Robotnavigation

Rasmus Friis Kjeldsen
razzle@diku.dk

Marts 2006

Indhold

Tabeller

Figurer

Kapitel 1

Indledning

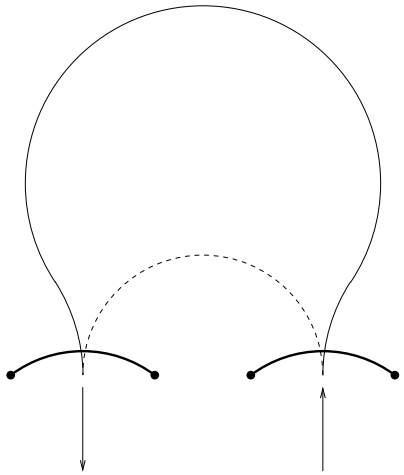
1.1 Motivation

Til deltagelsen i RoboCup-konkurrencen i 2004, jf. [rob, 2004], på Danmarks Tekniske Universitet havde jeg bygget en robot kaldet Murphy med køreegenskaber som en bil. Den havde baghjulstræk og styrbare forhjul. I konkurrencen gives der point for hver gang en port passeres. To af disse porte stod parallelt tæt ved siden af hinanden, se figur 1.1. Den oplagte måde at køre igennem dem på var at dreje skarpt 180°, når den første port var passeret. Dette sving var dog skarpere, end Murphy kunne dreje. En løsning blev fundet, nemlig ved at lade robotten dreje lidt til den modsatte side først, for så bagefter at dreje til den side porten stod på. En anden løsning kunne have været at lade robotten foretage en trepunktsvending, når den første port var passeret.

Denne problemstilling blev inspiration til opgaven [Rode and Kjeldsen, 2004], i hvilken en simulator blev udviklet, som kan simulere en bil-agtig robots bevægelse. Desuden løste opgaven problemet med at finde den korteste vej mellem to positioner, som det er muligt for en bil-agtig robot, der kun kører forlæns, at følge.

Succesen med opgaven, samt det indblik opgaven gav i problemstillingen om ruteplanlægning for en bil-agtig robot, gav inspirationen til dette speciale.

Et andet projekt, nemlig [Kjeldsen et al., 2001], som beskæftigede sig med positionsestimering ud fra visuel landmærkegenkendelse, var også en del af inspirationen til denne opgave. Her blev en robot placeret på en bane med seks landmærker. Robotten brugte et kamera til at måle afstanden til landmærkerne, og robottens position kunne beregnes ud fra triangulering.



Figur 1.1: Placeringen af to porte ved RoboCup 2004. Med stiplet linje er den oplagte, men for Murphy umulige, rute illustreret. Med fuld optrukket linje er illustreret en rute, som det er muligt for Murphy at følge.

1.2 Mål

Det primære mål for specialet er at kunne få robotten Murphy til at køre fra ét lokale til et andet lokale fx på DIKU, dog kun på samme etage. I det følgende vil dette mål blive nærmere specificeret, og det bliver også specificeret, hvordan målet skal opfyldes.

Målet er at få en bil-agtig robot til at køre fra en startposition til en målposition, uden at støde ind i forhindringer. Ved en position forstås både sted og orientering. Som hjælp får robotten et kort over forhindringerne samt placeringen af landmærker. Det antages, at forhindringerne er statiske, det vil sige, at de ikke flytter sig. Opgaven kan på en naturlig måde opdeles i tre dele, nemlig ruteplanlægning, positionsestimering og styring af robotten.

Ruteplanlægningen vanskeliggøres af robottens ikke-holonome egenskaber, som er illustreret på figur 1.1. Først vil ruteplanlægning uden hensyntagen til forhindringer blive beskrevet. Dernæst gennemgås kollisionsbestemmelse, og efter en diskussion af forskellige metoder til ruteplanlægning, bliver den valgte metode fremlagt.

Positionsestimeringen tager udgangspunkt i dead reckoning. Denne har dog en akkumulerende fejl, men ved at sammenholde med målinger af afstand og vinkel til landmærkerne, kan estimeringen forbedres. Til sammenholdningen af disse delvist redundante data anvendes partikelfiltrering.

Styringen af robotten involverer både styring af robottens hastighed og af dens forhjul, så en planlagt rute kan følges.

For at kunne løse opgaven er robotten Murphy blevet opgraderet. Konstruktionen af Murphy vil ikke blive betragtet som en del af opgaven, hvorfor en beskrivelse heraf er henlagt til appendiks A.

1.2.1 Et uforudset problem

Under dette speciales udarbejdning opstod det uforudsete problem, at jeg fik seneskedehindebetændelse i begge arme. Heldigvis havde jeg forinden nået at implementere en god del af programmet efter filosofien: Implementer hvert delproblem indtil det virker, eller til at det er oplagt, at det kommer til at virke. Den sidste afpudsning og implementation skulle foregå, når jeg vidste, hvor lang tid jeg havde til det. Dette førte til en svær afvejelse. Skulle jeg forsøge at få hele løsningen færdigimplementeret, eller måtte jeg stoppe med programmeringen og koncentrere mig om at få skrevet specialeteksten. Da der efter lang tids behandling af armene kun skete små fremskridt, og det viste sig vanskeligt og meget tidskrævende at diktere programtekst til en person, som tastede for mig, måtte jeg vælge at koncentrere mig om specialets tekst. Som følge heraf er det ønskede mål kun delvist implementeret. Specialet er herved blevet noget mere teoretisk end først planlagt, og afprøvningen af programmet er blevet væsentligt mindre fyldestgørende end det var tiltænkt.

1.2.2 Indholdet af den vedlagte cd-rom

Til dette speciale hører en cd-rom med følgende indhold:

- `rapport/rapport.pdf`: Dette speciale. Mange af specialets figurer er farvelagt og kan kun forstås til fulde, hvis de ses i farver. Hvis denne tekst er udskrevet i sort/hvid, anbefales det at se figurerne i cd-rommens udgave. Specialet kan også findes på internettet på adressen: http://www.reblag.dk/robot/speciale.
- `data`: Billeder brugt i afprøvningen.
- `docs`: pdf-filer af de fleste af de refererede artikler, som ikke kan findes frit på internettet.
- `matlab`: Testprogrammer skrevet i Matlab, som hovedsaglig er brugt til at generere specialets figurer.

- `src`: Kildeteksten til programmet, inkl. dokumentation i html-format i filen `src/html/index.html`.
- `avr`: Kildeteksten til programmet, som kører på styreelektronikens indlejrede processor.

1.3 Specialets opbygning

Først bliver ruteplanlægningen beskrevet i kapitel 2. Dernæst bliver positionsestimeringen gennemgået i kapitel 3. Kapitel 4 omhandler styring af robotten, og kapitel 5 beskriver implementationen af programmet. Til sidst giver kapitel 6 en konklusion på specialet.

Bagerst i specialet findes en litteraturliste på side 122. Appendiks A beskriver robotten Murphy, appendiks B opsummerer en række af de anvendte definitioner, appendiks C beskriver den fysiske udformning af landmærkerne, og appendiks D indeholder arbejdsbeskrivelsen for specialet.

Kapitel 2

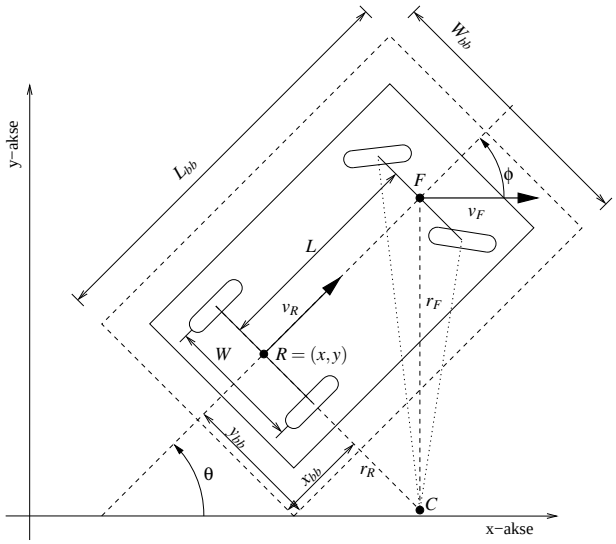
Ruteplanlægning

Det overordnede mål for ruteplanlægningen er at finde en rute fra et sted på et kort til et andet sted på kortet, som det er muligt for Murphy at følge, uden at støde ind i noget undervejs. Kan dette ikke lade sig gøre, må ruteplanlægningen vende tilbage med en fejlmeddelelse. Desuden vil det være ønskværdigt, at den fundne rute er, om ikke optimal (i en eller anden forstand), så i hvert fald udformet, så det vil tage Murphy et rimeligt tidsrum at følge den fra start til slut.

I det følgende vil den ovenstående noget åbne definition af ruteplanlægningsproblemet blive nærmere specificeret og konkretiseret.

Først opstilles en model af Murphy. Modellen vil blive grundlaget for ruteplanlægningen, ikke den fysiske robot Murphy. Denne model vil i det følgende blive refereret til som en *bil-agtig* robot, se figur 2.1. En bil-agtig robot har to baghjul, der sidder fast monteret på bagakslen, dvs. så de kan dreje rundt, men deres vinkel er fast. Desuden har den to forhjul, som også kan dreje rundt, og deres vinkel kan styres. ϕ angiver styrehjulenes vinkel. Man kan tænke sig denne vinkel som vinklen af et enkelt styrehjul monteret midt på forakslen, dvs. i punktet F. I dette kapitel modelleres robotten som havende kun et sådant styrehjul. Dette svarer til, at de to forhjul styres af en Ackermannstyring. En Ackermannstyring lader forhjulene under et sving følge cirkelbuer med lidt forskellig radius, så forhjulene ruller uden at skride ud, jf. [Lund, 2000a, p. 46], hvilket er antydet på figuren. I kinematikken for modellen antages nemlig, at alle hjul ruller uden at skride ud. Vinklen ϕ kan altså kontrolleres, men inden for et begrænset interval $-\phi_{max} \leq \phi \leq \phi_{max}, \phi_{max} < \frac{\pi}{2}$. Der antages ingen grænse for vinkelhastigheden $\dot{\phi}$. Derudover kan robottens hastighed også kontrolleres. Med hastigheden menes hastigheden v_R af punktet R i robottens længderetning. Der antages intet om fortegn og størrelse af v_R. Hastigheden af R ortogonalt på længderetningen vil altid være 0, da det jo blev antaget, at alle hjul ruller uden at skride. Robotten antages desuden kun at bevæge sig i

planet dvs. $\mathbb{R}^2$.



Figur 2.1: Model af en bil-agtig robot. Løst tegnet efter [Rode and Kjeldsen, 2004, figur 1]. Robottens position er givet ved punktet R, samt af orienteringen θ. L er afstanden mellem for- og bagakslen og W er afstanden mellem baghjulene. ϕ angiver styrevinklen, og v_R angiver robottens hastighed. L_{bb} og W_{bb} angiver længde og bredde og $(x_{bb},\ y_{bb})$ angiver positionen af robottens omgivende rektangel. Dette bruges til at definere det areal, robotten udfylder, til brug for kollisionsbestemmelsen.

En sådan definition af en bil-agtig robot kaldes også en Reeds og Shepp bil pga. deres artikel [Reeds and Shepp, 1990], og den er en udvidelse af Dubins' bil, jf. [Dubins, 1957], der kun kan køre forlæns, dvs. hvor $v_R \geq 0$. Helt formaliseret kan en bil-agtig robot defineres ved følgende differentialligninger, som beskriver dens kinematik, jf. [Rode and Kjeldsen, 2004]:

$$\begin{aligned} \dot{x} &= v_R \cos\theta \\ \dot{y} &= v_R \sin\theta \\ \dot{\theta} &= v_R \frac{\tan\phi}{L}, \qquad -\phi_{max} \leq \phi \leq \phi_{max} \end{aligned} \tag{2.1}$$

Ud fra definitionen af den bil-agtige robot, kan robottens position og positur, også kaldet robottens *konfiguration*, nu defineres. Med positur menes, at hvis robotten fx var en gribearm, så udgøres posituren af vinklen på dens led, og om dens gribehånd er åben eller lukket. Ifølge modellen har en bil-agtig robot fire frihedsgrader,

nemlig: Dens koordinater i planet $(x,\ y)$, dens orientering θ og vinklen af dens forhjul ϕ. Sidstnævnte vil dog blive ignoreret som frihedsgrad (men kun som frihedsgrad). Når robotten skal beskrives som værende et bestemt sted, er dens koordinater og orientering vigtig, i hvilken retning dens forhjul peger er mindre vigtigt.

Robottens konfiguration vil derfor overalt i denne tekst blive betegnet som dens *position*, og hermed menes triplen $(x,\ y,\ \theta)$. En bil-agtig robot siges således at have konfigurationsrummet $\mathcal{C} = \mathbb{R}^2 \times [0;\ 2\pi[$.

En bil-agtig robot har altså tre frihedsgrader, men kun to parametre som kan kontrolleres. Det er nu ikke i sig selv det, der gør ruteplanlægning for bil-agtige robotter indviklet. Tag fx en differentialstyret robot, som har en kinematik som et bæltekøretøj. Den har også tre frihedsgrader og to parametre som kan kontrolleres. At finde fra en position til en anden for en differentialstyret robot kan reduceres til tre bevægelser: Rotér indtil robotten peger mod målpositionen, kør ligeud til målpositionens koordinater er nået og rotér til sidst indtil målpositionens orientering er nået. Dette er samtidig den euklidisk korteste, men ikke nødvendigvis hurtigste rute, jf. [Balkcom and Mason, 2002].

Det der gør ruteplanlægningen for bil-agtige robotter indviklet, er de kinematiske begrænsninger på robottens bevægelse. Skal robotten fx ændre sin vinkel, må den også ændre sin position. Robotten siges at være ikke-holonom. For en mere formel definition af ikke-holonome robotter, se [LaValle, 2004, kap. 15].

Afsnit 2.1 beskriver, hvordan en rute, endda den korteste, mellem to positioner kan findes for en bil-agtig robot, når der ikke tages hensyn til forhindringer. Afsnit 2.2 beskriver robottens kort over omgivelserne. I afsnit 2.3 konstateres det, hvordan det kan afgøres, om robotten vil støde ind i de på kortet angivne forhindringer, hvis den følger en given rute. Til sidst bliver resultaterne af de tre ovennævnte afsnit brugt i afsnit 2.5 til vha. kortet at finde en rute fra en position til en anden, som robotten kan følge, uden at støde ind i de på kortet angivne forhindringer.

2.1 Den korteste rute mellem to positioner uden forhindringer

Dette afsnit beskriver, hvorledes den korteste rute mellem to positioner for en bil-agtig robot findes, når der *ikke* tages hensyn til forhindringer, og der udledes formler for en delmængde af problemet. Desuden udledes formler for en løsning, som nok ikke giver den korteste rute, men alligevel viser sig nyttig.

En helt centralt egenskab ved en bil-agtig robot er i denne sammenhæng robot-

tens mindste drejeradius ρ_{min}. Dette grunder i resultaterne i artiklen af Dubins [Dubins, 1957], som omhandler kurver med minimal længde med en begrænset krumning, samt artiklen af Reeds og Shepp [Reeds and Shepp, 1990], der med udgangspunkt i Dubins' arbejde udregner optimale ruter for en bil-agtig robot, som både kan køre forlæns og baglæns.

2.1.1 Notation og klasser

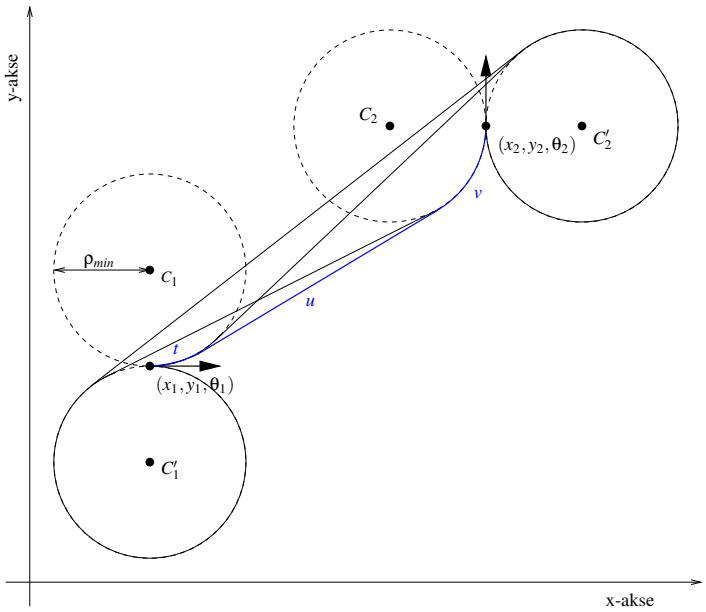
Reeds og Shepp viser, at den korteste rute mellem to positioner for en bil-agtig robot altid findes indenfor en vis klasse af ruter. Disse ruter består af sammensætningen af to cirkelbuer, en ret linje og to cirkelbuer, i kort notation $CCSCC$. Her står hvert C for en cirkelbue med radius ρ_{min} og S står for et segment af en ret linje. Længden af en eller flere af de fem kurvestykker kan være nul. Den resulterende kurve er naturligvis kontinuert, dvs. kurvestykkerne ligger i forlængelse af hinanden. Kurven er én gang differentiabel på nær i punkter, hvor der skiftes kørselsretning. Kørselsretningen kan desuden tilføjes med plus eller minus for forlæns hhv. baglæns, og hvis et kurvestykkes længde er nul, kan det udelades, fx $C^+C^-C^+$. Yderligere kan hvert kurvestykkes længde valgfrit tilføjes. Med denne præciserede notation indskrænker Reeds og Shepp løsningsrummet, der skal afsøges, til følgende mere specifikke klasser:

$$\begin{array}{llll} C^+C^-C^+, & C^+C^-C^-, & C^+S^+C^+, & C^+C_u^+C_u^-C^-, \\ C^+C_u^-C_u^-C^+, & C^-C_{\pi/2}^+S^+C^+, & C^-C_{\pi/2}^+S^+C_{\pi/2}^+C^-, & \\ C^+C^+C^-, & C^-S^+C_{\pi/2}^+C^+ & & \end{array} \tag{2.2}$$

samt de ovenstående klasser med samtlige fortegn vendt om, i alt atten klasser.

CS-notationen kan yderligere konkretiseres, så r angiver en cirkelbue med centrum til højre for cirkelbuens startposition set i forhold til startpositionens retning. Tilsvarende angiver l en cirkelbue med centrum til venstre, og s angiver en lige linje. Disse r, l og s kan specificeres yderligere med et plus eller et minus som angiver kørselsretningen, og med en længdeangivelse. Et eksempel herpå er $l_t^+ s_u^+ l_v^+$. Denne notation giver sammen med en udgangsposition og en given mindste drejeradius ρ_{min} en komplet kurvebeskrivelse, se figur 2.2. Slutpositionen vil så være implicit givet.

De enkelte delkurver l, r og s vil i det følgende blive betegnet som *segmenter*, og deres tilknyttede længder vil blive betegnet som *segmentlængder*. En kurve bestående af en række sammenhængende segmenter vil i det følgende blive betegnet som en *rute*.



Figur 2.2: En illustration af klassen $C^+S^+C^+$ med startposition $p_1 = (x_1, y_1, \theta_1)$ og slutposition $p_2 = (x_2, y_2, \theta_2)$. Klassen består af fire forskellige ruter, nemlig ruterne $l^+s^+l^+$ (som er den korteste, mere specifikt $l_t^+ s_u^+ l_v^+$, markeret med blåt), $l^+s^+r^+$, $r^+s^+l^+$ og $r^+s^+r^+$.

Robottens mindste drejeradius ρ_{min} kan findes ud fra robottens længde L, samt robottens maksimale styrevinkel ϕ_{max}. Ud fra figur 2.1 kan drejeradiussen r_R for midtpunktet R af bagakslen udledes ved trekantsbetragtning:

$$\tan\phi = \frac{L}{r_R} \qquad \Rightarrow \qquad r_R = \frac{L}{\tan\phi} \tag{2.3}$$

Ud fra formel 2.3 kan den mindste drejeradius findes som:

$$\rho_{min} = \frac{L}{\tan(\phi_{max})} \tag{2.4}$$

2.1.2 Normering og symmetri

Som det er antydet på figur 2.2, kan de atten klasser, der skal undersøges, give anledning til en voldsom mængde kombinationer, og at opstille formler til at løse dem alle vil være en helt uoverkommelig opgave. Heldigvis kan problemet forenkles, jf. [Reeds and Shepp, 1990, afsnit 8].

For det første er det tilstrækkeligt at kigge på ruter, hvor startpositionen er $p_1 = (0,0,0)$, og hvor drejeradius er 1, uden at miste generalitet. Dette skyldes, at der findes en entydig transformation mellem det generelle tilfælde med $p_1 = (x_1, y_1, \theta_1)$, $p_2 = (x_2, y_2, \theta_2)$ og drejeradius ρ og det normerede tilfælde med $p_1 = (0,0,0)$, $p_2 = (x, y, \theta)$ og drejeradius 1. Transformationen af p_1 og p_2 foretages som følger:

$$t_i = p_i - p_1 \tag{2.5}$$
$$t_i = rot(t_i, -\theta_1) \tag{2.6}$$
$$t_i = \frac{1}{\rho} t_i \tag{2.7}$$

Her betegner t_i de transformerede positioner. $rot(p, \alpha)$ betegner roteringen af positionen $p = (x, y, \theta)$ α radianer omkring punktet $(0,0)$, inklusiv rotation af orienteringen, dvs. α adderes til θ. ρ betegner radius af kurvens cirkelbuer, hvor $\rho = \rho_{min}$ for robotten. I formel 2.5 og 2.7 berøres kun x- og y-koordinaten, ikke orienteringen.

Når den ønskede rute er fundet i det normerede tilfælde kan den omvendte transformation give ruten i det generelle tilfælde. Den omvendte transformation foretages ved:

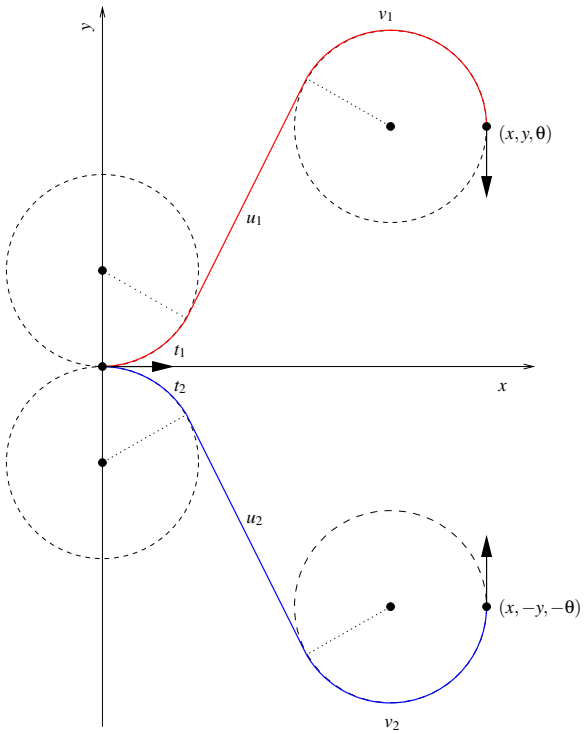
- Multiplicer alle segmentlængder med ρ.
- Sæt rutens startposition til den oprindelige startposition $p_1 = (x_1, y_1, \theta_1)$.

Den fundne rute vil nu gå fra $p_1 = (x_1, y_1, \theta_1)$ til $p_2 = (x_2, y_2, \theta_2)$.

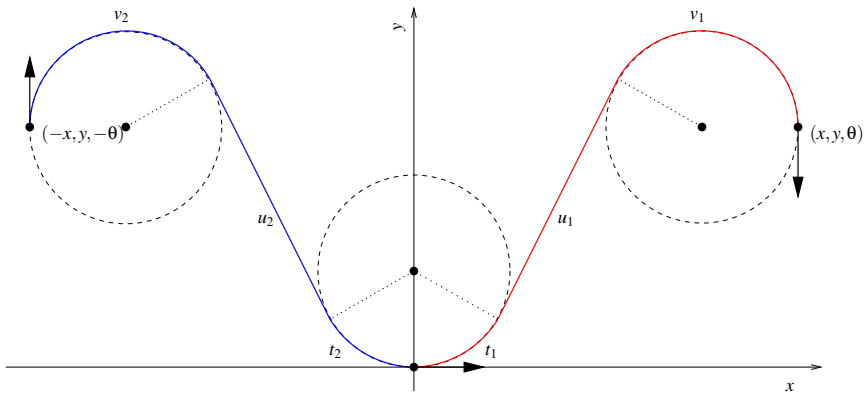
Normeringen giver en forenkling ved opstillingen af formlerne, men reducerer ikke antallet af kombinationer. Ud fra symmetribetragtninger, kan antallet af kombinationer heldigvis reduceres betragteligt.

