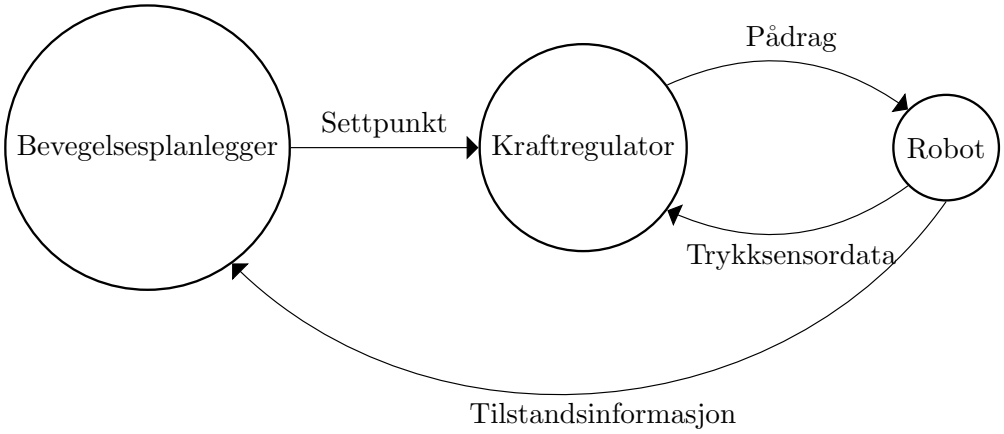
det en direkte sammenheng mellom milepælens konfigurasjonsvektor, mens ved hastighetsregulering kan konfigurasjonsvektoren deriveres. Tilsvarende er det nødvendig å tolke tilstandsinformasjon gitt av den fysiske roboten på korrekt vis.

Det er nødvendig å opprette et grensesnitt mot planleggingssystemet slik at hindrings- og annen verdensinformasjon kan gis til systemet. Under utformingen av et slikt grensesnitt bør det tas hensyn til de komponentene som skal kommunisere med planleggingssystemet for å kunne gi et hensiktsmessig format på grensesnittet. Et tenkt grensesnitt er vist i figur 8.1. Deler av robotsystemet kan angi et mål for planleggingsprosessen, samt holde planleggingssystemets verdensbilde oppdatert i forhold til hindringer i robotens arbeidsrom. Mer kompliserte manøvre, for eksempel hvis roboten skal gripe noe, kan planlegges som en serie med målkonfigurasjoner, der den siste konfigurasjonen er et grep om en gjenstand.

En potensiell flaskehals i systemet er antallet milepæler. En bedre strategi for plassering av milepæler kan gi gevinst både i form av mindre vedlikehold og kortere planleggingsruter. Som nevnt i del 2.7 kan man ved jevne mellomrom forsøke å redusere antallet milepæler og vurdere resultatet av milepælbeskjæringen. Vedlikeholdet den enkelte milepælen utgjør kan også reduseres ved å forenkle kontrolleren tilknyttet milepælen, for eksempel ved at milepælkontrollere har en lavere oppdateringsfrekvens enn robotkontrolleren eller har et enklere oppførselsoppsett.

På litt lengre sikt må robotens ikkeholonomiske aspekter tas grundig

i betraktning. Problemstillingen ignoreres mer eller mindre i implementasjonen utført under denne rapporten. Hvis man vil for eksempel lukeparkere med roboten må man ta hensyn til dette aspektet. En mulighet er å erstatte det nåværende kontrolleroppsettet til fordel for en lokal kurveplanlegger som bestemmer en mer nøyaktig plan mellom milepæler. En slik kurveplanlegger kan da i tillegg mer nøyaktig bestemme om to milepæler er sammenkoblede, i motsetning til den enkle synlighetstesten den nåværende implementasjonen bruker. En lokal planlegger bryter derimot med det grunnleggende prinsippet bak det elastiske veikartet, nemlig at optimalitet og kompletthet ofres til fordel for effektivitet. En for kostnadskrevende lokal planlegger kan gå på bekostning av evnen til å holde tritt med omgivelsesendringer. Figur 8.2 viser et tenkt design der kostnadskrevende kraftregulering[1] gjøres nærmere roboten, uten at planleggingssystemet er direkte involvert. Alternativt kan en egen oppførsel brukes til å gi et pådrag som følger en kurve som tar hensyn til ikkeholonomien. For eksempel en kontroller
$\Phi = \phi_{global} \triangleleft \phi_{curve} \triangleleft \phi_{avoidance}$, der ϕ_{curve} har en «intern kurve» den vil følge, og gir et pådrag deretter.



Figur 8.2.: Mulig arbeidsdeling mellom planlegger og arbeidskrevende regulering.