Spejlingssymmetri, eller spejling om x-aksen, har den egenskab, at hvis man kender en rute fra fx $p_0 = (0,0,0)$ til $p_1 = (x, y, \theta)$, kan en rute fra p_0 til $p_2 = (x, -y, -\theta)$ findes, ved at ombytte hver forekomst af r med l og hver forekomst af l med r. Har man fx en formel, som kan udregne ruten $r_t s_u l_v$ fra p_0 til p_1, kan den også bruges til at finde ruten $l_t s_u r_v$ fra p_0 til p_1 ved at give formlen $(x, -y, -\theta)$ som slutpunkt i stedet for p_1, og derefter bytte om på r og s i den fundne rute. Se figur 2.3.

Tidssymmetri, eller spejling om y-aksen, har en tilsvarende egenskab som spejlingssymmetri, blot skiftes fortegn på kørselsretningen i rutens segmenter. Har man fx en formel som kan udregne ruten $l^+ s^+ r^+$ fra $p_0 = (0,0,0)$ til $p_1 = (x, y, \theta)$, kan den også bruges til at finde ruten $l^- s^- r^-$ fra p_0 til p_1 ved at give formlen



Figur 2.3: Spejlingssymmetrien: Med rødt: Kurven $l^+_{t_1} s^+_{u_1} r^+_{v_1}$ fra $(0,0,0)$ til (x,y,θ). Med blåt: Kurven $r^+_{t_2} s^+_{u_2} l^+_{v_2}$ fra $(0,0,0)$ til $(x,-y,-\theta)$.



Figur 2.4: Tidssymmetrien: Med rødt: Kurven $l^+_{t_1} s^+_{u_1} r^+_{v_1}$ fra $(0,0,0)$ til (x,y,θ). Med blåt: Kurven $l^-_{t_2} s^-_{u_2} r^-_{v_2}$ fra $(0,0,0)$ til $(-x,y,-\theta)$.

$(-x, y, -\theta)$ som slutpunkt i stedet for p_1, og derefter skifte fortegn på kørselsretningerne i den fundne rute. Se figur 2.4.

For mere om normering og symmetrier, se [Rode and Kjeldsen, 2004, afsnit 4.3] og [Reeds and Shepp, 1990, afsnit 8].

Reeds og Shepp postulerer elleve formler, som sammen med spejlingssymmetrien, tidssymmetrien og en tredie symmetri "bakkesymmetrien" (som ikke bruges her) giver den korteste rute inden for samtlige de atten ruteklasser, hvori den korteste rute kan ligge, jf. tabel 2.2. Desværre indeholder i hvert fald to ud af de tre første formler nogle småfejl, nemlig [Reeds and Shepp, 1990, formel 8.2 og 8.3]. Disse formler blev udledt og dermed rettet i [Rode and Kjeldsen, 2004, afsnit 4.3], som dog kun beskæftiger sig med en robot der kan køre forlæns. I de følgende afsnit vil de anvendte formler blive udledt, incl. nogle mindre forbedringer, for både forlæns og baglæns kørsel. Hver formel udleder segmentlængderne for ruten fra $p_1 = (0,0,0)$ til $p_2 = (x, y, \theta)$.

2.1.3 Formler for kurven $l_t s_u^+ l_v$

Reeds og Shepp anvender to hjælpefunktioner. R defineres som transformationen til polære koordinater, dvs. når der skrives $(r, \theta) = R(x, y)$, er r og θ givet ved $r\cos\theta = x$, $r\sin\theta = y$, $r \geq 0$, $0 \leq \theta < 2\pi$. Specielt defineres $R(0,0) = (0,0)$. Dette er et vigtigt særtilfælde, som Reeds og Shepp glemmer at tage højde for. M defineres som $\phi = M(\theta)$, hvis $\phi \equiv \theta \bmod 2\pi$, $-\pi \leq \phi < \pi$.

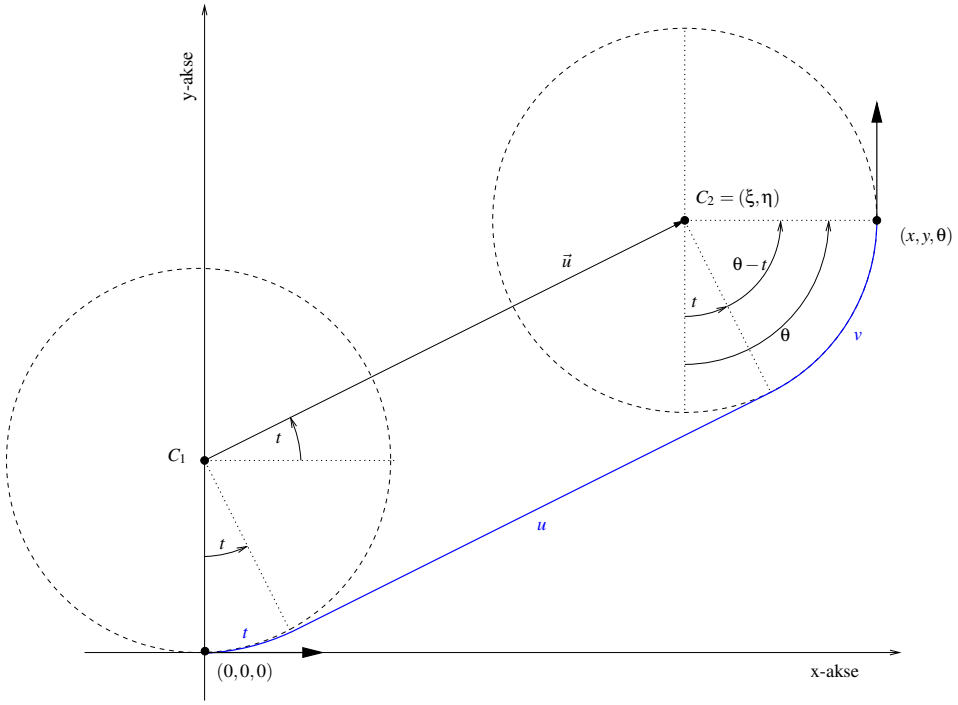
I det følgende anvender de så M og R i udledningen af $l_t^+ s_u^+ l_v^+$, læg mærke til de tre plusser, jf. [Reeds and Shepp, 1990, formel 8.1]. Først udregnes koordinaterne af slutcirkelens centrum $C_2 = (\xi, \eta)$. C_2 er altså centrum af den cirkel, som l_v ligger på, se figur 2.5:

$$\begin{aligned} \xi &= x - \sin\theta \\ \eta &= y + \cos\theta \end{aligned} \tag{2.8}$$

Herfra findes:

$$(u, t) = R(\xi, \eta - 1) \tag{2.9}$$

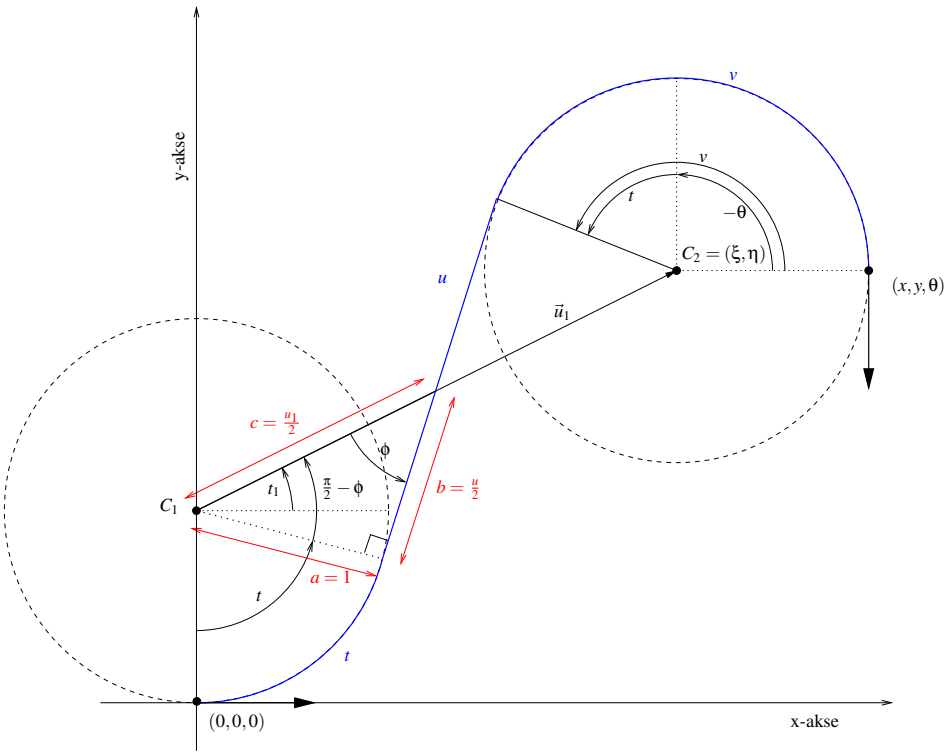
$$v = M(\theta - t) \tag{2.10}$$



Figur 2.5: Geometrisk udledning af længderne t, u og v for kurven $l_t s_u^+ l_v$ med startposition $(0,\ 0,\ 0)$ og slutposition $(x,\ y,\ \theta)$. Efter [Rode and Kjeldsen, 2004].

Problemet er så, at t og v kan blive negative jf. definitionen af R og M, og formel 2.10 kan derfor ikke give segmentlængderne for $l_t^+ s_u^+ l_v^+$. Her tror jeg, at det er gået lidt for hurtigt for forfatterne, da de skrev artiklen. Formel 2.10 giver nemlig med den anvendte definition af R og M den korteste rute inden for hele klassen $l_t s_u^+ l_v$. Det har helt sikkert været forfatternes intention, de har bare skrevet noget andet. Fejlen går igen i de fleste af de resterende ti formler [Reeds and Shepp, 1990, formel 8.2 - 8.4, 8.6, 8.9 - 8.11]. Fejlen er helt uden betydning, når man vil finde den korteste rute blandt alle de atten ruteklasser for en robot, som både kan køre forlæns og baglæns, men hvis robotten kun kan køre forlæns (eller kun baglæns for den sags skyld), og man derfor ønsker det specifikke resultat af hver formel, giver formlerne ikke de ventede resultater.

2.1.4 Formler for kurven $l_t s_u^+ r_v$



Figur 2.6: Geometrisk udledning af længderne t, u og v for kurven $l_t s_u^+ r_v$ med startposition $(0,\ 0,\ 0)$ og slutposition $(x,\ y,\ \theta)$. Efter [Rode and Kjeldsen, 2004].

Centrum af slutcirkelen $C_2 = (\xi, \eta)$ findes, jf. figur 2.6, som:

$$\begin{aligned} \xi &= x + \sin\theta \\ \eta &= y - \cos\theta \end{aligned} \tag{2.11}$$

Heraf findes:

$$(u_1, t_1) = R(\xi, \eta - 1) \tag{2.12}$$

Hvis $u_1 < 2$ findes ingen løsning. Ellers findes u først ud fra pythagoras:

$$\begin{aligned} \left(\frac{u_1}{2}\right)^2 &= \left(\frac{u}{2}\right)^2 + 1^2 \quad \Rightarrow \\ u &= \sqrt{u_1^2 - 4} \end{aligned} \tag{2.13}$$

Nu kan ϕ findes ud fra trekantsbetragtning, og t og v følger af ϕ:

$$\phi = \begin{cases} \frac{\pi}{2} & \text{hvis } u = 0 \\ \arctan\left(\frac{2}{u}\right) & \text{hvis } u \neq 0 \end{cases} \tag{2.14}$$

$$t = M(t_1 + \phi) \tag{2.15}$$

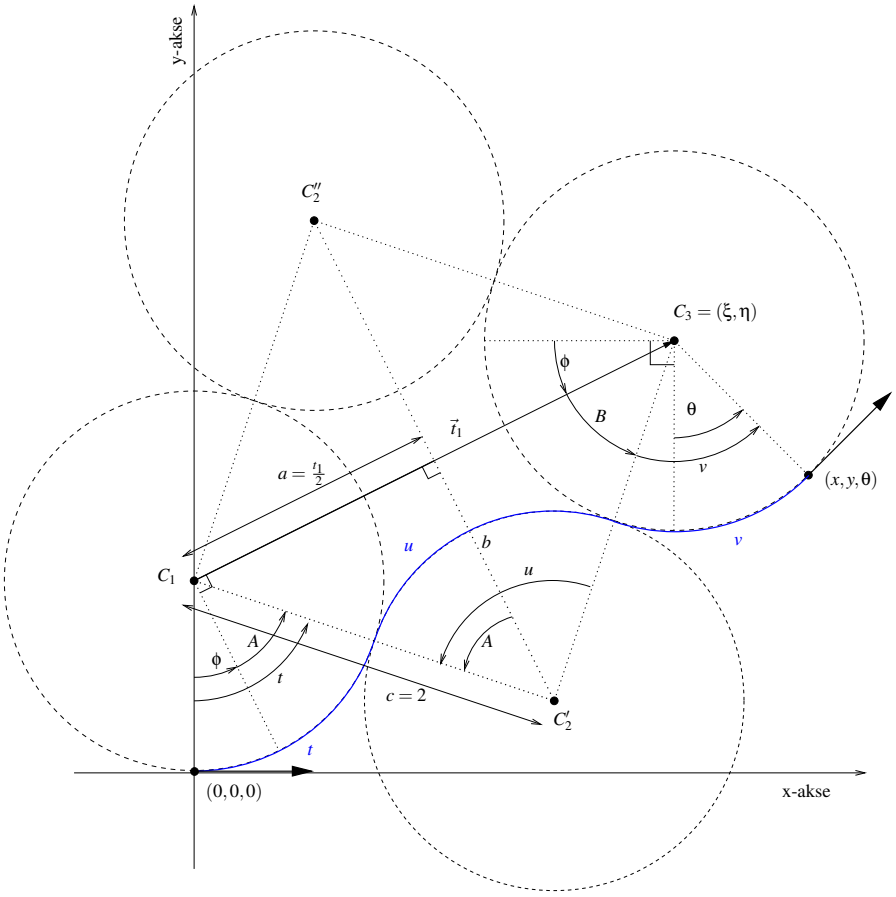
$$v = M(t - \theta) \tag{2.16}$$

2.1.5 Formler for kurven $l_t r_u l_v$

Centrum af slutcirkelen $C_3 = (\xi, \eta)$ findes, jf. figur 2.7, som:

$$\begin{aligned} \xi &= x - \sin\theta \\ \eta &= y + \cos\theta \end{aligned} \tag{2.17}$$

Herfra findes:



Figur 2.7: Geometrisk udledning af længderne t, u og v for kurven $l_t r_u l_v$ med startposition $(0,\ 0,\ 0)$ og slutposition $(x,\ y,\ \theta)$. Efter [Rode and Kjeldsen, 2004].

$$(t_1, \phi) = R(\xi, \eta - 1) \tag{2.18}$$

Hvis $t_1 > 4$ findes ingen løsning. Ellers findes A ved trekantsbetragtning:

$$A = \arcsin\left(\frac{t_1}{4}\right) \quad \vee \quad A = \pi - \arcsin\left(\frac{t_1}{4}\right) \tag{2.19}$$

De to værdier af A indsættes i det nedenstående, hvilket giver to løsninger:

$$\begin{aligned} t &= M(A + \phi) && (2.20)\\ u &= 2A && (2.21)\\ v &= M(\theta + A - \phi) && (2.22) \end{aligned}$$

2.1.6 Formler for kurven $s_t l_u s_v$

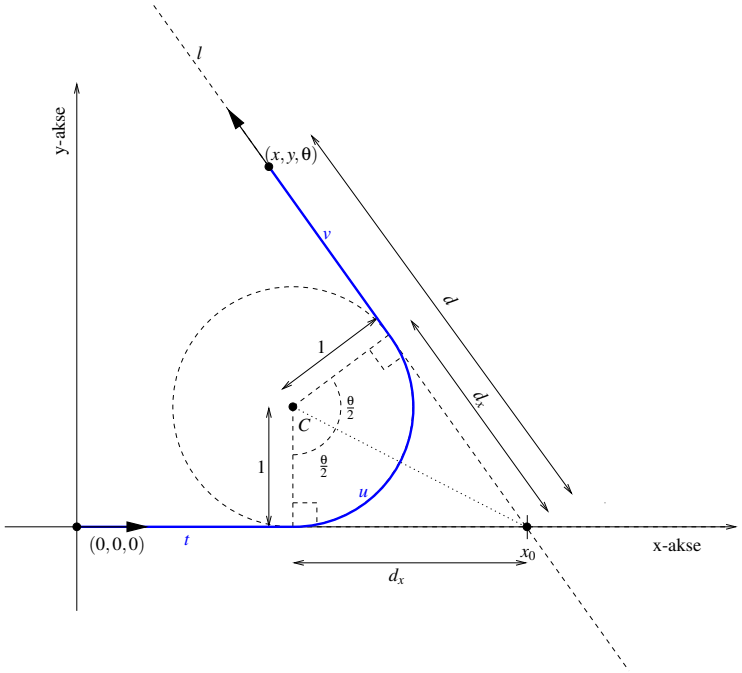
Kurven sls er ikke blandt de atten klasser nævnt i formel 2.2, men formlen for kurven udledes her, da den vil blive anvendt senere, nemlig i afsnit 2.5.

Længderne t, u og v udledes for $s_t l_u s_v$ ved hjælp af figur 2.8. Først findes stedet x_0 på x-aksen, hvor linjen l gennem $p = (x, y, \theta)$ skærer x-aksen. Ligningen for linjen l, som har hældningen $\tan\theta$:

$$\begin{aligned} Y &= aX + b \quad \Rightarrow \quad b = Y - aX : \\ b &= y - x\tan\theta && (2.23) \end{aligned}$$

Ligningen for x-aksen $Y = 0$ giver sammen med ligningen for l, stedet x_0:

$$\begin{aligned} aX + b &= 0 \quad \Rightarrow \quad X = \frac{-b}{a} : \\ x_0 &= \frac{x\tan\theta - y}{\tan\theta} \quad \Rightarrow \\ x_0 &= x - \frac{y}{\tan\theta}, \quad \text{for } \tan\theta \neq 0 && (2.24) \end{aligned}$$



Figur 2.8: Illustration af en kurve af formen $s_t l_u s_v$. $(0,0,0)$ er startpositionen, (x,y,θ) er slutpositionen, og det er de tre længder t, u og v, der ønskes fundet.

For $\tan\theta = 0$ haves altså ingen løsning i formel 2.24. Dette svarer til tilfældet, hvor $\theta = 0$ eller $\theta = \pi$. For $\theta = 0$ findes en løsning kun, hvis $y = 0$, nemlig fx $t = x$ og $u = v = 0$. For $\theta = \pi$ findes kun en løsning, hvis $y = 2$ (cirklens diameter), nemlig fx $t = x$, $u = \pi$ og $v = 0$.

Nu findes afstanden d_x vha. trekantsbetragtning:

$$\tan\left(\frac{\theta}{2}\right) = \frac{d_x}{1} \quad \Rightarrow \quad d_x = \tan\left(\frac{\theta}{2}\right) \tag{2.25}$$

Afstanden d er givet ved:

$$d = \sqrt{(x - x_0)^2 + y^2} \tag{2.26}$$

Længderne t, u og v kan nu findes som:

$$t = x_0 - d_x \tag{2.27}$$

$$u = M(\theta) \tag{2.28}$$

$$v = \begin{cases} \text{sign}(y)d - d_x, & \text{for } 0 < \theta < \pi \\ -\text{sign}(y)d - d_x, & \text{for } \pi < \theta < 2\pi \end{cases} \tag{2.29}$$

Her er funktionen $\text{sign}(y)$ defineret som:

$$\text{sign}(y) = \begin{cases} 1, & \text{hvis } y > 0 \\ -1, & \text{hvis } y < 0 \end{cases} \tag{2.30}$$

2.1.7 Delkonklusion og mulige forbedringer

I dette hovedafsnit er formlerne for kurverne $l_t s_u^+ l_v$, $l_t s_u^+ r_v$, $l_t r_u l_v$ og $s_t l_u s_v$ blevet udledt. I alt giver dette sammen med spejlings- og tidssymmetrien formlerne for klasserne CSC, CCC og SCS, jf. tabel 2.1.

De fundne formler kan, sammen med symmetrierne og normeringen, finde samtlige kurver på formen CSC, CCC og SCS mellem to vilkårlige positioner. For at finde

Formel	Spejling	Tid	Spejling+tid	Samlet	Klasse
ls^+l	rs^+r	ls^-l	rs^-r	lsl,rsr	
ls^+r	rs^+l	ls^-r	rs^-l	lsr,rsl	CSC
lrl	rlr	$-$	$-$	lrl,rlr	CCC^*
sls	srs	$-$	$-$	sls,srs	SCS

Tabel 2.1: Kombinationer af spejlings- og tidssymmetri for kurverne ls^+l, ls^+r, lrl og sls, og de resulterende klasser.
* Da lll svarer til lr_0l_0 osv.

den *korteste* rute indenfor klasserne CSC og CCC (SCS udelades, den anvendes til andet formål, se afsnit 2.5) må man altså først beregne længderne t, u og v for samtlige kurver indenfor klasserne, og så vælge den korteste.

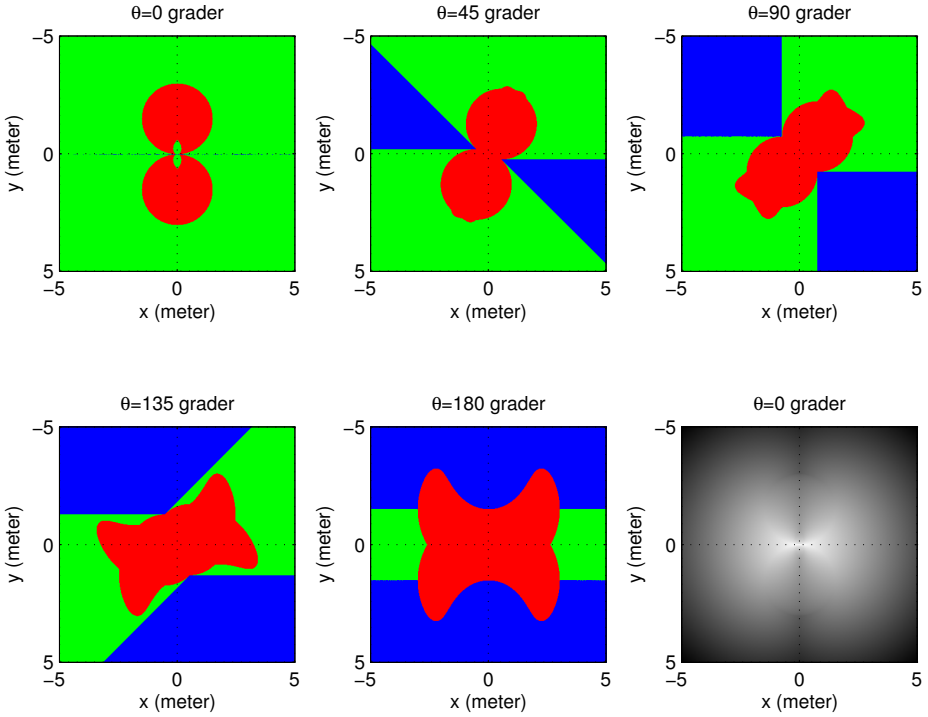
Denne metode definerer en afstandsfunktion dvs. en metrik $\| p_1\ p_2 \|$ på konfigurationsrummet. På figur 2.9 nederst til højre ses evalueringen af afstandsfunktionen for tilfældet $\| (0,0,0)\ (x,y,0) \|$. Figuren giver også en illustration af, hvilken klasse den optimale rute ligger indenfor forskellige steder i konfigurationsrummet.

Klasserne CSC og CCC dækker kun over otte af de i formel 2.2 nævnte mere specifikke atten klasser, nemlig over $C^+C^-C^+$, $C^+C^-C^-$, $C^+S^+C^+$, $C^+C^+C^-$ og de samme fire klasser med modsat fortegn. Formler for de resterende ti klasser er ikke udledt, selvom de står postuleret i [Reeds and Shepp, 1990]. Som nævnt indeholder i hvert fald to ud af de tre første af disse formler fejl, og da det i [Rode and Kjeldsen, 2004] viste sig at være temmelig tidskrævende at udlede og dermed rette disse formler, er udledningen og anvendelsen af de resterende formler ikke foretaget her. Klasserne CSC, CCC og SCS er dog tilstrækkelige til at give et tilfredsstillende resultat i den videre ruteplanlægning.

2.2 Robottens kort over omgivelserne

For at robotten skal kende sine omgivelser og specielt potentielle forhindringer, får den et kort. Kortet skal den bruge under ruteplanlægningen, så ruten kan planlægges på en sådan måde, at hvis robotten følger ruten, så støder den ikke ind i forhindringerne. Desuden skal kortet indeholde information om, hvor visse landmærker er placeret.

Kortet indeholder i alt tre former for information. For det første defineres kortets udstrækning. Da robotten kun bevæger sig i planet, er kortet todimensionelt, og dets udstrækning defineres som et rektangel. Hvad der findes uden for dette rek-



Figur 2.9: Illustration af metrikken på konfigurationsrummet. De fem første grafer viser, hvilken klasse den optimale rute ligger inden for, med startpositionen $p_1 = (0,0,0)$ fast og med varierende slutposition $p_2 = (x,y,\theta)$ for $\theta = \{0,\ 45,\ 90,\ 135,\ 180\}$ grader. Rød svarer til klassen CCC, grøn svarer til lsr og rsl og blå svarer til lsl og rsr. Grafen nederst til højre viser afstanden mellem $p_1 = (0,0,0)$ og $p_2 = (x,y,0)$ som en intensitet, hvor mørkere intensitet svarer til længere afstande.

tangel er udefineret. Den verden $\mathcal{W}$ robotten kan bevæge sig i begrænses altså til et todimensionelt rektangel.

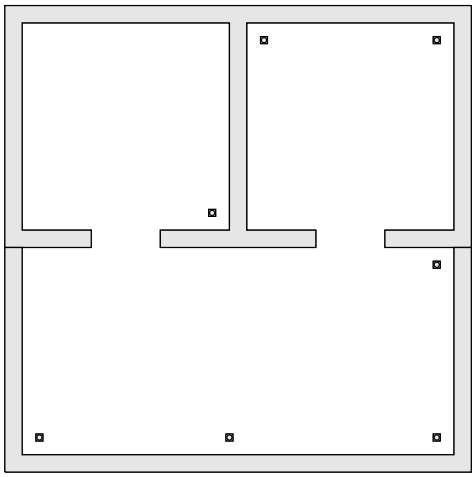
For det andet defineres et antal forhindringer. Disse defineres som todimensionelle polygoner med et endeligt antal kanter. Valget af polygoner som repræsentation af forhindringerne grunder i, at disse er meget simple at repræsentere og at de samtidigt er fleksible nok til at give en rimelig repræsentation af robottens omgivelser. Desuden er det simpelt at konstatere om to polygoner overlapper hinanden. Det sidste bliver nyttigt, når det senere i dette kapitel skal konstateres, om en rute er kollisionsfri.

For det tredje defineres placeringen af en række landmærker. Hvert landmærke har et unikt nummer, og en placering på kortet. Informationen om landmærkerne vil i kapitel 3 blive anvendt til positionsestimering. Se afsnit 3.2 for en beskrivelse af landmærkerne. Hvert landmærke opfattes samtidigt også som en forhindring, defineret ved et 10×10 cm kvadrat centreret om landmærkets koordinater.

Forhindringerne incl. landmærkerne udgører altså en delmængde $\mathcal{O} \subseteq \mathcal{W}$ af robottens verden.

For en visualisering af et kort, se figur 2.10, som forestiller tre kontorer med døre imellem incl. placering af en række landmærker.

Et kort repræsenteres ved en tekstfil, som indlæses, når kortet skal bruges.



Figur 2.10: Visualisering af et kort over tre kontorer. Kortet består af to polygoner, som udgør væggene, og af syv landmærker.

2.3 Kollisionsbestemmelse

I afsnit 2.1 blev det beskrevet, hvorledes en rute mellem to positioner kan findes for en bil-agtig robot, uden hensyntagen til forhindringer. Det ønskes nu i dette afsnit at bestemme om en sådan given rute vil føre til en kollision med de på et givet kort beskrevne forhindringer, hvis en bil-agtig robot følger ruten.

I afsnit 2.3.1 herunder vil det areal robotten passerer over, når den følger en given rute, blive bestemt. Dette areal vil blive udtrykt som en polygon. Hernæst beskriver afsnit 2.3.2, hvordan det konstateres, om to polygoner overlapper hinanden. Dette bruges i afsnit 2.3.3. Her beskrives det, hvordan det på en effektiv måde kan konstateres, om arealet en bil-agtig robot passerer over, når den følger en given rute, overlapper med mængden af forhindringer beskrevet på et givet kort.

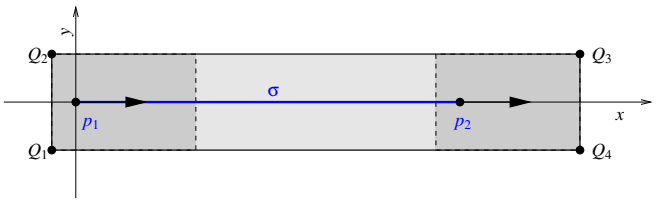
2.3.1 Arealet en bil-agtig robot passerer over, når den følger en given rute

I dette afsnit ønskes det at finde det areal en bil-agtig robot passerer over, når den følger en given rute γ. Ud fra dette areal kan det nemlig afgøres, om robotten vil kollidere med en eller flere forhindringer, når γ følges. Ruten γ består af en række segmenter σ_i. Hvert segment er enten et linjestykke eller en cirkelbue, jf. afsnit 2.1.

Formelt defineres arealet en bil-agtig robot passerer over, når den følger et segment σ_i, som foreningsmængden af de arealer robottens omgivende rektangel udgør, når origo i robottens lokale koordinatsystem R placeres på ethvert punkt P_j på σ_i, med robottens midterakse som tangent til σ_i i P_j. Arealet som robotten passerer over, når den følger en given rute γ er så naturligvis foreningsmængden af arealerne robotten passerer over, når hvert segment følges.

For at gøre kollisionsbestemmelsen så enkel som mulig, ønskes arealet udtrykt som polygoner, da forhindringerne på et kort også er udtrykt som polygoner. At konstatere, om en given rute vil føre til kollision, kan så afgøres blot ved at konstatere, om det passerede areals polygoner overlapper med en af forhindringspolygonerne, dvs. alene ved konstatering af overlap mellem polygoner.

Da hvert segment af ruten enten er et linjestykke eller en cirkelbue, betragtes disse to deltilfælde herunder.



Figur 2.11: Arealet, som passeres over, når linjestykket σ følges, markeret med gråt. p_1 er startpostionen, p_2 er slutposition og Q_i er hjørnepunkter i rektanglet.

2.3.1.1 Linjestykke

At finde arealet, som passeres over når et linjestykke følges, er ganske simpelt, og resultatet er naturligvis en polygon, nemlig et rektangel.

De fire hjørnepunkter $Q_1 - Q_4$ i polygonet findes først i robottens lokale koordinatsystem, se figur 2.11, figur 2.1 og appendiks A:

$$Q_1 = (-x_{bb},\ -y_{bb}) \tag{2.31}$$
$$Q_2 = (-x_{bb},\ W_{bb} - y_{bb}) \tag{2.32}$$
$$Q_3 = (l + L_{bb} - x_{bb},\ W_{bb} - y_{bb}) \tag{2.33}$$
$$Q_4 = (l + L_{bb} - x_{bb},\ -y_{bb}) \tag{2.34}$$

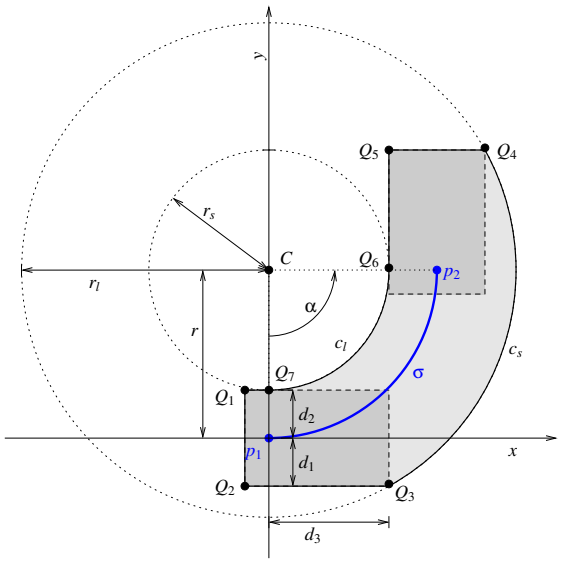
Her angiver l længden af σ, og x_{bb}, y_{bb}, W_{bb} og L_{bb} angiver størrelsen og positionen af det omgivende rektangel for robotten, jf. figur 2.1. Dette omgivende rektangel kan vælges som det mindste omgivende rektangel for robotten, hvis man vil udregne det præcise areal, robotten passerer over (hvis robotten er rektangulær). Alternativt kan man vælge det noget større, så det kollisionsfri areal er noget større end det rent faktisk passerede areal. Dette giver noget luft, så upræcisioner i robottens positionsestimat ikke med det samme fører til kollision.

For at flytte origo til det globale koordinatsystem, roteres samtlige punkter θ_1 radianer, hvor θ_1 er orienteringen af positionen p_1, hvorefter der adderes med vektoren (x_1, y_1), hvor x_1 og y_1 er koordinaterne af positionen p_1 i det globale koordinatsystem:

$$Q_i = \mathrm{Rot}(Q_i,\ \theta_1) + (x_1,\ y_1) \tag{2.35}$$

Her betegner $\mathrm{Rot}(P,\ \theta)$ rotationen af punktet P θ radianer om origo.

2.3.1.2 Cirkelbue



Figur 2.12: Arealet, som passeres over, når cirkelbuen σ følges, markeret med gråt. p_1 er startpostionen, p_2 er slutposition og Q_i er hjørnepunkterne i det passerede areal.

Samtlige punkter i det tilnærmede arealpolygon udtrykkes først i robottens lokale koordinatsystem, se figur 2.12 og appendiks A.3, da det er mest naturligt.

Afstandene d_1, d_2 og d_3 kan udregnes ud fra dimensionerne af robottens omgivende rektangel, samt ud fra placeringen af det omgivende rektangel i forhold til origo i robottens lokale koordinatsystem. Origo er sammenfaldende med punktet R, se figur 2.1:

$$d_1 = y_{bb} \tag{2.36}$$
$$d_2 = W_{bb} - y_{bb} \tag{2.37}$$
$$d_3 = L_{bb} - x_{bb} \tag{2.38}$$

Da punkternes koordinater afhænger af omløbsretningen, defineres y-hjælpekoordinaterne y_a og y_b som:

$$y_a = \begin{cases} -d_1 & \text{hvis } \sigma \text{ drejer mod venstre} \\ d_2 & \text{hvis } \sigma \text{ drejer mod højre} \end{cases} \tag{2.39}$$

$$y_b = \begin{cases} d_2 & \text{hvis } \sigma \text{ drejer mod venstre} \\ -d_1 & \text{hvis } \sigma \text{ drejer mod højre} \end{cases} \tag{2.40}$$

At finde koordinaterne på punkterne Q_1, Q_2, Q_3 og Q_7, i robottens lokale koordinatsystem er nu trivielt, jf. figur 2.12:

$$Q_1 = (-x_{bb},\ y_b) \tag{2.41}$$
$$Q_2 = (-x_{bb},\ y_a) \tag{2.42}$$
$$Q_3 = (d_3,\ y_a) \tag{2.43}$$
$$Q_7 = (0,\ y_b) \tag{2.44}$$

Koordinaterne for punkterne Q_4, Q_5 og Q_6 findes ved en rotation af punkterne Q_3, (d_3, y_b) og Q_7 på α radianer, hvor α kan findes ud fra cirkelbuens radius r og længde l som:

$$\alpha = \frac{l}{r} \tag{2.45}$$

Da der findes to forskellige omløbsretninger for en cirkelbue, defineres først hjælpevariabel s som:

$$s = \begin{cases} 1 & \text{hvis } \sigma \text{ drejer mod venstre} \\ -1 & \text{hvis } \sigma \text{ drejer mod højre} \end{cases} \tag{2.46}$$

Punkterne Q_4 – Q_6 kan nu findes som:

$$Q_4 = \mathrm{Rot}(Q_3,\ s\alpha,\ C) \tag{2.47}$$
$$Q_5 = \mathrm{Rot}((d_3,\ y_b),\ s\alpha,\ C) \tag{2.48}$$
$$Q_6 = \mathrm{Rot}(Q_7,\ s\alpha,\ C) \tag{2.49}$$

Her betegner funktionen $\mathrm{Rot}(P, \theta, C)$ rotationen af P θ radianer omkring punktet C.

Da arealet som før nævnt ønskes udtrykt som en polygon, må de to cirkelbuer c_l fra Q_3 til Q_4 og c_s fra Q_6 til Q_7 opløses i en række linjestykker. Endepunkterne af disse findes i det følgende, se også figur 2.13.

Først findes radius r_s af c_s ud fra Pythagoras. Da r_s afhænger af omløbsretningen af segmentets cirkelbue, defineres:

$$a_s = \begin{cases} r + d_1 & \text{hvis } \sigma \text{ drejer mod venstre} \\ r + d_2 & \text{hvis } \sigma \text{ drejer mod højre} \end{cases} \tag{2.50}$$

Nu kan radius r_s af c_s, som i øvrigt svarer til afstanden $\| C\ Q_3 \|$, findes som:

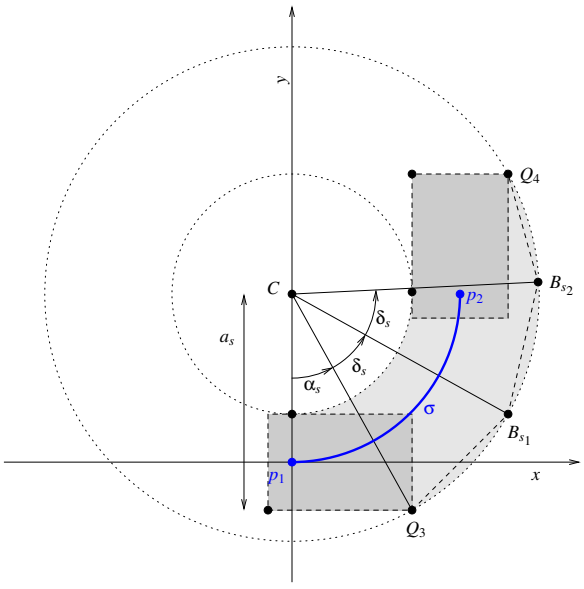
$$r_s = \| C\ Q_3 \| = \sqrt{a_s^2 + d_3^2} \tag{2.51}$$

Radius r_l af cirkelbuen c_s kan findes findes som:

$$r_l = \begin{cases} r - d_2 & \text{hvis } \sigma \text{ drejer mod venstre} \\ r - d_1 & \text{hvis } \sigma \text{ drejer mod højre} \end{cases} \tag{2.52}$$

Det antages her, at $r_l \geq 0$, hvilket den vil være, hvis det omgivne rektangel ikke er meget stort i forhold til drejeradiussen.

Da resten af udledningen af endepunkterne i opløsningen af de to cirkelbuer er tilsvarende for både c_s og c_l, vises herunder kun udledningen for c_s. Se figur 2.13 for en illustration af diverse størrelser nævnt i det nedenstående.



Figur 2.13: Illustration af vinklerne α_s og δ_s samt punkterne B_{s_i}.

Idet l som før nævnt betegner længden af segmentets cirkelbue, findes længden l_s af cirkelbuen c_s som:

$$l_s = l\frac{r_s}{r} \tag{2.53}$$

Cirkelbuen ønskes opløst i et antal punkter k_s ud fra et fast antal punkter pr. meter, K, hvilket giver:

$$k_s = \lfloor l_s K \rfloor \tag{2.54}$$

c_s skal så opløses i k_s punkter, hvilket giver $k_s + 1$ buestykker hver med buevinklen δ_s:

$$\delta_s = \frac{\alpha}{k_s + 1} \tag{2.55}$$

Punkterne B_{s_i} på c_s findes nu ved rotation og translation af en vektor v. Startvinklen α_s af v vælges som vinklen af vektoren fra c til Q_3:

$$\alpha_s = \arctan\left(\frac{d_3}{a_s}\right) \tag{2.56}$$

Nu kan v findes som $v = \mathrm{rot}((0,\ -sr_b),\ s\alpha_s)$, hvor $\mathrm{rot}(P, \alpha)$ roterer et punkt P α radianer omkring origo. Man kan tænke på v som en vektor fra c til Q_3, men v har origo i $(0,0)$.

Nu kan det i'te punkt B_{s_i} på c_s findes som:

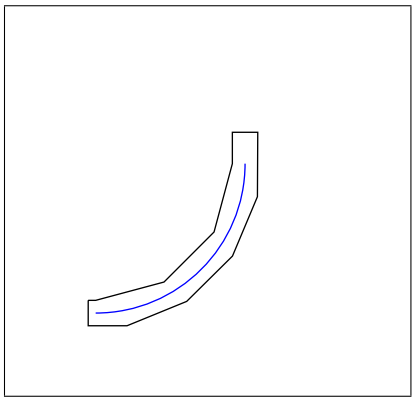
$$B_{s_i} = \mathrm{rot}(v,\ is\delta_s) + (0,\ sr) \tag{2.57}$$

Når vektoren $(0,\ sr)$ adderes i det ovenstående, flyttes origo af v til robottens lokale koordinatsystem.

Nu kendes punkterne Q_1 – Q_8 og B_{s_i}. Punkterne B_{l_i} som er opløsningen af c_l kan findes helt analogt til punkterne B_{s_i}. Nu haves altså samtlige ønskede punkter, men dog med origo i robottens lokale koordinatsystem. Origo flyttes til det globale koordinatsystem ved brug af formel 2.35.

I alt det ovenstående blev det antaget, at cirkelbuens længde er positiv. Det er den ikke, hvis robotten skal bakke langs cirkelbuen. Dette tilfælde klares dog let ved at observere, at robotten naturligvis kører over samme areal, hvis den kører forlæns fra p_1 til p_2, som hvis den bakker fra p_2 til p_1. Hvis buelængden derfor er negativ, sættes startpositionen blot til p_2, og der skiftes fortegn på buelængden, og så giver ovenstående udledning det rigtige resultat.

Figur 2.14 viser den beregnede tilnærmede polygon for en cirkelbue.



Figur 2.14: Den beregnede tilnærmede polygon (sort) for det af robotten passerede areal for cirkelbuen (blå) med længde $\frac{\pi}{2}$ m og radius 1,0 m, positiv omløbsretning, skala 1:50, $K = 2$.

2.3.2 Konstatering af overlap mellem to polygoner

I dette afsnit vil det blive belyst, hvordan det konstateres, om to polygoner overlapper hinanden. Hvor eller hvor meget de overlapper hinanden er ikke interessant i denne sammenhæng. Først defineres en polygon. En polygon $\mathcal{P}^n$ består af n sammenhængende linjestykker, som ikke tillades at skære hinanden. Polygonen repræsenteres ved de n linjestykkers startpunkter, se appendiks B for en helt præcis definition af polygoner og linjestykker.

At konstatere, om to polygoner $\mathcal{P}_1$ og $\mathcal{P}_2$ overlapper, dvs. at deres fællesmængde ikke er tom, kan gøres ved at teste tre tilfælde:

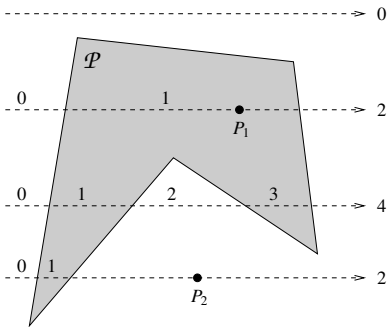
- $\mathcal{P}_1$ er helt indeholdt i $\mathcal{P}_2$.
- $\mathcal{P}_2$ er helt indeholdt i $\mathcal{P}_1$.
- En kant af $\mathcal{P}_1$ skærer en kant af $\mathcal{P}_2$.

Er mindst et af disse tre tilfælde opfyldte, overlapper de to polygoner.

2.3.2.1 En polygon indeholdt i en anden polygon

At konstatere, om en polygon $\mathcal{P}_1$ er helt indeholdt i en anden polygon $\mathcal{P}_2$, kan gøres ved at undersøge, om blot et enkelt punkt i $\mathcal{P}_1$, fx et hjørnepunkt fra $\mathcal{P}_1$, er indeholdt i $\mathcal{P}_2$. Om et punkt P er indeholdt i en polygon $\mathcal{P}$ kan konstateres ved en simpel skanlinjealgoritme, jf. [Bourke, 1987, løsning 1]. Algoritmen virker ved

at følge en horisontal linje gennem punktets y-koordinat, og så tælle antallet t af kanter fra $\mathcal{P}$ som passeres, når linjen følges fra $x = -\infty$ til P's x-koordinat, se figur 2.15. Hvis $t \equiv 1 (\mathrm{mod}\ 2)$, så ligger P inde i $\mathcal{P}$, ellers ligger P ikke i $\mathcal{P}$.



Figur 2.15: Konstatering om et punkt ligger inde i en polygon. Når en stiblet linje følges, tælles antallet t af gange en kant fra polygonen passeres. Gælder $t \equiv 1 \pmod 2$, så ligger punktet inde i polygonen.
For punktet P_1 gælder $t_1 \equiv 1 \pmod 2$, så P_1 ligger inde i $\mathcal{P}$. For punktet P_2 gælder $t_2 \equiv 0 \pmod 2$, så P_2 ligger ikke inde i $\mathcal{P}$.
Figuren er tegnet frit efter [Bourke, 1987].

2.3.2.2 En kant af en polygon skærer en kant af en anden polygon

At konstatere, om en kant fra en polygon $\mathcal{P}_1^n$ krydser en kant fra en polygon $\mathcal{P}_2^m$ kan gøres meget simpelt ved for hver af de n kanter i $\mathcal{P}_1^n$ at undersøge om den krydser en af de m kanter i $\mathcal{P}_2^m$.
Denne algoritme har naturligvis køretiden $O(nm)$. Der eksisterer algoritmer, de såkaldte *plane sweep* algoritmer, som kan reducere køretiden til (den bevisligt optimale) $O((n+m)\log(n+m))$, jf. [Chan, 1994]. Plane sweep algoritmen blev første gang foreslået af [Bentley and Ottmann, 1979].
Da disse er forholdsvis komplicerede at implementere, nøjes jeg i dette speciale med den simple algoritme med køretid $O(nm)$. I det næste afsnit beskrives, hvorledes antallet af sammenligninger minimeres, så valget af den simple algoritme ikke får den store betydning.

For at konstatere, om to linjesegmenter skærer hinanden, anvendes algoritmen beskrevet i [Sunday, 2001], som også giver deres eventuelle skæringspunkt. Kort beskrevet virker algoritmen ved først at finde et eventuelt skæringspunkt mellem de to linjer, som linjesegmenterne udspænder. Herefter konstateres det, om skæringspunktet ligger på begge linjesegmenter.

2.3.3 Konstatering af overlap mellem en polygon og en mængde af polygoner

Skal man afgøre, om en polygon $\mathcal{P}$ overlapper med en eller flere polygoner i en mængde af polygoner M, kan man naturligvis blot for hver polygon $\mathcal{P}_j \in M$ konstatere om $\mathcal{P}$ og $\mathcal{P}_j$ overlapper. Dette giver en lineær køretid $O(nl)$, hvor n er antallet af polygoner i M, og l er et udtryk for tiden det tager at sammenligne to polygoner kant for kant. Dette kan forbedres ved at kigge på det akseparallelle omgivende rektangel (AABB[1]) for polygonerne. Køretiden kan så reduceres til $O(n+kl)$, hvor k er antallet af polygoner i M, hvis AABB overlapper $\mathcal{P}$ og hvor $k \ll n$ i det generelle tilfælde. En yderligere forbedring kan opnås ved brug af en *sweep and prune* algoritme, så køretiden reduceres til $O(\log n + k_1 + k_2 l)$, hvor $k_2 \ll k_1 \ll n$ i det generelle tilfælde. Dette vil blive gennemgået i det følgende.

Sweep and prune algoritmen reducerer antallet af overlaptest ved at eliminiere test mod polygoner fra M som ligger langt fra $\mathcal{P}$, jf. [Cohen et al., 1995, p. 4–5].

Algoritmen består overordnet af tre trin:

1. Udregning af AABB for $\mathcal{P}$ og for hver $\mathcal{P}_j \in M$.
2. Sortering af polygonerne i M efter koordinaterne på deres AABB.
3. Afgøre, hvilke af polygonerne i M som har en AABB som overlapper $\mathcal{P}$'s AABB.

Når man har fundet de polygoner fra M, hvis AABB overlapper $\mathcal{P}$'s AABB, skal man naturligvis sidst afgøre, om selve polygonerne overlapper $\mathcal{P}$.

De to sidste af de ovenstående tre trin bliver beskrevet nærmere i det følgende. Da udregning af AABB for en polygon er triviel, vil det ikke blive nærmere beskrevet.

2.3.3.1 Sortering efter AABB

Først gøres en betragtning over betydningen af overlap af AABB for to polygoner. Sætningerne vil ikke blive bevist, da beviserne er oplagte.