[1]Regulering av trykkraften roboten påfører omgivelsene.

Hvis planleggingssystemet skal brukes på andre roboter, eller hvis robotoppsettet er utsatt for hyppige endringer kan det være hensiktsmessig å inkludere et strukturert konfigurasjonsoppsett. ROS inkluderer blant annet et kraftig skriptbasert system for tekstfilbasert konfigurasjon av et program. For eksempel kan man bruke en skriptfil til å beskrive en robot kinematisk, som deretter tolkes av planleggingssystemet. En klasseimplementasjon av «RobotModel»-grensesnittet (se figur 3.12 på side 50) kan benyttes til å opprette en intern robotmodell basert på en konfigurasjonsfil. Lettvint konfigurasjon av kontroll- og oppførselsparametre er også noe som kan gjøres via et strukturert konfigurasjonsoppsett.

8.2. Sammendrag

Kapittelet har kortfattet omtalt rapporten i sin helhet, og konkludert med at den er dekkende i forhold til oppgavebeskrivelsen. Like fullt er det mye forbedringspotensiale og flere utviklingsretninger systemet kan ta. Fremtidig utvikling av systemet foreslås å inkludere følgende punkter:

- Fullføre kontrollerimplementasjon.
- Implementere flere oppførsler etter behov.
- Tuning og forbedring av eksisterende oppførsler.
- Tettere kobling mot robotens kontrollgrensesnitt.
- Smartere plassering av milepæler.
- Opprettelse av et grensesnitt mot planleggingssystemet.
- Håndtering av ikkeholonomiske utfordringer
- Strukturert konfigurasjonsoppsett

A. CD

Vedlagt CD inneholder følgende:

- **ElasticRoadmap/** Det implementerte systemet i form av en Ros-pakke.
- **Rapport/** Oppgaven i .tex-format.
- **main.pdf** Oppgaven i pdf-format (B5).
- **main_a4.pdf** Oppgaven i pdf-format (A4).
- **sysdoc.pdf** Kodedokumentasjon i pdf-format.

Bibliografi

[1] Bullet physics library. `http://bulletphysics.org`.

[2] Doxygen. `http://www.doxygen.org`.

[3] Eigen. `http://eigen.tuxfamily.org`.

[4] Opengl. `http://www.opengl.org/`.

[5] Robot operating system. `http://www.ros.org`.

[6] Ronald C. Arkin. *Behavior-Based Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 5 1998.

[7] O. Brock B. Burns. Toward optimal configuration space sampling. *Robotics: Science and Systems*, 2005.

[8] Thomas H. Cormen. *Introduction to Algorithms*. The MIT Press, 3 edition, 2001.

[9] K. Harada J.-C. Latombe K. Hauser, T. Bretl. Using motion primitives in probabilistic sample-based planning for humanoid robots. *Springer Tracts in Advanced Robotics*, 47, 2008.

[10] O. Khatib. A unified approach to motion and force control of robot manipulators: The operational space formulation. *International Journal of Robotics and Automation*, 3:43–53, 1987.

[11] T. Kröger. *On-Line Trajectory Generation in Robotic Systems*. Springer, 2010.

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

[13] T.Siméon L. Jaillet. A prm-based motion planner for dynamically changing environments. *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2:1606–1611, September 2004.

[14] O. Khatib L. Sentis. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robots*, 2:505–518, 2005.

[15] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[16] R. A. Grupen M. Huber. A feedback control structure for on-line learning tasks. *Robotics and Autonomous Systems*, 22:303–315, 1997.

[17] M. Vidyasagar Mark W. Spong, Seth Hutchinson. *Robot Modeling And Control*. Wiley, 2006.

[18] O. Khatib O. Brock. Elastic strips: A framework for motion generation in human environments. *The International Journal of Robotics Research*, 21:1031–1052, 12 2002.

[19] Jae-Heung Park Oussama Khatib, Luis Sentis. A unified framework for whole-body humanoid robot control with multiple constraints and contacts. *Robotics and Automation*, pages 2641 – 2648, 2006.

[20] Patrick G. Xavier R. Gayle, Kristopher R. Klingler. Lazy reconfiguration forest (lrf) - an approach for motion planning with multiple tasks in dynamic environments. *IEEE International Conference on Robotics and Automation*, 2007.

[21] David E. Orin et. al. Roy Featherstone. *Springer Handbook of Robotics*. Springer, 2008.

[22] J. Borenstein Y. Koren. Potential field methods and their inherent limitations for mobile robot navigation. *Proceedings of the IEEE Conference on Robotics and Automation*, 1:1398–1404, 1991.

[23] O. Brock Y. Yang. Elastic roadmaps - motion generation for autonomous mobile manipulation. *Autonomous Robots*, 28:113–130, 2009.