Sætning 1 *Hvis to polygoner overlapper, så overlapper deres AABB.*

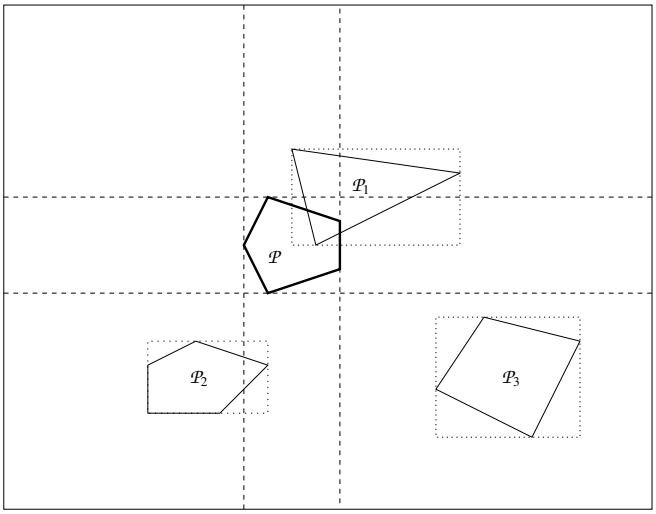
Projekterer man deres AABB ind på hhv. x-aksen og y-aksen giver dette anledning til to intervaller på både x-aksen og y-aksen.

[1] AABB er en forkortelse for den engelske betegnelse Axis Aligned Bounding Box.

Sætning 2 *To polygoners AABB overlapper, hvis og kun hvis deres projekterede intervaller overlapper på begge akser.*

Dvs. findes en akse, hvor intervallerne ikke overlapper, så overlapper de tilhørende AABB ikke. Sætning 2 svarer til [Zomorodian and Edelsbrunner, 2002, p. 4, property 1].

Ud fra sætning 1 og 2 kan det altså konkluderes, at man kan kun behøver at teste to polygoner for overlap, hvis de projekterede intervaller for polygonernes AABB overlapper på begge akser, se figur 2.16.



Figur 2.16: Illustration af overlap mellem en polygon $\mathcal{P}$ og en mængde af polygoner $\mathcal{P}_i$. Figuren illustrerer tre deltilfælde, nemlig $\mathcal{P}$ overlapper $\mathcal{P}_1$, $\mathcal{P}$ overlapper ikke $\mathcal{P}_2$, men de akseparallelle omgivne rektangler for $\mathcal{P}$ og $\mathcal{P}_2$ overlapper. $\mathcal{P}$ og $\mathcal{P}_3$ overlapper på ingen måde hinanden.

Problemet med at reducere antallet af overlapsammenligninger mellem polygoners AABB kan altså reduceres til en-dimensionelle intervalsammenligninger.

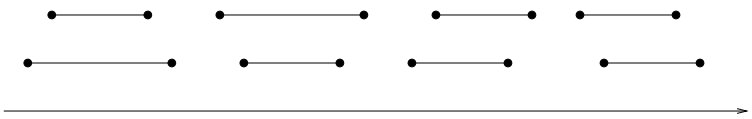
Den her beskrevne algoritme til konstatering om en mængde af AABB'er overlapper en givet AABB følger den i [Zomorodian and Edelsbrunner, 2002] beskrevne fremgangsmåde, men mere sofistikerede teknikker findes, fx jf. [de Berg et al., 2000, afsnit 5 og 10]. Valget har været en afvejning: Mere sofistikerede algoritmer er mere komplicerede og tager dermed længere tid at implementere, men kan potentielt mindske algoritmens køretid yderligere.

2.3.3.2 Intervaloverlap

Når det skal afgøres, om to intervaller overlapper, dvs. har en ikke-tom fællesmængde, får man brug for følgende egenskab:

Sætning 3 *To intervaller overlapper, dvs. har en ikke-tom fællesmængde, hvis og kun hvis et af intervallerne indeholder startpunktet af det andet interval.*

I forbindelse med intervaller menes der med et punkt, her og i det følgende, et endimensionalt punkt, altså et reelt tal eller et heltal. Sætning 3 er illustreret på figur 2.17. Den svarer til property 2 i [Zomorodian and Edelsbrunner, 2002, p. 4].



Figur 2.17: De fire måder to lukkede intervaller kan overlappe på. Frit efter figur 2 i [Zomorodian and Edelsbrunner, 2002, p. 4]

Lad I_x være de projekterede intervaller af M's AABB på x-aksen. At netop x-aksen er valgt er vilkårligt, det kunne lige så godt have været y-aksen. Det ønskes nu at finde de intervaller $O_x \subseteq I_x$ som overlapper med det på x-aksen projekterede interval i_x af polygonen $\mathcal{P}$'s AABB.

For at finde O_x, må vi altså ifølge sætning 3 finde foreningsmængden af de intervaller af I_x som indeholder startpunktet af i_x, og de intervaller af I_x, hvis startpunkter er indeholdt i i_x.

To datastrukturer vil i det følgende blive introduceret. Den første, nemlig *range*-træet, er velegnet til at finde de punkter af en mængde af punkter, som er indeholdt i et givet interval. Det andet, nemlig *segment*-træet, er velegent til at finde de intervaller af en mængde af intervaller, som indeholder et givet punkt.

2.3.3.3 Range-træ

Et *range*-træ, jf. [Zomorodian and Edelsbrunner, 2002, p. 6], er som før nævnt en velegnet datastruktur til at lagre (en-dimensionale) punkter i, når der ønskes et effektivt svar på, hvilke af punkterne, der er indeholdt i et givet interval. Range-træet tager $O(n \log n)$ tid at opbygge, hvor n er antallet af punkter. Et svar på, hvilke punkter der er indeholdt i et givet interval tager $O(\log n + k)$ tid, hvor k er antallet af punkter i svaret, jf. [de Berg et al., 2000, afsnit 5.1].

Et range-træ er et balanceret binært søgetræ, nærværende speciale implementerer det som et AVL-træ, jf. [Decker, 1989, afsnit 7.1]. Hver knude i træet indeholder et unikt punkt (et flydende kommatal eller et heltal).

Når der foretages et opslag i træet, bruges følgende algoritme:

`rtree_query(node, start, end)`
Inddata: Roden af range-træet, intervallets startpunkt, intervallets slutpunkt.
Uddata: Alle de punkter, som er indeholdt i intervallet.

1. **if** node.val>=start **and** node.val<=end **then** report(node)
2. **if** exists(node.left) **and** start>node.val **then** `rtree_query(node.left, start, end)`
3. **if** exists(node.right) **and** node.val<end **then** `rtree_query(node.right, start, end)`

Når range-træet bruges i sammenhæng med test af overlap mellem polygoners AABB, indsættes startpunkterne af alle intervallerne i I_x. Hver knude indeholder nu et unikt startpunkt, samt en liste af de intervaller af I_x, som har dette punkt som startpunkt. I et binært søgetræ er det et krav, at hver knude indeholder et unikt punkt.

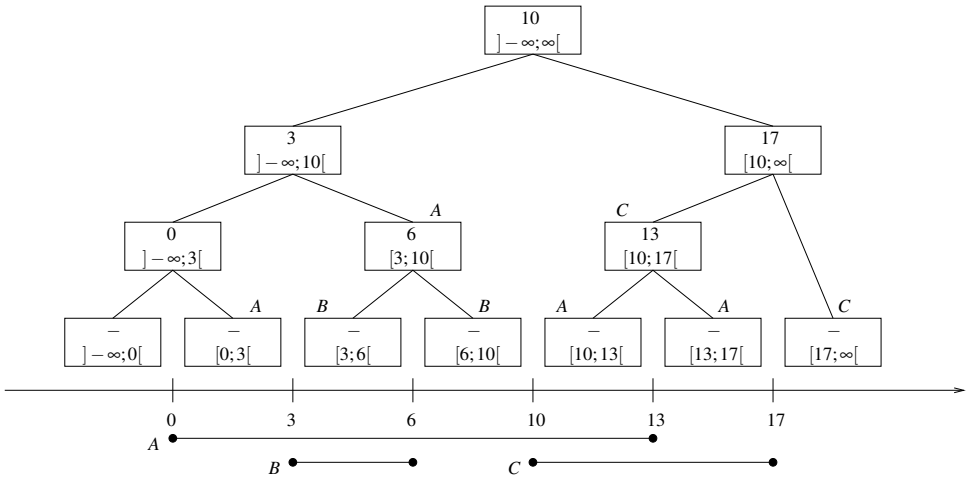
2.3.3.4 Segmenttræ

Et *segment*-træ er en velegnet datastruktur til at lagre intervaller i, når der ønskes et effektivt svar på, hvilke af disse intervaller, som indeholder et givet punkt, jf. [de Berg et al., 2000, afsnit 10.3]. Et segmenttræ tager $O(n \log n)$ tid at opbygge, hvor n er antallet af intervaller. Et svar på, hvilke intervaller, der er indeholder i et givet punkt tager $O(\log n + k)$ tid, hvor k er antallet af intervaller i svaret.

Den grundlæggende struktur i segmenttræet er et balanceret binært søgetræ. Hver knude indeholder et start- og et slutpunkt, som angiver det interval knuden repræsenterer, samt en "opdelings-værdi", som angiver, at det venstre barn repræsenterer et interval med slutpunkt i denne opdelingsværdi, og at højre barn repræsenterer et interval med startpunkt i denne opdelingsværdi. Opdelingsværdien er derfor redundant, men sparer en smule køretid.

Træets struktur er bedst illustreret ved en figur, se figur 2.18.

Som det ses repræsenterer hver forældreknude foreningsmængden af de intervaller knudens to børn repræsenterer. Træet opdeler på denne måde den reelle akse i



Figur 2.18: Illustration af segmenttræ. Træet indeholder tre intervaller $A = [0;\ 13]$, $B = [3;\ 6]$ og $C = [10;\ 17]$. Hver knude indeholder start- og slutpunkt i det interval knuden og dens undertræ repræsenterer, samt en opdelingsværdi, som angiver opdelingen af højre og venstre undertræs intervaller. Ved siden af knuderne er markeret, hvilke af intervallerne A, B og C knuden er et delinterval af.

et antal ikkeoverlappende halvåbne intervaller. Hver knude er desuden tilknyttet en liste af de intervaller, som det interval knuden repræsenterer er en delmængde af. Dette er illustreret med intervalnavnene A, B og C ved siden af knuderne på figur 2.18. Hvis en knudes interval repræsenterer en delmængde af et givent interval, er de intervaller dens børn repræsenterer naturligvis også en delmængde af det givne interval, og dette tilknyttes derfor ikke til børneknuderne, da denne redundans blot ville fylde op i hukommelsen og ikke give nogen køretidsforbedring.

Et segmenttræ opbygges ud fra en mængde I af intervaller i tre trin:

1. Intervallernes start- og slutpunkter indsættes i træet, som var det et almindeligt binært søgetræ. Disse svarer til de ovennævnte opdelingsværdier, og er det øverste tal i hver knude på figur 2.18.

2. Træet gennemløbes, og for hver knude afgøres, hvilket delinterval af den reelle akse knuden repræsenterer. På figur 2.18 ses det interval hver knude repræsenterer nederst i knuden.

3. Sidst afgøres det for hvert interval $i \in I$, hvilke knuder i træet, som repræsenterer et delinterval af i, og disse knuder får tilføjet i til deres intervalliste. Dette svarer til intervalnavnene A, B og C ved siden af knuderne på figur

2.18. Da hvert interval $i \in I$ er lukket, tilføjes i også til en knudes intervalliste, hvis slutpunktet af i er det samme som startpunktet for det interval knuden repræsenterer.

For at foretage et opslag i træet, kan følgende algoritme anvendes:

`stree_query(node, point)`
Inddata: Roden af segment-træet, et punkt.
Uddata: Alle de intervaller som indeholder punktet.

1. **if** node.start<=point **and** node.end>=point **then** report(node)
2. **if** exists(node.left) **then** `stree_query(node.left, point)`
3. **if** exists(node.right) **then** `stree_query(node.right, point)`

2.3.3.5 Den samlede sweep and prune algoritme

Her opsummeres den samlede algoritme til effektiv konstatering af overlap mellem et givet polygon $\mathcal{P}$ og en givet fast mængde M af polygoner. At M er fast er vigtigt, da det tager $O(n \log n)$ tid at opbygge hhv. range- og segmenttræet for M, så hvis det ikke genbruges, bliver køretiden $O(n \log n)$ og ikke de lovede $O(\log n + k)$.

Først opbygges datastrukturerne:

- For hver polygon $\mathcal{P}_j \in M$ udregnes polygonens AABB.
- Lad I_x være projektionen af mængden M's AABB ind på x-aksen.
- Indsæt I_x i et range-træ og i et segmenttræ.

Når der skal bestemmes, om en polygon $\mathcal{P}$ overlapper med en eller flere af polygonerne i M, udføres følgende tre ting:

1. Find de polygoner $\mathcal{P}_x \subseteq M$, hvis AABB overlapper med $\mathcal{P}$'s AABB på x-aksen, ved søgning i range-træet og segmenttræet.
2. Find de polygoner $\mathcal{P}_y \subseteq \mathcal{P}_x$, hvis AABB overlapper med $\mathcal{P}$'s AABB på y-aksen.
3. For hver polygon i $\mathcal{P}_y$ undersøges, om det overlapper med $\mathcal{P}$ ud fra algoritmen beskrevet i afsnit 2.3.2.

I indledningen til dette afsnit blev en køretid på $O(\log n + k_1 + k_2 l)$ nævnt. Det vil blive begrundet i det følgende.

Tiden for opbygningen af datastrukturerne medregnes ikke, da de kan genbruges til efterfølgende søgninger. Trin 1 i det ovenstående tager $O(\log n + k_1)$, jf. afsnit 2.3.3.3 og afsnit 2.3.3.4, hvor k_1 er antallet af polygonerne i $\mathcal{P}_x$. Trin 2 tager k_1 polygoner som inddata og giver k_2 polygoner som uddata, $k_2 \leq k_1$, dvs. en køretid på $O(k_1)$. Trin 3 undersøger om de k_2 polygoner overlapper med $\mathcal{P}$, hvilket tager $O(k_2 l)$, hvor l er et mål for, hvor lang tid testen for polygonoverlap tager amortiseret. Dette giver i alt en køretid på $O(\log n + k_1 + k_1 + k_2 l) = O(\log n + k_1 + k_2 l)$ som lovet.

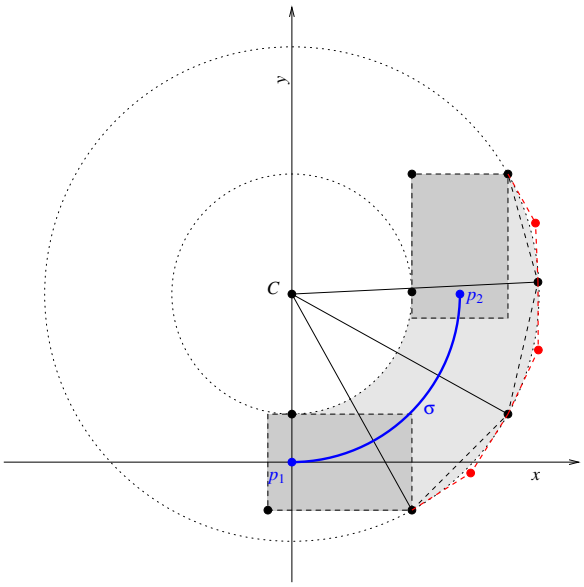
2.3.4 Delkonklusion og mulige forbedringer

Dette afsnit fandt først det areal en bil-agtig robot passerer over, når den følger en given rute. Arealet blev udtrykt som en polygon, for at lette den følgende kollisionsbestemmelse. Herefter blev det beskrevet, hvorledes det konstateres, om to polygoner overlapper. Til sidst blev en algoritme beskrevet, som på en effektiv måde konstaterer, om en polygon overlapper med en mængde af polygoner. I alt gav afsnittet en løsning på, hvordan det effektivt kan bestemmes, om en given rute vil føre til kollision med en eller flere forhindringer, hvis den følges af en bil-agtig robot.

Afsnittet lægger op til en del forbedringsmuligheder. Det passerede areal for en cirkelbue blev tilnærmet med en polygon, som ikke indeholder hele det faktisk passerede areal. Det kan føre til ubemærkede kollisioner, hvis robottens omgivne rektangel ikke gøres passsende større. Det fundne polygon burde som minimum indeholde hele det faktisk passerede areal, se figur 2.19.

Idet det passerede areal ved cirkelbuer kun bliver tilnærmet med en polygon, kan dette naturligvis også forbedres, så det faktisk passerede areal anvendes. En algoritme til bestemmelse af overlap mellem en polygon og en geometrisk form bestående af linjestykker og cirkelbuer ville så skulle findes. Det er ikke utænkeligt, at dette kunne føre til både øget præcision og mindsket køretid.

Hvis den samlede køretid for hele kollisionsbestemmelsen ønskes forbedret, burde det nærmere undersøges, præcis i hvilke dele af algoritmen, der bruges mest køretid. Køretiden for kollisionsbestemmelsen mellem to polygoner kan som nævnt reduceres ved plane sweep algoritmen. Køretiden for bestemmelse af overlap mellem en polygons AABB og en mængde af polygoners AABB'er kan også reduceres, fx ved brug af kd-træer, jf. [de Berg et al., 2000].



Figur 2.19: Illustration af en bedre tilnærmelse af det passerede areal for en cirkelbue, markeret med rødt.

2.4 Planlægningsalgoritmer

Det i afsnit 2.1, 2.2 og 2.3 gennemgåede stof har et helt specifikt formål, nemlig til brug for den valgte ruteplanlægningsalgoritme, men før den gennemgås, vil dette afsnit give en kort introduktion til et udvalg af forskellige metoder til bevægelsesplanlægning.

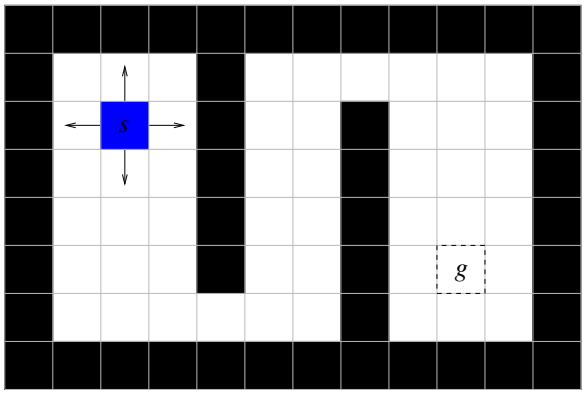
Først defineres forhindringsområdet i konfigurationsrummet til brug i det følgende, jf. [LaValle, 2004]. Givet en verden $\mathcal{W}$, et konfigurationsrum $\mathcal{C}$ og et forhindringsområde $O \subset \mathcal{W}$, defineres forhindringsområdet $\mathcal{C}_{obs} \subseteq \mathcal{C}$ i konfigurationsrummet for en robot $\mathcal{A}$. Robotten skal have en fast udstrækning $\mathcal{A} \subset \mathcal{W}$. Udstrækningen kan fx være defineret som det omgivende rektangel for en bil-agtig robot, jf. figur 2.1. Lad $x \in \mathcal{C}$ definere konfigurationen af $\mathcal{A}$. Forhindringsområdet i konfigurationsrummet $\mathcal{C}_{obs} \subseteq \mathcal{C}$ defineres nu som:

$$\mathcal{C}_{obs} = \{x \in \mathcal{C} \mid \mathcal{A}(x) \cap O \neq \emptyset\} \tag{2.58}$$

Resten af konfigurationsrummet $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$ er det frie rum, hvor robotten kan befinde sig uden at kollidere med forhindringerne.

2.4.1 Diskrete planlægningsalgoritmer

Hvis konfigurationsrummet for en robot har et endeligt eller bare tælleligt antal konfigurationer, kan man betragte konfigurationsrummet som en graf, hvor hver knude er en konfiguration og med kanter svarende til de mulige bevægelser fra en konfiguration til en anden, eventuelt vægtet med bevægelsens omkostninger eller længde. På figur 2.20 ses et eksempel på en robot med et endeligt konfigurationsrum.

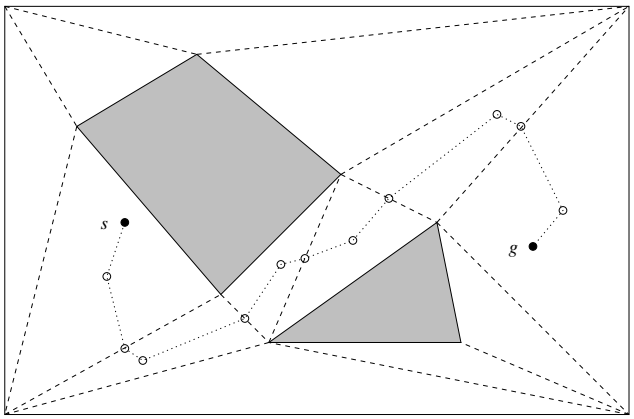


Figur 2.20: Eksempel på robot med endeligt konfigurationsrum. $\mathcal{C}_{obs}$ er markeret med sorte tern, og $\mathcal{C}_{free}$ er markeret med hvide tern. Robotten er kvadratisk og markeret med blåt, og har fire mulige bevægelser: Et tern op, ned, til højre eller til venstre. Der skal planlægges en rute fra startkonfigurationen s til målkonfigurationen g.

Planlægningsproblemet kan nu løses ved de gængse grafsøgningsalgoritmer såsom bredde-først, dybde-først eller Dijkstras algoritme. Specielt velegnet er dog A*-algoritmen, som man kan betragte som en udvidelse af Dijkstras algoritme. A*-algoritmen virker ved at anvende en heuristik, som estimerer omkostningerne eller længden af vejen fra en given konfiguration til målkonfigurationen. Algoritmen besøger således først samtlige knuder som kan nås fra startknuden og som har den laveste omkostning, derefter de knuder som kan nås fra de allerede besøgte knuder og som har den næstlaveste omkostning, osv. For at algoritmen kan finde den korteste vej, må omkostningsheuristikken altid give et underestimat af omkostningen. Dette kan fx for robotten på figur 2.20 være den euklidiske afstand eller bedre 1-normen, for jo tættere heuristikken er på den sande omkostning, jo færre knuder besøges, og jo hurtigere kører algoritmen.

2.4.2 Celleopdeling

Celleopdeling (eng: cell decomposition) kræver ikke et endeligt eller tælleligt konfigurationsrum. Til gengæld kræves en hensigtsmæssig eksplicit repræsentation af det frie rum $\mathcal{C}_{free}$ af konfigurationsrummet. Celleopdeling virker så ved at opdele $\mathcal{C}_{free}$ i "simple celler". Efter opdelingen kan naboceller repræsenteres som naboer i en graf, og denne graf kan nu gennemsøges for den korteste rute med de i afsnit 2.4.1 nævnte metoder. For et eksempel på celleopdeling, se figur 2.21.



Figur 2.21: Illustration af celleopdeling. Konfigurationsrummet er todimensionelt, og robotten er punktformet. $\mathcal{C}_{obs}$ er repræsenteret med grå polygoner. Som simple celler er her valgt trekanter. En vej fra startkonfigurationen s til målet g findes ved først at forbinde s til centrum af den trekant s ligger i, derefter forbindes nabotrekanter via deres fælles kant og til sidst forbindes g.

Selvom celleopdeling kan håndtere ikke-tællelige konfigurationsrum, har algoritmen dog nogle ulemper. $\mathcal{C}_{free}$ kan være meget svær at finde, fx i tilfældet for en bil-agtig robot, og også en bil-agtig robots ikke-holonome egenskaber giver problemer. Den kan jo ikke bevæge sig langs en lige linje, for så at rotere på stedet for at følge en lige linje i en anden retning.

2.4.3 Stikprøvebaserede planlægningsalgoritmer

Stikprøvebaserede (eng: sampling-based) planlægningsalgoritmer undersøger konfigurationsrummet ved en række stikprøver af dette. Ideen er at undgå den eksplicitte konstruktion af $\mathcal{C}_{free}$. I stedet spørger algoritmen en "sort kasse" (black box), om en bevægelse fra en konfiguration x_1 til en anden konfiguration x_2 kan findes uden kollision. Den sorte kasse står så for genereringen af bevægelsen og for kol-

lisionsbestemmelsen. På denne måde kan algoritmen adskilles fra den konkrete geometriske model, jf. [LaValle, 2004].

2.5 Statistisk bevægelsesplanlægning

Den valgte bevægelsesplanlægningsmetode er baseret på artiklen i [Svestka, 1993], og vil i det følgende blive omtalt som Švestkas metode. Švestkas metode er en stikprøvebaseret bevægelsesplanlægningsmetode, jf. afsnit 2.4.3. Grunden til at denne metode blev valgt er, at den er let at forstå og forholdsvis let at implementere, og så kan den i følge artiklen løse bevægelsesplanlægningsproblemet for en bil-agtig robot. Švestkas metode er nemlig rettet mod en helt konkret kinematik, nemlig kinematikken for en bil-agtig robot. Dog kan algoritmen sagtens løse bevægelsesplanlægningsproblemer for andre typer robotter, da den geometriske model for robotten er gemt i en sort kasse, som nævnt i afsnit 2.4.3.

Metoden består af to dele. Først en udforskning af konfigurationsrummet, hierarkisk inddelt i en global og en lokal metode. Švestka kalder den sorte kasse for den lokale metode, idet den her bliver anvendt mellem konfigurationer som ligger forholdsvis tæt i konfigurationsrummet. Dernæst udføres en udglatning af den ved udforskningen fundne, som regel temmelig "grimme", bevægelse. Herunder vil Švestkas metode blive gennemgået, og ellers henvises til [Svestka, 1993].

Da metoden herunder bliver anvendt på en helt specifik type robot, nemlig en bil-agtig robot, vil de tilsvarende termer blive anvendt, dvs. rute i stedet for bevægelse, position i stedet for konfiguration etc.

2.5.1 Den globale metode

Inddata består af en startposition s, en slutposition g og en mængde af forhindringer O. Metoden opbygger en ikke-orienteret graf $\mathcal{G}$, efterhånden som konfigurationsrummet udforskes. Det antages at en lokal metode eksisterer, som giver en kollisionsfri rute mellem to positioner, hvis den kan, og ellers fejler. Lad funktionen:

$$\texttt{exists\_path} \in \mathcal{C} \times \mathcal{C} \times O \rightarrow boolean \tag{2.59}$$

angive, om den lokale funktion returnerer en rute, eller om den fejler. Funktionen:

$$\texttt{construct\_path} \in \mathcal{C} \times \mathcal{C} \times O \rightarrow path \tag{2.60}$$

returnerer den rute, den lokale funktion udregner, hvis den ikke fejler.

Lad V være mængden af grafens knuder og E være mængden af grafens kanter. Pseudokode for den globale metode kan ses i tabel 2.2.

- $V = \{s, g\}$
- $E = \emptyset$
- `while` der ikke er en vej fra s til g i $\mathcal{G}$
  - c = En tilfældigt valgt fri position, $c \in \mathcal{C}_{free}$
  - V = $V \cup \{c\}$
  - $N(c)$ = En mængde af naboer til c, $N(c) \subseteq V$
  - E = $E \cup \{(c,n) \mid n \in N(c) \wedge \texttt{exists\_path}(c,n)\}$
- l_p = Vejen i $\mathcal{G}$ fra s til g, repræsenteret som en liste af positioner
- Konstruer en fri rute fra s til g ved at sammensætte de delruter, som `construct_path` giver, når den kaldes for hver par af på hinanden følgende knuder i l_p.

Tabel 2.2: Pseudokode for den globale metode.

Mængden $N(c)$ af naboer af c defineres som de $n \in V \setminus \{c\}$, for hvilke der gælder, at den euklidiske afstand i konfigurationsrummet mellem n og c er mindre end en vis maksimal afstand d_{max}. Desuden skal der gælde, at hvis der er flere af disse n som er sammenhængende i $\mathcal{G}$, medtages kun den knude n, i den sammnhængende mængde, som er euklidisk tættest på c. Grafen $\mathcal{G}$ vil således aldrig indeholde cykler.

2.5.2 Den lokale metode

Švestka definerer den lokale metode, så den set fra den globale metode er en sort kasse, som enten returnerer en rute mellem to givne positioner, eller fejler. Han giver flere forskellige forslag til lokale metoder for en bil-agtig robot, og evaluerer herefter metoderne efter, hvor hurtige de er til at finde en rute. De tre første forslag til en lokal metode gennemgås herunder, de resterende metoder bygger på

potentialfelter, og vil ikke blive gennemgået her. De giver nemlig iflg. Švestkas eksperimentielle afprøvning ikke bedre resultater end de metoder, som ikke bygger på potentialfelter, jf. [Svestka, 1993, p. 78f].

Švestka bruger en notation for ruter, som svarer til notationen gennemgået i afsnit 2.1.1, blot anvendes A i stedet for C og S i stedet for L. Švestka har tilsyneladende ikke læst artiklerne [Dubins, 1957] og [Reeds and Shepp, 1990], som ellers er helt fundamentale på området, for han refererer ikke til dem.

I Švestkas artikel beskrives kollisionsbestemmelse ikke. Det antages blot at der findes en funtion, som kan sige, om en position ligger inde i en forhindring, og at der findes en funktion, som kan sige, om en rute er kollisionsfri. Til dette formål kan kollisionsbestemmelsen gennemgået i afsnit 2.3, netop med henblik på dette, naturligvis bruges.

2.5.2.1 Den lokale ALA-metode

Švestkas ALA-metode finder en rute mellem to givne positioner, ved at finde den korteste rute γ mellem positionerne i klassen CSC. Er γ kollisionsfri returneres ruten, ellers fejler metoden. Artiklen nævner ikke konkret, hvorledes ruten findes, men her er det blev gennemgået i afsnit 2.1.

2.5.2.2 Den lokale LAL-metode

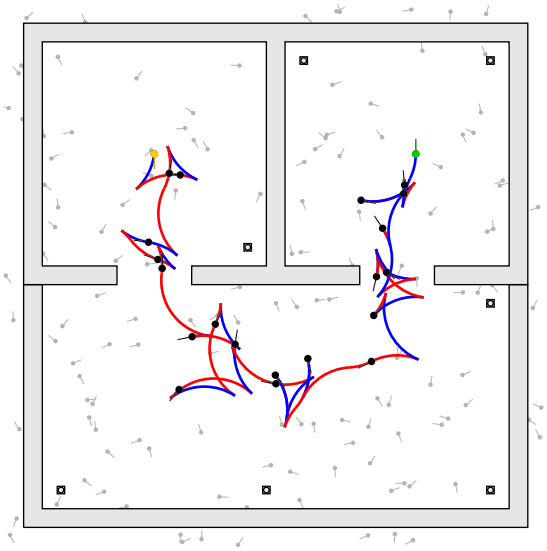
Švestkas LAL-metode finder en rute mellem to givne positioner, ved at finde den korteste rute γ mellem positionerne i klassen SCS. Er γ kollisionsfri returneres ruten, ellers fejler metoden. I følge afsnit 2.1.1, kan klassen SCS aldrig give den korteste rute mellem to positioner, og derfor kan valget synes mærkeligt, alligevel har den sine fordele, som det ses herunder. Det skal dog nævnes, at Švestka mest fokuserer på køretid og ikke så meget på at finde den korteste rute. Formler for klassen SCS blev fundet i afsnit 2.1.

2.5.2.3 Den lokale ALA-LAL-metode

Švestkas ALA-LAL-metode finder en rute mellem to givne positioner, ved først at prøve ALA-metoden. Hvis den har succes, returneres den rute, ALA-metoden fandt. I modsat fald prøves LAL-metoden, og hvis den har succes, returneres den fundne rute. Ellers fejler metoden. Ifølge eksperimentielle resultater, jf. [Svestka, 1993, p. 78f], er denne metode altid mindst lige så hurtig som både ALA-metoden og LAL-metoden, og for det meste hurtigere.

2.5.3 Udglatning

Når den globale metode har fundet en kollisionsfri rute γ mellem to positioner, som er mulig for en bil-agtig robot at følge, stopper metoden. Ruten γ vil dog, pga. metodens randomiserede virkemåde, formentlig være temmelig "grim", dvs. unødvendig kompliceret, se figur 2.22.



Figur 2.22: Illustration af resultatet af ruteplanlægningen på et kort over tre rum. Der skal findes en rute fra startpositionen $(2;\ 5,5;\ \frac{3\pi}{2})$ markeret med gult til målpositionen $(5,5;\ 5,5;\ \frac{\pi}{2})$ markeret med grønt. Kortet er $7,25 \times 7,25$ meter stort. Samtlige positioner i grafen er markeret med lysegråt, på nær dem, som ruten går igennem. De er markeret med sort. Et blåt kurvesegment betyder, at ruten skal følges forlæns, et rødt betyder at ruten skal følges baglæns. Den samlede rutelængde er $30,96$ meter.

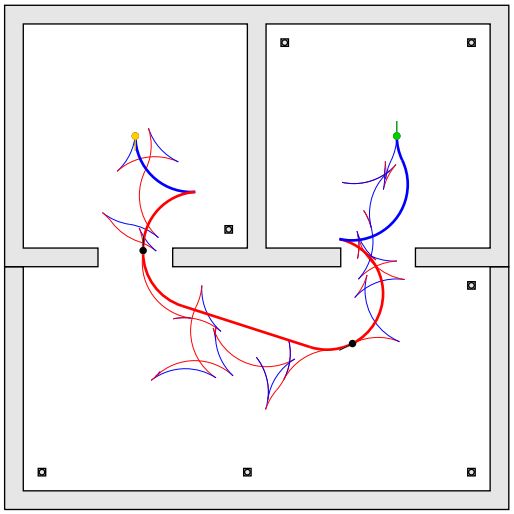
Dette råder Švestka bod på, ved at foretage udglatning af ruten. Udglatningen vil generelt fjerne unødvendige kørselsretningsskift, dvs. skift mellem at køre fremad og bagud. Desuden vil udglatningen forsøge at fjerne overflødige omveje, så unødvendigt snoede dele af ruten vil blive erstattet af mere direkte veje. I alt resulterer udglatningen i en garanteret højest lige så lang, men generelt væsentlig kortere rute, som formentlig vil indeholde færre kørselsretningsskift. Švestka foreslår to former for udglatning, som gennemgås herunder.

2.5.3.1 Positionsreduktion

Når den globale metode først har fundet en vej gennem grafen $\mathcal{G}$, består denne vej af en liste l_p af positioner, hvor den første position er startpositionen s og den sidste position er målpositionen g, se afsnit 2.5.1. Før denne liste af positioner oversættes til en rute vha. `construct_path`, foretages en positionsreduktion[2].

Positionsreduktionen foretages ved at udvælge en tilfældig position fra positionslisten, som ikke er s eller g: $p_i \in l_p \setminus \{s, g\}$. Nu forsøges p_{i-1} og p_{i+1}, som er hhv. den forrige og den næste position i l_p forbundet ved den lokale metode. Lykkedes dette, fjernes p_i fra l_p. Dette gentages et vist antal gange.

Hver gang det lykkes at fjerne et punkt fra l_p, vil den resulterende rute med meget stor sandsynlighed blive kortere, men i hvert fald ikke længere. Derfor vil den samlede positionsreduktion resultere i en højest lige så lang, men generelt langt kortere rute. Antallet af retningsskift kan der ikke siges noget specifikt om, men generelt vil det også falde. Se figur 2.23 for et eksempel på positionsreduktion.



Figur 2.23: Ruteplanlægningen fra figur 2.22, men nu med den positionsreducerede rute markeret med fed streg. Den oprindelige rute er markeret med tynd streg. Den reducerede længde er $9,60$ meter.

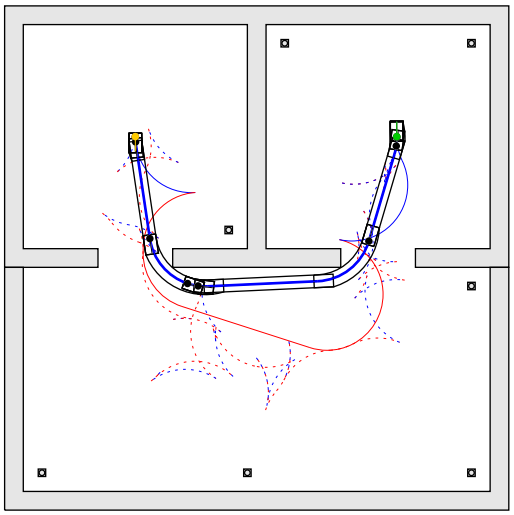
[2] Švestka bruger begrebet "graph smoothing", som egentligt er misvisende, da det er en liste af positioner, som udglattes, ikke grafen selv.

2.5.3.2 Ruteudglatning

Når positionsreduktionen er foretaget, oversættes listen af positioner l_p til en rute γ.

Ruteudglatningen foretages ved at udvælge to tilfældige positioner p_1 og p_2 på ruten, hvor p_1 ligger før p_2. Nu forsøges p_1 og p_2 forbundet ved den lokale metode. Lykkedes dette, erstattes rutestykket mellem p_1 og p_2 i γ med det nye rutestykke fundet med den lokale metode. Dette gentages et vist antal gange.

Ruteudglatningen vil, lige som positionsreduktionen, altid resultere i en højest lige så lang men generelt kortere rute, og antallet af retningsskift vil generelt også falde. Faktisk er ruteudglatningen en bedre metode til at udglatte og dermed forkorte ruten end positionsreduktionen. Dette kan ses ved, at hvis der foretages tilstrækkeligt mange iterationer af ruteudglatningen, vil p_1 og p_2 på et tidspunkt blive valgt vilkårligt tæt på p_{i-1} og p_{i+1} fra positionsreduktionen. Ruteudglatningen har dog den ulempe, at den er beregningsmæssigt meget tungere end positionsreduktionen, jf. [Svestka, 1993, p. 39f]. Derfor giver det god mening, først at foretage positionsreduktion, og derefter yderligere udglatte med ruteudglatningen. For et eksempel på ruteudglatning se figur 2.24.



Figur 2.24: Ruteplanlægningen fra figur 2.23, men nu med den både positionsreducerede og ruteudglattede rute markeret med fed streg. Den positionsreducerede rute er markeret med tynd streg og den oprindelige rute er markeret med tynd stiplet streg. Det areal robotten passerer over ved at følge ruten, er indrammet med tynd sort streg. Den resulterende rutes længde er $6,33$ meter.

Det formodes, at grunden til, at ruteudglatningen er så meget mere beregningsmæssigt tungere end positionsreduktion, er fordi hver iteration af positionsreduktionen er meget lokal, dvs. kigger på et forholdsvist lille stykke af ruten, hvorimod ruteudglatningen slet ikke er lokal, da vilkårligt store dele af ruten kan blive udvalgt i hver iteration. Dermed har hver iteration i positionsreduktionen større chance for at finde en kortere rute end hver iteration i ruteudglatningen.

2.5.4 Implementation

I det følgende bliver det beskrevet, hvordan ruteplanlægningsalgoritmen i tabel 2.2 er implementeret.

Grafen $\mathcal{G}$ er repræsenteret med multilister, jf. [Liljedahl and Christensen, 1992]. Dette er en effektiv datastruktur, specielt når der skal indsættes knuder i den, hvilket tager $O(1)$ tid, idet for hver iteration af algoritmen bliver en ny knude ofte indsat. Desuden er multilister fordelagtige, når grafen som skal repræsenteres har forholdsvis få kanter i forhold til antallet af knuder. Dette har $\mathcal{G}$, da den til en hver tid er en skov af træer, idet der aldrig tilføjes kanter som vil resultere i cykler i $\mathcal{G}$.

Programmet kan dog godt håndtere cykler i $\mathcal{G}$. Disse kan være fordelagtige, hvis den korteste rute ønskes fundet, se afsnit 2.5.6. Derfor er Dijkstras algoritme også implementeret, så den korteste vej i $\mathcal{G}$ kan findes.

For at finde ud af om der findes en vej fra start til mål i $\mathcal{G}$ (uden cykler) på en effektiv måde, så det kan konstateres ved hver iteration af algoritmen uden at påvirke køretiden væsentligt, har hver knude tilknyttet et komponentnummer. Når en knude n_1 forbindes til en anden knude n_2, opdateres komponentnummeret for n_1 til komponentnummeret for n_2. Forbindes to komponenter, dvs. træer, opdateres den ene knudes komponentnummer så de får fælles komponentnummer. På denne måde kan det altid konstateres om start og mål er forbundne, ved blot at konstatere om de har samme komponentnummer.

For at kunne genbruge den genererede graf for et givent kort, kan grafen gemmes og indlæses fra en fil. Således behøves en graf for et givent kort kun at blive genereret helt fra bunden en enkelt gang. Til efterfølgende forespørgelser kan den gemte graf genbruges, også for andre end de oprindelige start- og målpositioner. Den indlæste graf kan blot tilføjes ekstra knuder indtil de nye start- og målpositioner er forbundne.

For at mindske køretiden det tager at finde (euklidiske) naboer til en knude, hvilket anvendes i operationen $N(c)$, indsættes samtlige knuder, dvs. positioner, i et range-træ, jf. afsnit 2.3.3.3. Dette giver en køretid for $N(c)$ på $O(\log n + k)$, hvor n er antallet af knuder i $\mathcal{G}$ og hvor k er antallet af knuder, hvis y-koordinater lig-

ger indenfor naboafstanden d_{max}. k er som regel meget mindre end n, men vokser desværre lineært med n. Hvordan dette kan forbedres bliver behandlet i afsnit 2.5.6.

Den lokale metode som anvendes her, er den korteste rute indenfor klasserne CCC og CSC. Om denne strategi er bedre end kun at anvende ruten fra CSC er ikke blevet afprøvet pga. det i indledningens afsnit 1.2.1 beskrevne problem. Ligeledes havde det også været ønskværdigt at implementere Švestkas lokale ALA-LAL (svarende til CSC-SCS) metode. Her er kun funktionen til at finde ruten indenfor klassen SCS implementeret, den er ikke blevet integreret i ruteplanlægningen.

2.5.5 Afprøvning

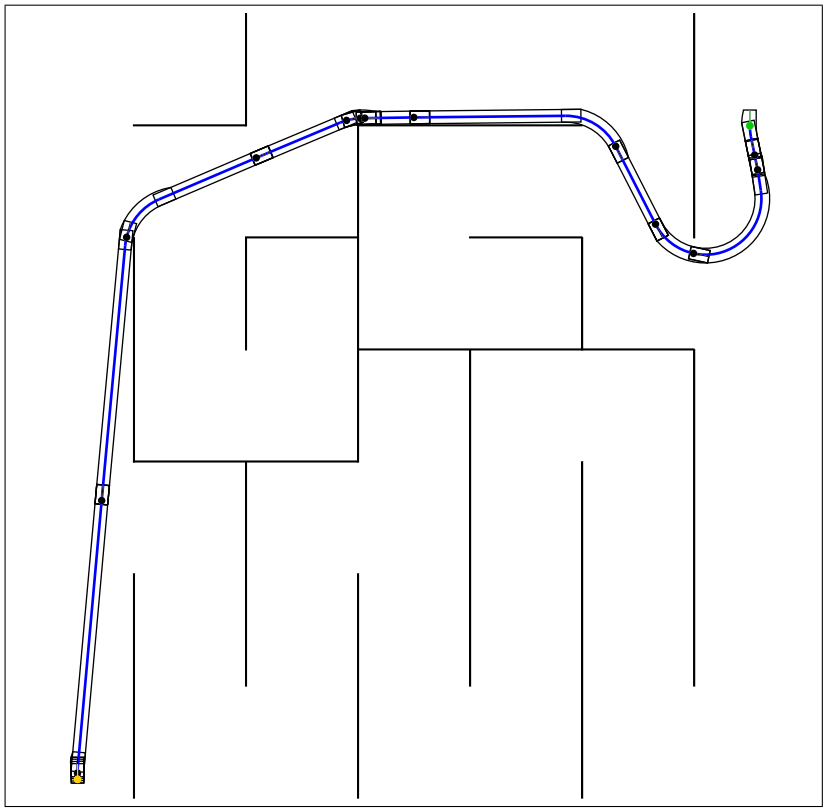
I dette afsnit vil ruteplanlægningen blive afprøvet. Der vil ikke blive foretaget en formel tilbundsgående afprøvning, men metodens fordele og ulemper vil blive beskrevet. Som det kunne ses i figur 2.24, finder algoritmen en fornuftig rute fra start til mål på kontorkortet. Da algoritmen jo har et vist element af tilfældighed, er der dog en vis variation i den løsning algoritmen finder, hvis algoritmen anvendes flere gange på samme problem. Denne variation er dog mest udtalt ved "problematiske" kort. Herunder vil specielt to tilfælde blive fremhævet. Konklusionen i afsnit 2.5.6 giver forslag til løsning på de her belyste problemer.

2.5.5.1 Flere alternative løsningsmuligheder

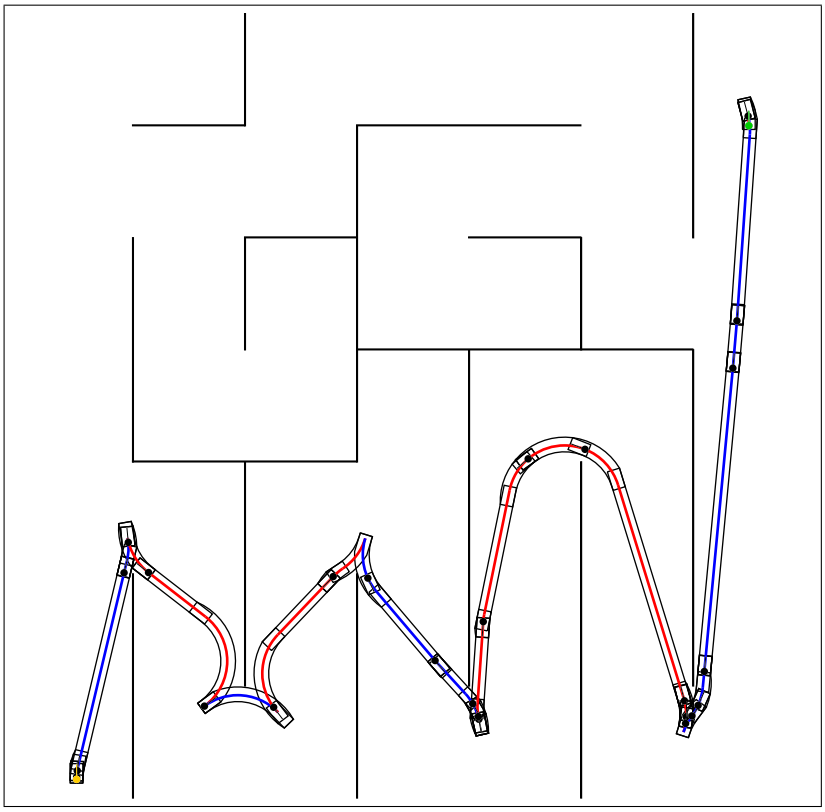
Figur 2.25 viser et kort over en labyrint. Der er grundlæggende to muligheder for at finde fra start til mål, og på figuren er den korte rute fundet. Figur 2.26 viser samme kort, men her er den lange rute fundet. Ved at køre algoritmen 25 gange på samme kort, blev den korte rute fundet 18 gange (72 %) og den lange rute blev fundet 7 gange (28 %). Generelt er sandsynligheden for at finde den korteste rute størst, med mindre den er specielt "snæver", hvilket kan gøre ruten svær at finde. Grunden til, at den korteste rute ikke altid findes, er at algoritmen stopper så snart en vej er fundet i grafen. Der er ikke noget krav om, at det skal være den korteste vej der findes.

2.5.5.2 Retningsskift er gratis

Figur 2.27 viser et kort, hvor robotten skal bakke ind i en smal korridor i højre side. På figuren er vist et tilfælde, hvor algoritmen finder en fornuftig rute.

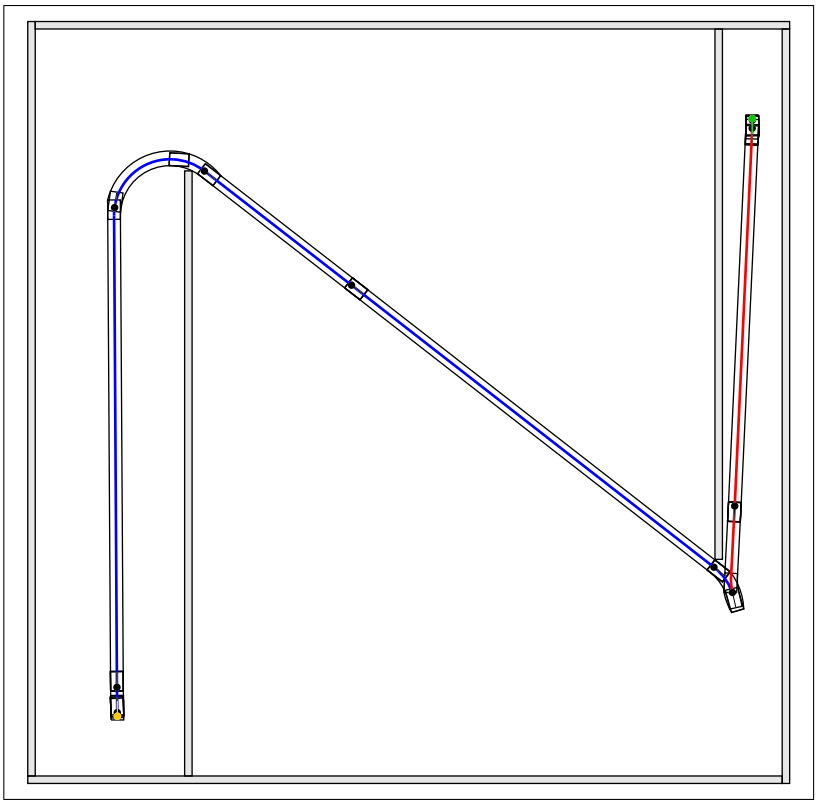


Figur 2.25: Kort over en labyrint med to alternative løsningsmuligheder, når der skal findes en rute fra start (markeret med gult) til slut (markeret med grønt), og blindgyder ikke tælles med som løsningsmuligheder. Her har ruteplanlægningen fundet den korteste rute gennem labyrinten.

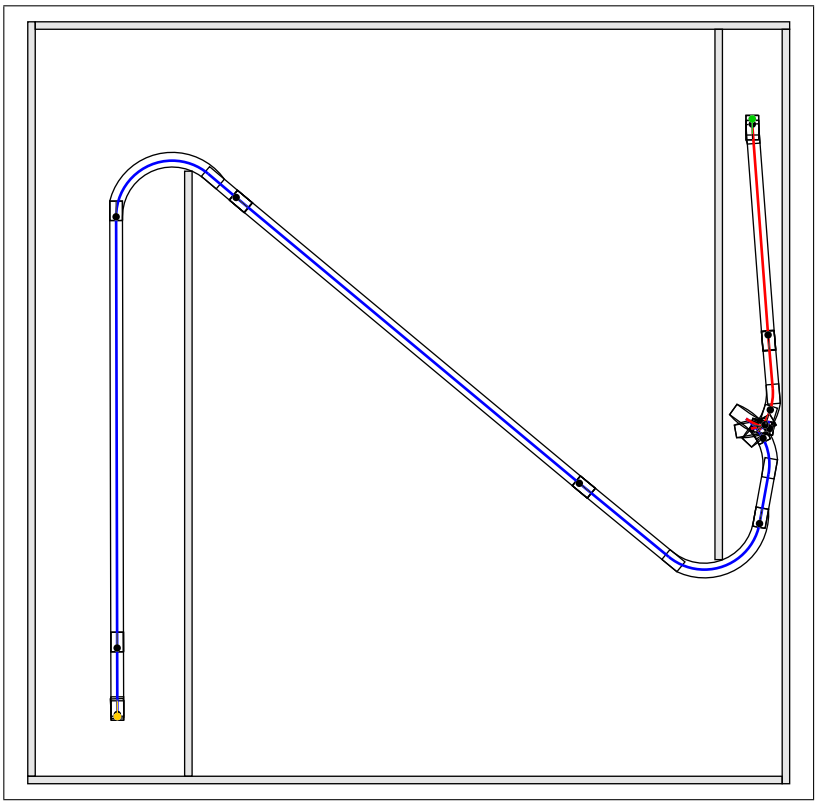


Figur 2.26: Samme kort som på figur 2.25, men her har ruteplanlægningen fundet den længste rute.

Korridorens bredde er valgt, så der netop er plads til at robotten kan vende inde i korridoren, men det kræver mange skift mellem forlæns og baglæns kørsel. På figur 2.28 er vist et tilfælde, hvor algoritmen finder en rute med et sådant retningsskift. Selv ruteudglatningen kan i dette tilfælde ikke flytte den uheldige placering af vendestedet. Selve den komplicerede vendemanøvre gør kun rutens længde marginalt længere, men hvis en fysisk robot skal foretage de mange skift mellem forlæns og baglæns kørsel vil det naturligvis tage væsentligt længere tid end den enkle løsning i figur 2.27. Hvis de mange skift mellem forlæns og baglæns kørsel skal undgås, må det indbygges i algoritmen, at de har en vis omkostning.



Figur 2.27: Kort med to korridorer. Specielt den højre korridor er udfordrende for ruteplanlægningen, da den er meget smal. Den er lige netop bred nok til at robotten kan vende inde i korridoren.
Der skal findes en rute fra start (markeret med gult) til slut (markeret med grønt). Bemærk, at slutpositionen peger nedad, dvs. $\theta_2 = \frac{3\pi}{2}$. Den korteste løsning er at robotten bakker hele vejen ind i den smalle korridor, og det er den løsning, der er fundet her.



Figur 2.28: Samme situation som på figur 2.27, men her er robotten startet med at køre forlæns ind i den smalle korridor, og vender så undervejs, hvorved robotten skal skifte mange gange mellem at bakke og at køre fremad.

2.5.5.3 Ruteplanlægningens køretid

Algoritmens køretid afhænger naturligvis af kortets størrelse, men den afhænger også af, hvor kompliceret kortet er. Det er ikke helt intuitivt klart, hvad der gør et kort kompliceret. Fx kræver korridorkortet i figur 2.27 to til tre gange så mange knuder i grafen, før en løsning findes end labyrintkortet i figur 2.25. Dette skyldes, at labyrintkortet har forholdsvis brede gange, hvor korridorkortet har en lang smal korridor. Den smalle korridor har på samme tid et forholdsvist lille areal, som gør sandsynligheden for at knuder tilføjes indenfor arealet mindre, og den smalle korridor begrænser samtidig stærkt det antal af de tilføjede knuder indenfor arealet, som faktisk resulterer i frugtbare delruter.

I det ekstreme tilfælde, hvor der simpelthen ikke findes en mulig rute mellem start og mål, fejler algoritmen ikke, men kører i det uendelige. Dette kan der naturligvis rettes op på, ved at algoritmen opgiver når et vist antal knuder er tilføjet grafen.

2.5.6 Delkonklusion og mulige forbedringer

I dette afsnit blev den statistiske ruteplanlægning gennemgået. Algoritmen anvender en global og en lokal metode. Pseudokode for den globale metode blev vist, og den lokale metode svarer til det i afsnit 2.1, 2.2 og 2.3 gennemgåede stof. Da den fundne rute som regel er temmelig "grim", dvs. unødigt kompliceret, blev to metoder til udglatning af denne beskrevet. Til sidst blev implementationen af algoritmen beskrevet, og nogle særtilfælde, som algoritmen har svært ved at løse blev beskrevet.

I det følgende vil nogle forslag til forbedring af algoritmen blive gennemgået. Forslagene falder i to kategorier, nemlig dem som forbedrer algoritmens køretid, og dem som får algoritmen til at give bedre svar. Med bedre svar menes her resulterende ruter, som det vil tage robotten kortere tid at følge fra start til mål. Til sidst i afsnittet vil valget af selve algoritmen blive diskuteret.

2.5.6.1 Forbedring af algoritmens køretid

Det er operationen $N(c)$, som tager hovedparten af køretiden i hver iteration af algoritmen, når der er tilføjet mange knuder til grafen. Køretiden for $N(c)$ er $O(\log\ n\ +\ k)$, men da k som nævnt vokser lineært med n, vil dette k være det dominerende led, når n er stor. Når der skal søges i et todimensionelt rektangel, er et kd-træ normalt særdeles velegnet som datastruktur, jf. [de Berg et al., 2000, afsnit 5.2]. Et kd-træ giver en søgetid på $O(\sqrt{n}+k)$, hvor k er antallet af rappor-

terede punkter, hvilket er meget bedre end den nuværende løsning implementeret med et range-træ, jf. afsnit 2.3.3.3. Desværre er det særdeles tidskrævende at indsætte nye punkter dynamisk i et kd-træ, hvilket ruteplanlægningsalgoritmen gør meget ofte. [Procopiuc et al., 2002] beskriver et dynamisk kd-træ, som kan håndtere meget frekventielle opdateringer effektivt, men algoritmen er indviklet. [Atramentov and LaValle, 2002] foreslår en algoritme, som vedligeholder en vektor af kd-træer, og som har en søgetid som et kd-træ, men som kan håndtere indsættelse af nye punkter i logaritmisk tid. Denne metode virker som et oplagt valg, omend den er noget mere kompliceret at implementere end range-træet. Hvis et kd-træ implementeres, kan dette også bruges i stedet for range-træet som anvendes til kollisionsbestemmelsen i afsnit 2.3.3.

Algoritmens køretid kan også forbedres ved at gøre naboafstanden d_{max}, som $N(c)$ anvender, adaptiv. Når der er meget få knuder i grafen vil $N(c)$ sjældent finde nogen naboer, og når der er mange knuder i grafen vil $N(c)$ finde mange naboer, hvis d_{max} holdes fast.

Det foreslåes her at gøre d_{max} adaptiv, således at d_{max} afhænger af antallet n af knuder i grafen. En mulighed kunne være at sætte $d_{max} = k\frac{1}{\sqrt{n}}$, hvor k er en konstant, som afhænger af størrelsen af kortets areal. Herved vil det fundne antal naboknuder være omtrent konstant, uafhængig af antallet af knuder i grafen. Sandsynligheden for at finde naboer, når der er meget få knuder i grafen vil så være god, og samtidig vil køretiden for $N(c)$ være rimelig, når der er mange knuder i grafen. Det virker også rimeligt, at den lokale metode bliver mere og mere lokal, jo flere knuder der er i grafen. Der vindes ikke noget ved at forbinde knuder, som er langt fra hinanden, når der er mange knuder i grafen, idet de med stor sandsynlighed allerede er forbundet på et tidligere tidspunkt. Konstanten k skal naturligvis vælges med omhu. Vælges den for stor, giver det ingen køretidsforbedring, tværtimod. Vælges den for lille, bliver start og mål sandsynligvis aldrig forbundet.

Den tilfældigt valgte konfiguration c findes som udfald af en fordeling. I beskrivelsen af den globale metode i afsnit 2.5.1 var det underforstået, at denne fordeling er en ligefordeling over hele konfigurationsrummet. Algoritmens køretid kan dog forbedres ved at anvende andre fordelinger end en ligefordeling. Ideen er, at det er let at finde vej i store åbne områder, hvor der ikke er nogen forhindringer. Det er derimod svært, dvs. kræver stor tæthed af de valgte konfigurationer, når der skal findes vej tæt på forhindringerne, fx gennem smalle korridorer. For at afspejle dette, kan en fordeling anvendes, som giver større sandsynlighed for at konfigurationer tilføjes tæt på forhindringerne, end langt fra forhindringerne. Både [Svestka, 1993, kapitel 5], [LaValle, 2004, afsnit 5.6.3] og [Aarno et al., 2004] foreslår en række heuristikker til forbedring af fordelingen af de valgte konfigurationer. Det ville være interessant at implementere og afprøve disse, for at evaluere heuristikkernes

indflydelse på algoritmens køretid.

2.5.6.2 Optimering af ruten

Med optimering af en given rute menes her reducering af den tid, robotten vil være om at køre langs den givne rute. Denne tid kan optimeres på to måder. Enten kan ruten gøres kortere, eller også kan antallet af skift mellem forlæns og baglæns kørsel reduceres. Antallet af retningsskift kan reduceres, ved at tilføje ruten lidt kunstig ekstra længde for hvert retningsskift, og så optimere rutens længde. I det følgende gives nogle forslag til, hvordan ruteplanlægningsalgoritmen kan optimeres, så den giver kortere ruter som uddata. Rutens længde kan optimeres både ved opbygningen af grafen og under udglatningen af den fundne rute.

Ved opbygning af grafen stopper algoritmen i samme øjeblik en vej i grafen findes fra start til mål. En metode til at give kortere resulterende ruter er at tillade cykler i grafen. Den simpleste løsning er at tillade forbindelse mellem den tilfældigt valgte position c og alle dens naboer fundet ved $N(c)$, jf. pseudokoden i tabel 2.2. Dette kalder [Nieuwenhuisen and Overmars, 2004] for nearest-n-metoden. Artiklen definerer *brugbarheden* af en kant, og bruger denne definition til kun at tilføje kanter til de nabopositioner til en position c, som har en vis mindste brugbarhed. Dette giver en bedre køretid, men dog ikke en kortere resulterende rute end nearest-n-metoden.

Švestka beskriver to former for udglatning, som beskrevet i afsnit 2.5.3. Man kan dog tænke sig en række andre metoder til ruteudglatning.

Analogt til positionsreduktionen kan to tilfældige positioner p_1 og p_2 udvælges fra positionslisten, $p_1 \neq p_2$, og de kan forsøges forbundet ved den lokale metode. Lykkes det, fjernes de mellemliggende positioner i positionslisten. Denne metode vil formentlig give kortere ruter end positionsreduktionen, men køretiden vil formentlig være større.

Der kan også tilføjes lidt støj til en tilfældig udvalgt position enten i positionslisten eller på ruten generelt, og derefter konstateres, om den resulterende rute er kortere (og kollisionsfri). Man kan tænke sig støjen tilføjet som en "nedkøling" af positionerne, så der startes med at tilføjes meget støj, og så i de efterfølgende iterationer tilføjes mindre og mindre støj.

2.5.6.3 Andre ruteplanlægningsalgoritmer

Švestkas statistiske ruteplanlægning er ikke den eneste algoritme, som kan løse ruteplanlægningsproblemet for bil-agtige robotter. At komme ind på dem alle her

vil være meget omfangsrigt, og derfor henvises i stedet til bøgerne [LaValle, 2004] og [Luca et al., 1999] som giver en grundig gennemgang af det generelle bevægelsesplanlægningsproblem.

2.6 Delkonklusion og mulige forbedringer

Kapitlet startede med at definere et problem, nemlig ruteplanlægningsproblemet for en bil-agtig robot, med hensyn til forhindringer. Afsnit 2.1 løste problemet uden hensyntagen til forhindringer, her blev det endda beskrevet, hvordan den korteste rute findes. Formler for tre klasser af ruter blev udledt til brug for den videre ruteplanlægning. I afsnit 2.2 blev et kort over forhindringer defineret. Afsnit 2.3 gennemgik, hvorledes det bestemmes, om en rute vil give anledning til kollision med de på et givet kort definerede forhindringer, hvis den følges af robotten. Det bemærkes, at dette svarer til en implicit repræsentation af $\mathcal{C}_{obs}$. Afsnit 2.4 gennemgik en række planlægningsalgoritmer, samt nogle fordele og ulemper ved disse. Til sidst blev den valgte ruteplanlægningsalgoritme beskrevet i afsnit 2.5, og det blev vist, at den kan løse problemet. Desuden blev en række specialtilfælde, som algoritmen havde svært ved at løse, gennemgået.

Hvis ruteplanlægningen skal forbedres, kan det grundlæggende gøres på to måder: Enten kan de enkelte dele af den gennemgåede løsning forbedres, eller også kan selve præmisserne for ruteplanlægningen ændres, dvs. ændring af modellen af den bil-agtige robot, jf. indledningen til kapitlet, og specielt formel 2.1.

I delkonklusionerne til dette kapitels afsnit er forbedringer til de respektive afsnit allerede beskrevet, dog uden at se ud over afsnittenes præmisser. Dette vil blive gjort i det følgende.

2.6.1 Begrænset hjulvinkelhastighed

Modellen af den bil-agtige robot, jf. formel 2.1, tager ikke hensyn til den fysiske robot Murphys dynamik. Specielt blev der ikke taget hensyn til begrænsningen på Murphys acceleration $\dot{v}_R$ og hjulvinkelhastigheden $\dot{\phi}$. Når Murphy skal følge en rute fx af formlen S^+C^+, må robotten faktisk stoppe helt op i punktet hvor S og C mødes, for at ændre hjulvinklen, hvis ruten skal følges helt eksakt. Hvordan der bliver taget hensyn til dette gennemgås i kapitel 4, men det ville være hensigtsmæssigt, hvis der blev taget højde for problemet allerede i ruteplanlægningsfasen.

Hvis robottens hastighed v_R er konstant, og forhjulsvinklen ændres med en konstant hastighed $\dot{\phi}$, bevæger en bil-agtig robot sig langs en Cornu spiral, jf.

[Kostov and Degtiariova-Kostova, 1995]. En Cornu spiral er en kurve, hvis krumning er proportional med kurvens længde. Cornu spiraler anvendes derfor også ved anlægning af veje. Når en lige strækning af fx en motorvej skal gå ind i et sving, forbindes den lige strækning og svinget (som har en konstant krumning) med et stykke af en Cornu spiral. På den måde kan føreren af en bil, som kører på vejen, nøjes med at dreje rattet med en konstant hastighed, når svinget nås. For mere om Cornu spiraler, se [Fabricius-Bjerre, 1977] og [Vestergaard, 2004].

Desværre er repræsentationen af en Cornu spiral ret kompliceret. En parameterfremstilling af en Cornu spirals kurve involverer to integraler, de såkaldte Fresnel integraler.

Hvis Cornu spiraler skal medtages i ruteplanlægningen, behøves kun den lokale metode ændret. Dette er dog også kompliceret nok. Det skal udledes, hvorledes en rute fra en position til en anden position uden hensyntagen til forhindringer beregnes, med både linjestykker, cirkelbuer og Cornu spiraler. Desuden skal det passerede areal for en Cornu spiral udledes, til brug for kollisionsbestemmelsen. For en sådan løsning, se [Scheuer and Fraichard, 1997] og [Fraichard and Scheuer, 2004].

2.6.2 Andre forslag til forbedringer

I den gennemgåede ruteplanlægning har det været underforstået, at en optimal rute er en rute, som robotten er så kort tid som muligt om at følge. Andre optimalitetsmål kunne tænkes. Dette kunne fx være et ønske om den sikreste rute, altså en rute som fører robotten så langt uden om forhindringerne som muligt, eller eventuelt et kompromis mellem sikkerheden og den tid robotten skal bruge for at følge ruten. Et andet optimalitetsmål kunne være, at robotten er så sikker som muligt på sin position, ved at vælge en rute der sikrer maksimal udsigt til landmærkerne.

Kapitel 3

Positionsestimering

Positionsestimering er et nøgleproblem inden for området mobile robotter, jf. [Thrun et al., 2000]. For at kunne anvende resultatet af ruteplanlægningen, må robotten kende sin position. Kun hvis robotten kender sin position, kan den vide, hvor på en given rute den befinder sig - en nødvendighed for at kunne planlægge og udføre den videre kørsel.

Som beskrevet i arbejdsbeskrivelsen, jf. appendiks D, skal robotten anvende dead reckoning, dvs. positionsestimering ud fra information om, hvor langt og i hvilken retning robotten har kørt. Dette vil blive gennemgået i afsnit 3.1 herunder. Som nævnt i arbejdsbeskrivelsen skal robotten bestemme afstand og retning til visse landmærker med kendt position. Dette gennemgås i afsnit 3.2. I afsnit 3.3 vil det dernæst blive vist, hvordan positionsestimatet fra dead reckoningen kan kombineres med informationen fra landmærkegenkendelsen og på den måde give en mere præcis positionsestimering. Til sidst vil resultatet af kapitlet blive diskuteret i afsnit 3.4.

3.1 Dead reckoning

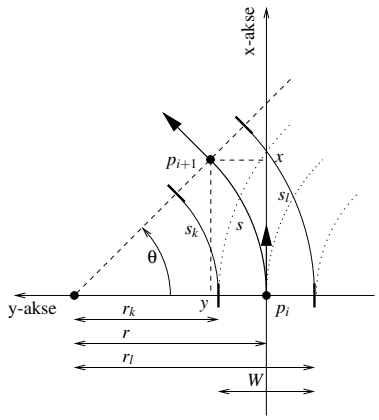
Den anvendte robot Murphy har et tachometer på hvert baghjul. Et tachometer fungerer på den måde, at for hver gang hjulet, det er monteret på, har drejet et vist stykke, giver tachometeret en puls, et “tick”. Dette registreres af robottens elektronik, og et register i dens hukommelse inkrementeres. Ud fra tachometrenes værdier kan robottens position beregnes. Beregningen af positionen, når robotten har kørt et vist stykke, bygger på den position robotten havde før det kørte stykke. En sådan positionsberegning kaldes *dead reckoning* i litteraturen på området

og dette udtryk vil derfor blive brugt i det følgende i stedet for det danske udtryk bestikregning.

I det følgende vil positionsberegningen blive vist. Først ud fra antagelsen, at baghjulene ruller perfekt på underlaget. Herefter korrigeres det fundne resultat ud fra et forsøg foretaget på Murphy-robotten. Sidst estimeres, hvor nøjagtigt det korrigerede resultat er.

3.1.1 Beregning af positionsændring

Ved at aflæse registret for hvert tachometer og multiplicere med en skaleringskonstant kan det bestemmes, hvor langt hvert baghjul har kørt på underlaget. De kørte længder betegnes s_h og s_v for hhv. højre og venstre baghjul. Kendes robottens position p_i, og robotten herefter kører et vist stykke, kan robottens nye position p_{i+1} beregnes ud fra, hvor langt hvert baghjul har roteret. Det antages her, at robottens baghjul ruller perfekt på underlaget uden at skride. Dette tilfælde blev vist i [Rode and Kjeldsen, 2004], og udledningen af formlerne herfor vil blive resumeret i det følgende, hvilket resulterer i formel 3.10. Se figur 3.1, som definerer de anvendte betegnelser. Algoritmen er iterativ og antager, at hver gang robotten har kørt et vist stykke, har den fulgt en cirkelbue. Desuden antages det, at robotten er bil-agtig, dvs. med faste baghjul og drejelige forhjul, og forhjulene modelleres som var robotten en tricykel, dvs. de drejelige forhjul modelleres som et enkelt drejeligt hjul placeret midt på forakslen.



Figur 3.1: Dead reckoning. Bemærk at x-aksen peger opad og y-aksen peger mod venstre, koordinatsystemet er relativt til p_i. p_i er startpositionen, p_{i+1} er slutpositionen, W er afstanden mellem hjulene. s_k er den strækning, som det hjul, der har kørt kortest, har kørt. s_l er den strækning, som det hjul, der har kørt længst har kørt.

Ud fra figuren ses at:

$$s_k = r_k\theta, \qquad s_l = r_l\theta \qquad \Rightarrow \qquad \frac{s_k}{r_k} = \frac{s_l}{r_l} = \theta \tag{3.1}$$

I det ovenstående er s_k og s_l defineret som:

$$s_k = \min\{s_h,\ s_v\}, \qquad s_l = \max\{s_h,\ s_v\} \tag{3.2}$$

Der gælder, at $r_l = r_k + W$, hvor W er afstanden mellem hjulene, så r_k kan findes som:

$$\frac{s_k}{r_k} = \frac{s_l}{r_k + W} \qquad \Rightarrow \tag{3.3}$$

$$r_k = \frac{s_k W}{s_l - s_k} \tag{3.4}$$

Drejeradiussen r kan findes ud fra r_k:

$$r = r_k + \frac{W}{2} \tag{3.5}$$

Den drejede vinkel θ kan findes ud fra r_k, jf. formel 3.1:

$$\theta = o\frac{s_k}{r_k} \tag{3.6}$$

Her betegner o fortegnet for omløbsretningen, som findes som:

$$o = \begin{cases} 1 & \text{hvis } s_v < s_h \\ -1 & \text{hvis } s_v > s_h \end{cases} \tag{3.7}$$

x og y kan findes som:

$$x = r\sin\theta \tag{3.8}$$

$$y = or(1 - \cos\theta) \tag{3.9}$$

Den nye position kan nu findes som:

$$p_{i+1} = p_i + (x,\ y,\ \theta) \tag{3.10}$$

I udledningen af formel 3.10 blev det antaget, at $s_k \neq s_l$, samt $s_k > 0$. Disse særtilfælde skal der naturligvis tages højde for, hvilket også er gjort i programmet.

Længden af det kørte stykke mellem hver opdatering af positionen skal vælges så det hverken er for kort eller for langt: Vælges det for langt, vil en eventuel ændring i hjulvinklen blive "overset", da algoritmen antager, at hjulvinklen er fast under kørslen af stykket. Vælges det for kort vil tachometermålingernes diskretisering af de målte afstande få stor indflydelse på positionsberegningen.

I [Rode and Kjeldsen, 2004] blev den optimale længde fundet empirisk: Ved en tachometeropløsning på 160 ticks pr. meter skal det kørte stykke være ca. 100 mm. Da Murphy har en tachometeropløsning på ca. 615 ticks pr. meter, skal det kørte stykke være tilsvarende mindre, nemlig ca. 26 mm ($100 \cdot \frac{160}{615} \approx 26$). Med det kørte stykke menes det tilbagelagte stykke (den tilbagelagte cirkelbue), som midterpunktet af bagakselen har tilbagelagt. Dette kan, jf. figur 3.1, beregnes som:

$$s = \frac{s_h + s_v}{2} \tag{3.11}$$

Dette kan man overbevise sig om ved at løse nedenstående to ligninger med to ubekendte (mellemregningerne udelades):

$$s_k = \theta(r - \frac{W}{2}) \tag{3.12}$$

$$s_l = \theta(r + \frac{W}{2}) \tag{3.13}$$

Ud fra r og θ kan s findes, hvilket giver formel 3.11.

3.1.2 Afprøvning

Algoritmen i det forrige afsnit antog, at robottens baghjul rullede perfekt på underlaget uden at skride. Det blev vist i [Rode and Kjeldsen, 2004], at under denne antagelse virker algoritmen praktisk talt perfekt. I realiteten vil antagelsen om, at

baghjulene ruller perfekt dog ikke holde på den fysiske robot. I det følgende sammenstilles det teoretiske tilfælde med udfaldet af en afprøvning på den fysiske robot Murphy.

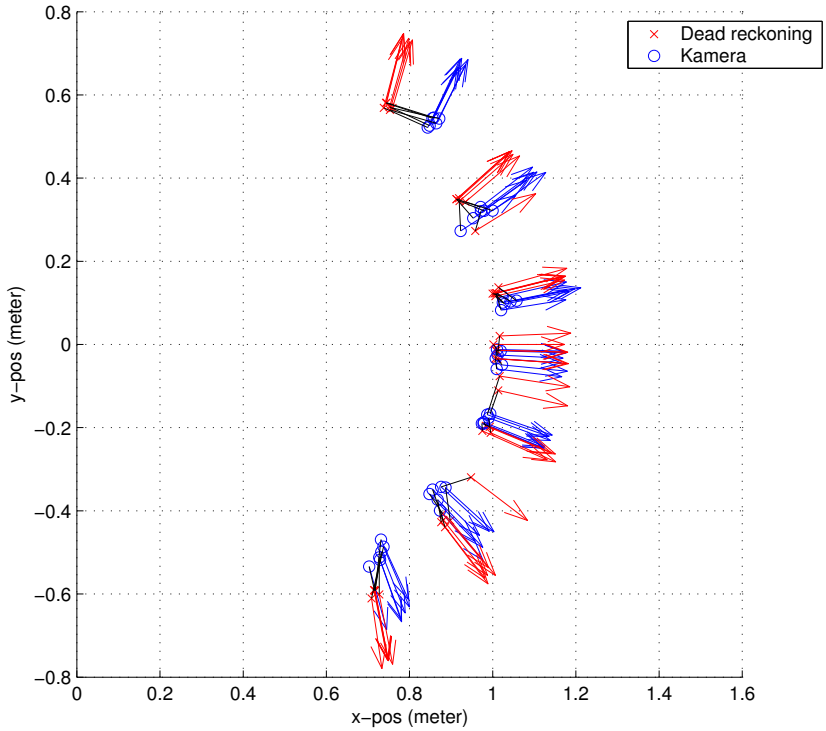
Eksperimentet undersøger én variabel parameter, nemlig hjulvinklens indflydelse på hjulenes udskriden under kørslen. Resten af parametrene fastlåses. Resultatet af dette vil blive anvendt senere i afsnit 3.3.2. Robotten sættes til at køre en (bue)længde på 1,0 meter med konstant hjulvinkel og med en konstant kørehastighed på $v = 0,5\,m/s$. Afstanden beregnes dog ud fra robottens tachometre, så den har en vis usikkerhed. Dette gælder også hastigheden, der reguleres af robottens hastighedsregulator. Der køres på underlaget i billedlaboratoriet på DIKU, som er et jævnt linoleumsgulv.

Ved hjælp af et kamera monteret i loftet måles den absolutte reelle position $\hat{p}_1$, samt orientering af robotten, når den har accelereret op til kørehastigheden. Når robotten herefter har kørt strækningen på 1,0 meter, måles slutpositionen $\hat{p}_2$. Dette gøres rent praktisk ved at optage en film af hele kørslen og identificere $\hat{p}_1$ og $\hat{p}_2$ ved at få robotten til at blinke med en lysdiode på de respektive positioner. Samtidig noteres tachometrenes værdier. Forsøget gentages for forskellige hjulvinkler $\phi = \{-15,\ -10,\ -5,\ 0,\ 5,\ 10,\ 15\}$ grader. Hjulvinklen holdes konstant under hver kørsel, og der køres seks gange for hver hjulvinkel. Resultatet ses på figur 3.2, hvorpå den ud fra tachometrenes målinger beregnede position også er indtegnet. Antallet af de valgte hjulvinkler og gentagelser er en afbalancering: jo flere gentagelser og hjulvinkler jo bedre datamateriale, men jo længere tager forsøget at udføre.

Af figur 3.2 ses det, at der er en vis systematik i de fejl, positionsberegningen ud fra tachometermålingerne giver. Det er dog svært at se ud fra figuren, hvordan fejlen afhænger af de indgående størrelser. Det ønskes derfor at se på de beregnede og de med kameraet målte længder højre hhv. venstre baghjul har kørt for herigennem at se, om systematikken kan gennemskues.

Ud fra målingerne foretaget med kameraet kan strækningerne som højre og venstre baghjul reelt har kørt, samt den reelle hjulvinkel beregnes. Dette gøres i det følgende. En del af de betegnelser, som blev brugt i forrige afsnit, bliver genbrugt herunder, men betegner nu den tilsvarende størrelse, her beregnet ud fra *kameraets* målinger. For at skelne dem fra de tilsvarende størrelser målt/beregnet ud fra tachometrene, betegnes kameraets størrelser med en hat over, fx $\hat{p}$.

Enhederne for målingerne $\hat{p}_1 = (x_1,\ y_1,\ \theta_1)$ og $\hat{p}_2 = (x_2,\ y_2,\ \theta_2)$ foretaget med kameraet, konverteres fra pixel til meter, idet kameraets opløsning er målt til $k_o = 325,69$ pixel pr. meter, og y-aksen vendes opad, da kamerakoordinatsystemet er



Figur 3.2: Den ud fra kameraets målinger beregnede position, samt positionen beregnet ud fra robottens tachometermålinger. Korresponderende positioner er forbundet med sorte linjer.

et venstrehåndskoordinatsystem:

$$P_i = \frac{1}{k_o} \begin{pmatrix} x_i \\ -y_i \end{pmatrix} \tag{3.14}$$

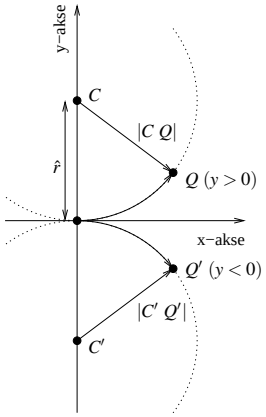
Nu konverteres punktet P_2 til et nyt højrehåndskoordinatsystem med origo i P_1 og med førsteaksen roteret θ_1 radianer:

$$Q = \mathrm{rot}(P_2 - P_1,\ -\theta_1) \tag{3.15}$$

Drejeradiussen $\hat{r}$ målt med kameraet kan findes ud fra figur 3.3 som:

$$\hat{r} = \| CQ \| = \sqrt{(Q_x - 0)^2 + (Q_y - \hat{r})^2} \quad \Rightarrow \tag{3.16}$$

$$\hat{r} = \frac{Q_x^2 + Q_y^2}{2|Q_y|} \tag{3.17}$$



Figur 3.3: Figur som skitserer udledningen af drejeradiussen $\hat{r}$ samt udledningen af vektoren $|C\ Q|$. Læg mærke til de to tilfælde $y < 0$ og $y > 0$.

Her er $Q_y = 0$ et specialtilfælde, der selvfølgelig skal tages hensyn til. Dette er let, da det svarer til, at robotten har kørt præcist ligeud. For at finde strækningerne $\hat{s}_h$ og $\hat{s}_v$ kørt med højre hhv. venstre baghjul findes ændringen i orienteringen $\hat{\theta}$, som kan findes ud fra vinklen $\mathrm{ang}(|CQ|)$ af vektoren $|CQ|$:

$$\hat{\theta} = \mathrm{sign}(Q_y)(\mathrm{ang}(Q_x,\ |Q_y| - \hat{r}) - \frac{\pi}{2}) \tag{3.18}$$

I det ovenstående tages der hensyn til en symmetri, som giver et fortegnsskift, hvilket resulterer i den indviklede formulering. $\mathrm{sign}(x)$ betegner fortegnet af x:

$$\mathrm{sign}(x) = \begin{cases} 1 & \text{hvis } x \geq 0 \\ -1 & \text{hvis } x < 0 \end{cases} \tag{3.19}$$

Radiussen for cirklen som det højre hhv. venstre baghjul følger findes som:

$$\hat{r}_h = \hat{r} + \mathrm{sign}(Q_y)\frac{W}{2} \tag{3.20}$$

$$\hat{r}_v = \hat{r} - \mathrm{sign}(Q_y)\frac{W}{2} \tag{3.21}$$

Endelig kan $\hat{s}_h$ og $\hat{s}_v$ findes som:

$$\hat{s}_h = \hat{r}_h\hat{\theta} \tag{3.22}$$

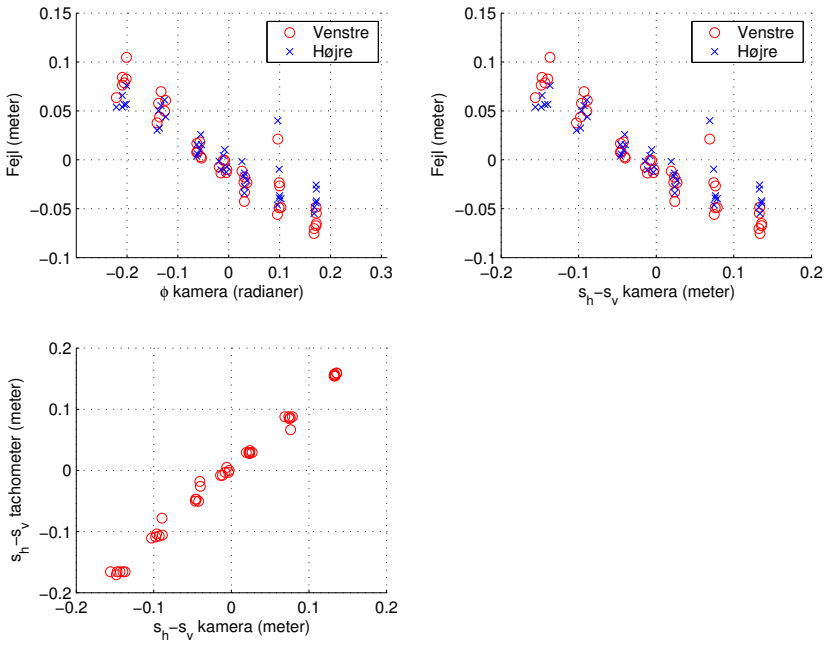
$$\hat{s}_v = \hat{r}_v\hat{\theta} \tag{3.23}$$

Ønskes den faktiske hjulvinkel $\hat{\phi}$ udregnet, kan den findes som:

$$\hat{\phi} = \arctan\left(\frac{L}{\hat{r}}\right)\mathrm{sign}(Q_y) \tag{3.24}$$

Nu haves s_h og s_v både udregnet ud fra kameraets målinger og ud fra robottens tachometre, og fejlen i tachometrenes målinger kan findes, se figur 3.4.

Som det ses af figuren afhænger fejlen $s_h - \hat{s}_h$ og $s_v - \hat{s}_v$ til dels af, hvor skarpt robotten drejer. En sådan sammenhæng kaldes en *systematisk* fejl, jf. [Borenstein et al., 1996, afsnit 5.1]. Fejlen har dog også et vist element af tilfældighed, en såkaldt *usystematisk* fejl. Den systematiske del af fejlen kan der kompenseres for analytisk, hvilket vil blive gjort i det følgende. Den usystematiske del af fejlen kan der ikke kompenseres for direkte, men det kan estimeres, hvor stor del den udgør, hvilket vil blive gjort i afsnit 3.1.4.



Figur 3.4: Øverst til venstre: Fejlen i hhv. s_h og s_v som funktion af den ud fra kameraets målinger beregnede hjulvinkel $\hat{\phi}$.
Øverst til højre: Fejlen i hhv. s_h og s_v som funktion af den ud fra kameraets målinger beregnede difference $\hat{s}_h - \hat{s}_v$.
Nederst til venstre: Differencen $s_h - s_v$ som funktion af differencen $\hat{s}_h - \hat{s}_v$.

3.1.3 Korrektion af systematisk fejl

På øverste venstre graf på figur 3.4 ses det, at der er en omtrent lineær sammenhæng mellem den ud fra kameramålingerne beregnede hjulvinkel og fejlen af den kørte længde for både højre og venstre baghjul. Med fejlen menes differencen mellem den ud fra robottens tachometre målte kørte længde og den med kameramålingerne beregnede kørte længde $s - \hat{s}$ for både højre og venstre baghjul. Faktisk behøves en hjulvinkel ikke blive beregnet. Der er nemlig også en omtrent lineær afhængighed mellem differencen $\hat{s}_h - \hat{s}_v$ ud fra kameraets målinger og fejlen, jf. øverste højre graf på figur 3.4. Kameraets målinger haves naturligvis ikke, når robotten kører autonomt, men som det ses af nederste venstre graf på figur 3.4, så er der også en omtrent lineær afhængighed mellem $\hat{s}_h - \hat{s}_v$ og $s_h - s_v$.

Fejlen på højre hhv. venstre tachometermåling kan altså til dels forudsiges ud fra differencen $s_h - s_v$ mellem dem! Tilnærmes data på hhv. øverste og nederste venstre graf på figur 3.5 til en ret linje (giver de stiplede linjer på graferne), separat for hvert datasæt, og korrigeres hvert datasæt for den lineære afhængighed, giver det resultatet i de to højre grafer på figur 3.5. Korrektionen udregnes som:

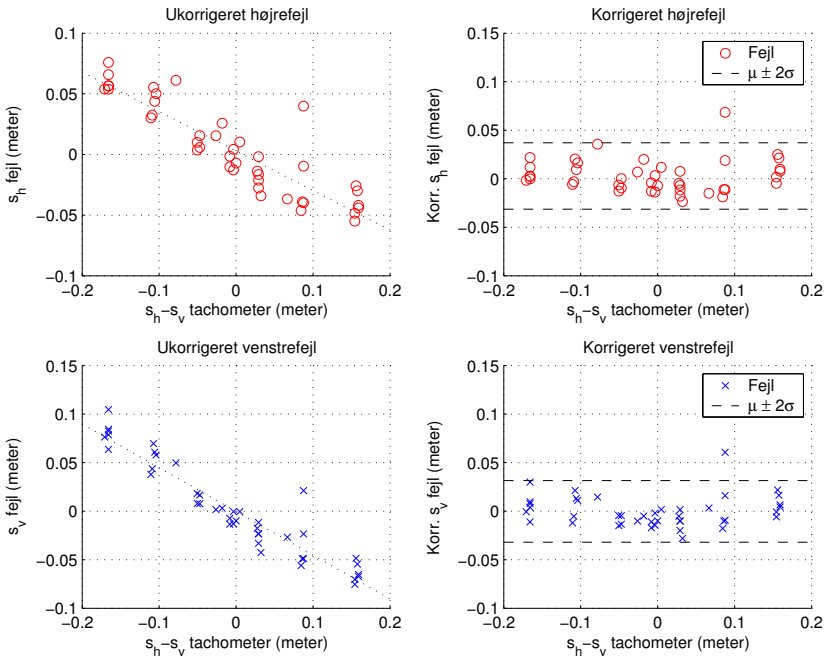
$$\tilde{s}_h = s_h - a_h(s_h - s_v) \tag{3.25}$$
$$\tilde{s}_v = s_v - a_v(s_h - s_v) \tag{3.26}$$

Hældningen af linjerne er ud fra den lineære tilnærmelse fundet til $a_h = -0,33$ og $a_v = -0,45$. Som det ses af graferne, reducerer korrektionerne fejlen betragteligt.

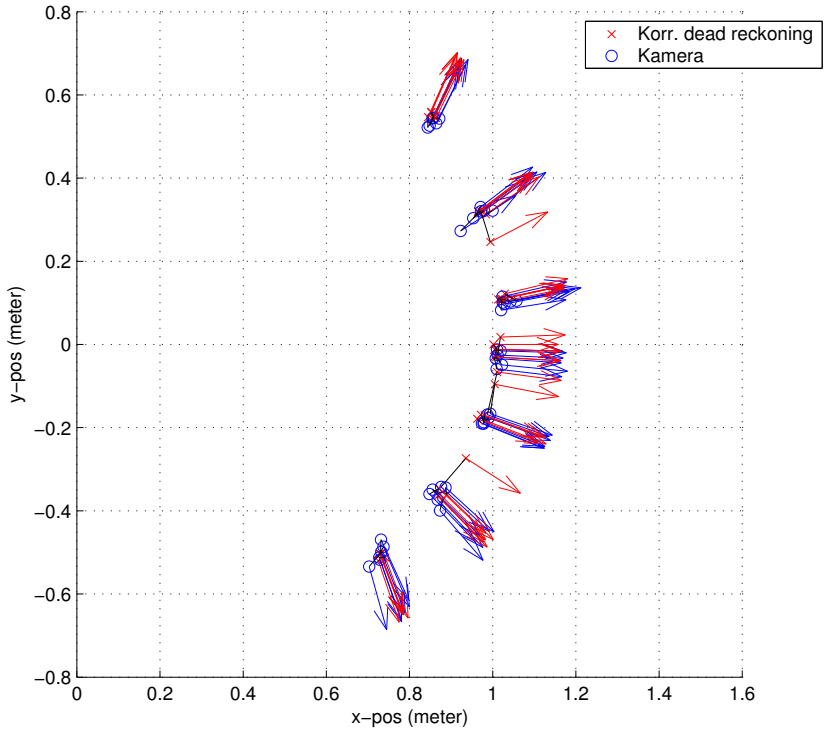
Beregnes nu de korrigerede positioner ud fra datasættene afbilledet på graferne i højre side af figur 3.5, fås resultatet som vist på figur 3.6. Sammenholdes denne med figur 3.2, ses det, at fejlen er reduceret betragteligt. Beregnes den gennemsnitlige afstand mellem kamerapositionen og tachometerpositionen fra figur 3.2, giver det en fejl på $0,0624$ meter. Beregnes den gennemsnitlige afstand ud fra figur 3.6, giver det en fejl på $0,0266$ meter, hvilket er en pæn reduktion.

Valget af en lineær approksimation blev gjort, da det virkede rimeligt ud fra graferne af måledata. Andre approksimationer kan muligvis give bedre resultater, fx en lineær approksimation med forskellig hældning for $s_h - s_v > 0$ hhv. $s_h - s_v < 0$. Dette antyder graferne i højre side af figur 3.5, da målepunkterne langt fra $s_h - s_v = 0$ generelt ligger noget over 0, og målepunkterne tæt på $s_h - s_v = 0$ ligger noget under 0. En komparativ analyse er dog her udeladt.

Som systematisk fejl havde jeg forventet, at hvis robotten fx drejede til venstre, så ville den målte værdi på venstre tachometer være lidt for stor og ligeledes ville



Figur 3.5: De to venstrestående grafer: Fejlen $s_h - \hat{s}_h$ hhv. $s_v - \hat{s}_v$ i den kørte strækning som funktion af differencen $s_h - s_v$.
De to højrestående grafer: Den korrigerede fejl $\tilde{s}_h - \hat{s}_h$ hhv. $\tilde{s}_v - \hat{s}_v$ i den kørte strækning som funktion af differencen $s_h - s_v$, samt 95 % konfidensintervaller.



Figur 3.6: Den ud fra kameraets målinger beregnede position, samt den korrigerede position beregnet ud fra datasættene afbilledet på graferne i højre side af figur 3.5. Korresponderende positioner er forbundet med sorte linjer.

værdien på højre tachometer være lidt for lille. Dette forventede jeg, idet jeg antog, at motorkraften på højre og venstre baghjul ville være lige stor, hvilket dog er lidt af en tilsnigelse. Som det ses af øverste venstre graf på figur 3.4 gav eksperimentet ikke det resultat, jeg havde forventet: drejes til venstre bliver både den højre og den venstre tachometerværdi for lille, og modsat, når der drejes til højre. Dette resultat tillægger jeg det faktum, at hvert baghjul er drevet af en separat motor, hvorved motorkraften på baghjulene næsten med garanti ikke bliver lige stor, også selvom de får samme strømspænding. To motorer af samme model har nemlig sjældent helt samme karakteristika.

3.1.4 Usystematisk fejl

Den del af fejlen, der ikke kan kompenseres for, kaldes den usystematiske fejl. I det følgende vil fejlen $\tilde{s}_h - \hat{s}_h$ hhv. $\tilde{s}_v - \hat{s}_v$ mellem de korrigerede tachometermålinger og kameramålingerne blive betragtet som usystematisk – også selvom en bedre korrektion end den viste måske findes. En usystematisk fejl kan der i sagens natur ikke korrigeres for, men dens størrelse kan estimeres. Dette vil blive gjort her, da resultatet heraf vil blive brugt senere i afsnit 3.3.2.

Tabel 3.1 opsummerer middelværdi og spredning for $\tilde{s}_h - \hat{s}_h$ og $\tilde{s}_v - \hat{s}_v$. Se også højre side af figur 3.5, hvor de er indtegnet som 95 %-konfidensintervaller.

Middelværdi	**Spredning**	**95 %-konfidens**
$\mu_h = 0,0029$ m	$\sigma_{h1} = 0,0171$	$[-31;\ 37]$ mm
$\mu_v = -0,00024$ m	$\sigma_{v1} = 0,0159$	$[-32;\ 32]$ mm
$\mu_d = 0,027$ m	$\sigma_{d1} = 0,0231$	$[-19;\ 73]$ mm
$\mu_\theta = -0,013\ (-0,84°)$	$\sigma_{\theta 1} = 0,0631$	$[-8,9°;\ 7,2°]$

Tabel 3.1: Middelværdi, spredning og 95 %-konfidensintervaller for $\tilde{s}_h - \hat{s}_h$ og $\tilde{s}_v - \hat{s}_v$, samt for $\| \tilde{p}\ \hat{p} \|$ og $\tilde{\theta} - \hat{\theta}$.

Som det ses, er middelværdierne μ_h og μ_v tæt på 0. Havde de været langt fra 0, ville det være et udtryk for en væsentlig systematisk fejl, der ville kunne kompenseres for ved at subtrahere middelværdierne fra hhv. $\tilde{s}_h$ og $\tilde{s}_v$. De fire spredninger i tabellen er indekseret med et 1-tal, for at understrege, at de betegner spredningen ved en kørt afstand på $1,0$ meter.

I tabel 3.1 ses også gennemsnit og spredning for afstanden $\| \tilde{p}\ \hat{p} \|$ af den ud fra de korrigerede længder beregnede position, samt gennemsnit og spredning for orienteringen $\tilde{\theta} - \hat{\theta}$. At de sidstnævnte spredninger er noget større end σ_{h1} og σ_{v1},

er et udtryk for den ikkelineære egenskab ved formel 3.10: En lille forskel mellem s_h og s_v giver både en stor euklidisk afstand mellem de resulterende positioner og en stor forskel mellem de resulterende orienteringer. Hvorledes dette kommer til udtryk, vil blive illustreret grafisk i afsnit 3.3.2. Størrelserne μ_d, σ_{d1}, μ_θ og $\sigma_{\theta 1}$ bør ikke forveksles med størrelserne med samme navne i afsnit 3.2.11, da de ikke er relaterede.

Varians-kovarians-matricen findes også, da den vil blive anvendt senere:

$$\Sigma_1 = 10^{-4} \cdot \begin{bmatrix} 2,92 & 2,47 \\ 2,47 & 2,53 \end{bmatrix} \tag{3.27}$$

Som det ses, er kovariansen forholdsvis stor, så der er en vis statistisk sammenhæng mellem $\tilde{s}_h - \hat{s}_h$ og $\tilde{s}_v - \hat{s}_v$.

I det ovenstående blev det antaget, at kameraets målinger er fejlfri, men det er strengt taget ikke tilfældet: Måleværdierne har en præcision på ± 1 px ($\pm 3,1$ mm) spatialt, $\pm 3,5$ px ($\pm 10,7$ mm) temporalt og $\pm 0,5^\circ$ på orienteringen. Dette vil der dog ikke blive taget yderligere hensyn til.

3.1.5 Delkonklusion og mulige forbedringer

I denne sektion er det blevet vist, hvordan dead reckoning for en bil-agtig robot kan beregnes. Det blev vist, hvordan der kan korrigeres for en systematisk fejl – formentlig den vigtigste ved kørsel med konstant fart på et jævnt gulv. Sidst blev den resterende usystematiske fejl vurderet statistisk.

Som mulige forbedringer kan her nævnes: Flere eksperimenter med kørsel fx inklusiv acceleration og kørsel på ujævnt gulv. Bedre korrektion af den systematiske fejl, måske kan en bedre approksimation end den anvendte lineære approksimation findes. Robotten kan forbedres, da de ideelle hjul til dead reckoning er hårde, tynde og har godt vejgreb, robotten Murphy har forholdsvis bløde ca. 10 mm brede hjul. Motorerne er desuden monteret på de samme hjul som tachometrene. Dette kunne forbedres ved at flytte motorerne til forhjulene, hvilket dog er indviklet rent konstruktionsmæssigt. Alternativt kunne tachometrene flyttes til en trailer hægtet bag på Murphy, men dette ville ændre robottens køreegenskaber radikalt.

3.2 Landmærker

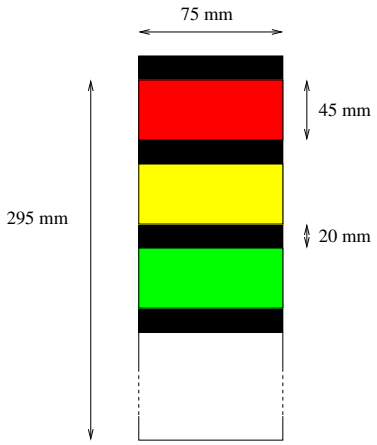
Ved projektets start blev det bestemt at anvende landmærkegenkendelse til at forbedre positionsbestemmelsen beregnet ud fra dead reckoning. Dead reckoning

har som nævnt en akkumulerende fejl, så over lange afstande bliver dead reckoning meget upræcist. At bruge landmærkegenkendelse er inspireret af et tidligere projekt, jf. [Kjeldsen et al., 2001].

For at lette landmærkegenkendelsen vælges det at anvende *kunstige* landmærker. Med kunstige landmærker menes her landmærker med et ensartet udseende, som på forhånd placeres på udvalgte steder i det område, hvor robotten skal køre. Naturlige landmærker kunne også være valgt. Med naturlige menes bestemte kendetegn, som allerede findes i det område, hvor robotten skal køre. Disse kunne inkludere døre, vinduer, hjørner og teksturer, som er genkendelige. Naturlige landmærker vil dog generelt være en del sværere at genkende, dvs. kræve en mere avanceret algoritme og formodentlig også mere processorkraft.

3.2.1 Design

Da valget er faldet på kunstigt introducerede landmærker, kan disse designes, så de er så lette at genkende som muligt. Igen inspireret af det tidligere projekt jf. [Kjeldsen et al., 2001], vælges en cylindrisk form. Alle landmærker er farvet sorte, og hvert landmærke kan unikt identificeres ud fra tre bånd i klare farver, se figur 3.7. Båndenes farver udgøres af "fluorescerende" papir. Hvert bånds farve vælges ud af de fire farvemuligheder pink, orange, gul og grøn. For en beskrivelse af den fysiske udformning af landmærkerne og mere om valget af det fluorescerende papir, se appendiks C.



Figur 3.7: Landmærke.

Landmærkernes cylindriske form med farvede bånd er valgt, da landmærkerne så ser ens ud, uafhængigt af hvorfra robotten ser landmærkerne. Dette er betinget af,

at robotten udelukkende kan bevæge sig i planen. Kun landmærkernes størrelse i billedet ændrer sig, og der skal derfor ikke tages hensyn til rotation og perspektivisk forvrængning. Afstanden til et landmærke kan derfor let beregnes, da den er omvendt proportional med landmærkets højde og bredde. I den internationale robotkonkurrence RoboCup [Federation, 2005], anvendes et lignende landmærkedesign, jf. [Committee, 2004].

3.2.2 Inddata fra kameraet

Det valgte kamera er et Philips PCVC740 “ToUCam Pro”. Dette kamera blev valgt, da jeg i forvejen har erfaring med det, se [Bidstrup et al., 2002], og fordi der findes en driver til kameraet under Linux.

Kameraet giver som uddata et billede i YUV420P-format, i en opløsning på op til 640x480 pixel ved 15 billeder pr. sekund. Da YUV420P koder intensitetsinformationen i billedet ved fuld opløsning, men kun koder farveinformationen ved den kvarte opløsning, se [Corporation, 2004, kapitel 6], nedskaleres intensitetsplanet til den kvarte opløsning, nemlig 320x240 pixel. Det er fortrinsvis farveinformationen i billedet, som er interessant på grund af landmærkernes design. Denne opløsning er samtidig omtrent den højeste, robottens indlejrede processor kan håndtere, når billedet skal realtidsprocesseres.

3.2.3 Klassificering af pixels

Landmærkerne er kendetegnet ved de stærkt farvede bånd, så hver enkelt pixel i billedet kan klassificeres som formodentlig hørende til et landmærkes farvede bånd eller ej, ud fra pixellens farve.

Et hensigtsmæssigt billedformat at genkende farver i er HSI-formatet. RGB-formatet repræsenterer den enkelte pixel ved de primære spektrale komponenter af rød, grøn og blå. HSI-formatet derimod, repræsenterer hver pixel ved dens farve (Hue), dens farvemætning (Saturation) og dens intensitet (Intensity), jf. [Gonzalez and Woods, 1993, afsnit 4.6.2].

Billedet fra kameraet i YUV420P-formatet konverteres først til RGB-formatet, og derefter til HSI-formatet, da det anvendte billedbehandlingsbibliotek IPP, jf. [Corporation, 2004], ikke understøtter direkte konvertering. For at lette den videre behandling af billedet, afskæres visse pixel-værdier nu, ud fra tærskelværdier for hhv. farve, farvemætning og intensitet. Pixels, som ikke har en farve inden for spektret fra pink til grøn, samt pixels, som ikke har en vis farvemætning og inten-

sitet, sættes til (0,0,0). Pixels i billedet, som ikke har værdien (0,0,0), stammer nu potentielt fra et landmærkes farvebånd.

3.2.4 Segmentering

Billedet segmenteres nu ved nabosammenligninger. Algoritmen virker som *connected component labeling* og er implementeret stort set som beskrevet i [Gonzalez and Woods, 1993, afsnit 2.4.3]. Naboer er defineret som værende 8-forbundne. Samtlige (0,0,0) pixels ignoreres, man kan tænke på disse som del af et baggrundssegment.

En symmetrisk sammenligningsfunktion `pixel_cmp(x, y)` benyttes til at afgøre, om to nabopixels x og y er ens eller ej. Sammenligningen foregår, som før nævnt, i HSI-farverummet, og hvis de to pixels hhv. farve, farvemætning og farveintensitet er tilstrækkelig ens, giver sammenligningen "sand", ellers giver den "falsk". Tærskelværdierne for, hvornår værdierne er tilstrækkelig ens, er fundet empirisk, og de faktisk anvendte værdier kan ses i tabel 3.2.

Kanal	**Difference**
Farve (H)	< 7
Farvemætning (S)	< 15
Intensitet (I)	< 50

Tabel 3.2: Tærskelværdier for, hvornår to pixels i HSI-farverummet betragtes som værende tilstrækkeligt ens til at blive betragtet som del af samme segment. Værdierne for hver kanal i det anvendte HSI-farverum ligger i intervallet [0; 255].

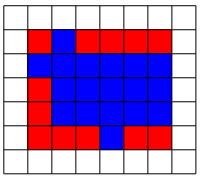
3.2.5 Segmentheuristikker

Hvert af de fundne segmenter testes nu mod en række heuristikker. Da hvert segment skal stamme fra et farvet bånd på et landmærke, kan eventuelle segmenter, som ikke ligner et farvet bånd, frasorteres. Hvert segment bør ligne et lidt horisontalt aflangt rektangel, jf. figur 3.7, og dette afprøves ved de følgende heuristikker, der dog tillader en vis variation, som kan opstå ved klassificeringen:

- Segmenter skal være større end 7 pixels, for at frasortere støj.
- Segmenters omgivne rektangel skal være bredere end det er højt.

- Bredden af segmenters omgivne rektangel skal være mindre end fire gange højden.
- Antallet af segmentets pixels skal udgøre mindst 75 % af areallet af det omgivne rektangel.

De anvendte heuristikker er fundet empirisk. Det omgivne rektangel for et segment er her defineret, så de ekstreme pixels af segmentet ligger *på* rektanglet, dvs. som illustreret på figur 3.8. Med ekstreme menes her den øverste hhv. den nederste pixel, og den pixel længst til venstre, hhv. længst til højre i billedet.



Figur 3.8: Det omgivne rektangel for et segment. Med blåt: De pixels segmentet består af. Med rødt: Det omgivne rektangel, hvor dette ikke er overlappet af segmentet.

3.2.6 Klassificering af segmenter

For at klassificere de fundne segmenter findes hvert segments gennemsnitsfarve, dvs. gennemsnittet af segmentets pixels. Gennemsnitsfarven findes i RGB-farverummet og konverteres herefter til HSI-farverummet, hvor gennemsnitsfarven nu er givet ved hue-værdien. Grunden, til at der ikke direkte tages gennemsnit af HSI-farverummet, er, at hue-værdien ligger på en farvecirkel i intervallet $[0;\ 255]$ hvor både 0 og 255 angiver (røde) farver, som ligger tæt på hinanden. Hvert segment klassificeres nu ud fra dens gennemsnitsfarve. Ligger farven inden for et givent interval, klassificeres segmentet til at have den tilsvarende farve. De segmenter, der ikke kan klassificeres som hørende til en farve, kasseres. For de anvendte intervaller og de farver, som intervallet bliver klassificeret til, se tabel 3.3.

3.2.7 Genkendelse af landmærker

For at finde ud af, hvilke segmenter der hører til samme landmærke, sammenholdes segmenterne. Derved fås et korrespondanceproblem, for hvis hvert segment skal sammenholdes med ethvert andet segment for at konstatere, om de hører til samme landmærke, giver det en køretid på $O(n^2)$, hvor n er antallet af segmenter.

Interval	Farve
[230; 255, 0; 15]	Rød
[16; 40]	Orange
[41; 60]	Gul
[65; 120]	Grøn
Resten	Uklassificeret

Tabel 3.3: Intervaller for farvekanalen i HSI-rummet samt de farver, som intervallet bliver klassificeret til.

For at reducere køretiden sorteres segmenterne derfor efter stigende x-position, hvor x-positionen er x-positionen af segmentets massecenter. Segmenter, der hører til samme landmærke, vil nu ligge tæt på hinanden i listen, og køretiden vil amortiseret være $O(n)$. Listen gennemløbes, og kun segmenter, som ligger en vis afstand længere henne i listen sammenlignes. Denne afstand regnes som afstanden mellem x-positionerne af segmenterne, og den må maksimalt være 75 % af bredden, hvilket er fundet empirisk. Matchningen foregår ved, at en række heuristikker skal være opfyldt. Heuristikkerne er fundet ud fra geometrisk betragning af landmærkernes opbygning, og kravenes rigiditet er fundet empirisk:

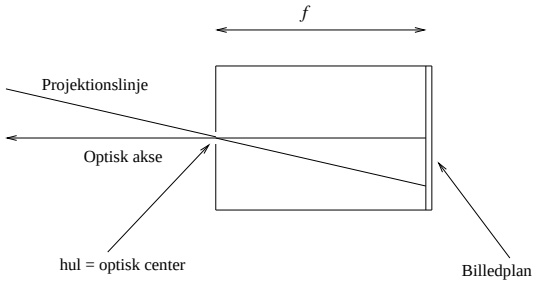
- Segmenternes omgivne rektangler skal ligge inden for en vis vertikal afstand, svarende til det sorte mellemrum mellem båndene på landmærket. Højden af mellemrummet mellem båndene på et landmærke er $\frac{2}{9}$ af summen af båndenes højder, jf. figur 3.7. Derfor skal følgende ulighed være opfyldt, for at to segmenter kan stamme fra to nabobånd på et landmærke: $|h_m - \frac{2(h_1+h_2)}{9}| \leq h_t$. Her er h_m højden af mellemrummet mellem de to segmenters omgivne rektangler, og h_1 og h_2 er rektanglernes højder. h_t er en tærskelværdi, som tillader en lille afvigelse i differencen, empirisk fundet til $h_t = 5$.
- Segmenterne skal være omtrent lige høje, det ene må ikke være mere end dobbelt så højt som det andet.
- Segmenterne skal være omtrent lige brede, det ene må ikke være mere end dobbelt så bredt som det andet.

Når den ovenstående matchning er foretaget, haves nu en mængde af grupper af segmenter. Disse grupper undersøges for at se, om de er landmærker. Landmærkerne er konstrueret, så ingen af farveringene har samme farve på samme landmærke. Empirisk har det nemlig vist sig, at i sjældne tilfælde vil støj blive til

segmenter, som vil blive matchet som et muligt landmærke. Men i disse tilfælde vil alle segmenterne som regel have samme farve. Desuden er landmærkerne kendetegnet ved at have netop tre bånd. Derfor kasseres de segmentgrupper, hvor flere segmenter har samme farve, samt de segmentgrupper, som ikke indeholder netop tre segmenter, da ingen af disse kan stamme fra landmærker.

3.2.8 Beregning af brændvidde

I det følgende vil det anvendte kameras brændvidde f blive anvendt. Da brændvidden for kameraet ikke er kendt, må den måles. Kameraet modelleres som et *hulkamera* (eng: pin-hole camera), se figur 3.9.



Figur 3.9: Hulkamera. Efter [Olsen, 1995, p. 5].

For at måle brændvidden (uden at skille kameraet ad) tages et billede af et objekt med kendt længde l på en kendt afstand d. Objektets længde i billedet måles og betegnes L, og nu kan brændvidden f udregnes ved trekantsberegning som:

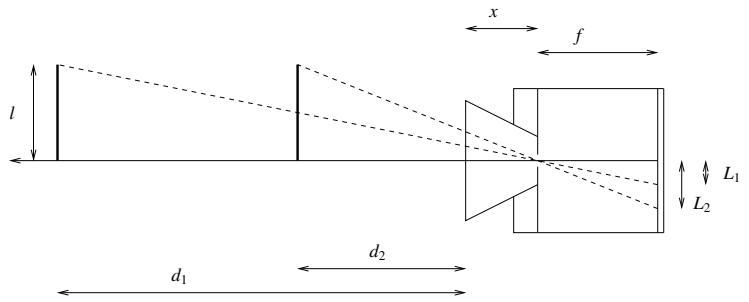
$$f = L\frac{d}{l} \tag{3.28}$$

Desværre kendes den nøjagtige position x langs den optiske akse af det optiske center heller ikke, da den ikke kan ses uden på kameraet, men den kan beregnes, se figur 3.10.

Brændvidden afhænger af x:

$$f = L\frac{d+x}{l} \tag{3.29}$$

For at beregne både f og x tages to billeder af et kendt objekt ved to forskellige afstande, og nu kan to ligninger med to ubekendte løses:



Figur 3.10: Beregning af brændvidden f og positionen af optisk center x.

$$f = L_1 \frac{d_1 + x}{l}, \tag{3.30}$$

$$f = L_2 \frac{d_2 + x}{l} \quad \Rightarrow \tag{3.31}$$

$$L_1 \frac{d_1 + x}{l} = L_2 \frac{d_2 + x}{l} \quad \Rightarrow \tag{3.32}$$

$$x = \frac{L_2 d_2 - L_1 d_1}{L_1 - L_2} \tag{3.33}$$

Ud fra ovenstående metode er brændvidden og positionen for det anvendte kamera beregnet til:

$$f = 518 \text{ pixel} \tag{3.34}$$

$$x = 11,3 \text{ mm} \tag{3.35}$$

3.2.9 Beregning af afstand til landmærke

Afstanden til et landmærke beregnes ved finde landmærkets omgivne rektangel, hvilket findes som det omgivne rektangel af de omgivne rektangler af hvert segment i landmærket. Med landmærkets omgivne rektangel menes her altså det omgivne rektangel af landmærkets farvede bånd. Afstanden fra kameraets optiske center til landmærket er omvendt proportionalt både med det omgivne rektangels højde og bredde. Afstanden beregnes kun ud fra højden af to grunde: For det første vil højden altid være større en bredden af det omgivne rekangel. Dette giver en

større præcision, da kvantificeringsfejlen bliver mindre, eftersom der er flere pixels. For det andet er bredden af det enkelte segment ofte usikker. Dette skyldes, at papiret, som udgør farverne på landmærkerne, ikke udviser en perfekt diffus refleksion af lyset, jf. [Foley et al., 1996, p. 723], hvilket ville have været ideelt. I denne situation ville lysintensiteten af hele arealet af hvert segment være den samme, hvis det reflekterede lys ligeledes er diffust. Da man ikke kan gå ud fra, at lyset er perfekt diffust, og da båndenes farve kun giver en tilnærmelsesvis perfekt diffus reflektion, vil yderste højre eller venstre kant af et landmærkes segmenter meget ofte være enten væsentlig mørkere eller lysere end hovedparten af det enkelte segments areal. Denne yderste kant vil derfor ofte ikke blive klassificeret som en del af segmentet, hvilket gør bredden usikker.

Højden af landmærkets omgivne rektangel kan nu bruges til vha. formel 3.29 at beregne afstanden d_l til landmærket som:

$$d_l = \frac{f h_l}{y_2 - y_1 + 2} - x + \frac{D_l}{2} \tag{3.36}$$

Her angiver h_l højden af landmærkets omgivne rektangel. h_l beregnes ud fra højden af de tre bånd på landmærket, samt højden af de to mellemrum, dvs. $h_l = 45 + 20 + 45 + 20 + 45 = 175$ mm, jf. figur 3.7. D_l angiver diameteren af landmærket. Den halve diameter, dvs. radiussen adderes for at finde afstanden til centrum af landmærket. Ellers ville det være afstanden til yderkanten af landmærket, der blev beregnet. Grunden til, at der adderes 2 til differencen $y_2 - y_1$ i formel 3.36 er følgende: For det første skal der adderes 1 til differencen af det omgivne rektangels y-koordinater for at finde dets højde, se figur 3.8. For det andet vil billedet af den øverste kant af det øverste bånd på landmærket og den nederste kant af det nederste bånd på landmærket give pixels med værdier, som er en blanding af bånd og mellemrum. (Dette sker for alle segmenter, men her er vi kun interesseret i at betragte højden af hele landmærket.) Gennemsnitlig tilhører halvdelen af disse blandingspixels landmærkets bånd og den anden halvdel tilhører noget, der ikke er bånd. Men ved segmentering jf. afsnit 3.2.4 skal pixellernes værdi ligge meget tæt på hinanden for at blive klassificeret som tilhørende samme segment. Derfor vil disse blandingspixels kun sjældent blive klassificeret som en del af det segment, som udgør båndet. Empirisk har det vist sig, at denne skævhed kan udjævnes væsentlig ved at addere 1 til differencen, der udregner det omgivne rektangels højde.

3.2.10 Beregning af vinkel til landmærke

Vinklen mellem kameraets optiske akse og et landmærke beregnes som følgende: For at finde vinklen kigges på landmærkets omgivne rektangel. Her findes afstanden L fra midten af billedet til midten af det omgivne rektangel:

$$\begin{aligned} L &= \frac{w_i}{2} - \frac{x_2 + x_1}{2} \quad \Rightarrow \\ L &= \frac{w_i - x_2 + x_1}{2} \end{aligned} \tag{3.37}$$

Vinklen θ til landmærket kan nu findes ud fra trekantsberegning som:

$$\theta = \arctan\left(\frac{L}{f}\right) \tag{3.38}$$

Den lidt upræcise bredde af landmærket som omtalt i afsnit 3.2.9 er ikke nær så betydningsfuld her, da nogle få pixels fejl i bredden kun vil give en lille fejl i vinklen.

3.2.11 Afprøvning

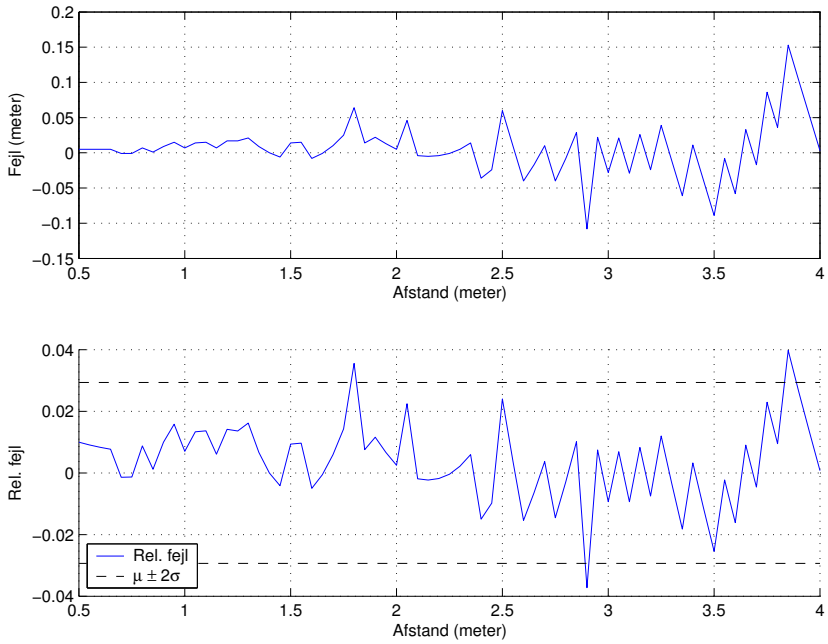
I dette afsnit vil der blive foretaget en afprøvning af landmærkegenkendelsen. Det, der vil blive afprøvet, er præcisionen af den fundne afstand og vinkel til et identificeret landmærke. Denne kan nemlig objektivt afprøves empirisk, modsat selve genkendelsen af landmærket. Genkendelsen af landmærket afhænger af mange forhold, som er svære at kontrollere præcist, nemlig forhold omkring belysningen af landmærket, kameraets automatiske kalibrering af belysningens farvetemperatur, motion blur, samt at baggrunden er tilstrækkelig forskellig fra landmærket. Dog er det let at afprøve, på hvor lang afstand et landmærke kan genkendes. Et landmærke kan genkendes i intervallet $[0,5;4,0]$ meter. På kortere afstand kan hele landmærket ikke være i synsfeltet, og på længere afstand er kameraets opløsning for lille, til at det kan skelne de farvede bånd fra hinanden.

Nu afprøves først præcisionen af den fundne afstand til landmærket. Den reelle afstand $\hat{d}$ til et landmærke måles med et målebånd og sammenholdes med den beregnede afstand d. Dette gøres for en række afstande i intervallet $[0,5;4,0]$ meter. Dette er som nævnt det afstandsinterval, inden for hvilket et landmærke

kan genkendes med succes. En sådan opmåling er foretaget, se figur 3.11 øverst. Som det ses af figuren, stiger fejlen omtrent lineært med afstanden til landmærket. Det giver derfor mening at beregne den relative fejl r_d:

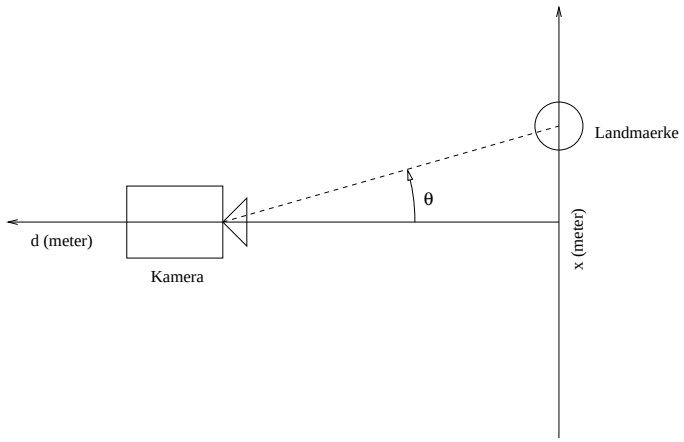
$$r_d = \frac{d - \hat{d}}{\hat{d}} \tag{3.39}$$

Den relative fejls standardafvigelse kan nu beregnes, og den er beregnet til $\sigma_{d1} = 0,013$. σ_{d1} er standardafvigelsen ved en afstand på $1,0$ meter. Ved en større eller mindre afstand er σ_d tilsvarende større hhv. mindre. Gennemsnitsfejlen er fundet til $\mu_d = 3,8$ mm. Samlet giver dette et 95 %konfidensinterval på $[-22;\ 30]$ mm, se figur 3.11 nederst.



Figur 3.11: Øverst: Fejl i den beregnede afstand som funktion af den reelle afstand til et landmærke. Med fejl menes differencen $d - \hat{d}$ mellem den med kameraet beregnede og den målte afstand.
Nederst: Den relative fejl $\frac{d-\hat{d}}{\hat{d}}$ som funktion af den reelle afstand til et landmærke, inkl. 95 %-konfidensinterval.

Dernæst afprøves præcisionen af den fundne vinkel til landmærket. Med vinkel menes her vinklen mellem kameraets optiske akse, jf. 3.9, og en ret linje fra kameraets optiske center til centrum af landmærket, se figur 3.12.



Figur 3.12: Skitse af forsøgsopstilling til afprøvning af beregningen af vinklen θ til et landmærke.

Kameraet placeres i en fast afstand d fra en linje vinkelret på kameraets optiske akse. Et landmærke sættes på en række placeringer i et interval vinkelret på denne linje. Intervallet er tilpasset, så hele landmærket er synligt fra kameraet for alle placeringer i intervallet, jf. figur 3.12. Dette gentages for forskellige værdier af d. Resultatet heraf ses på figur 3.13.

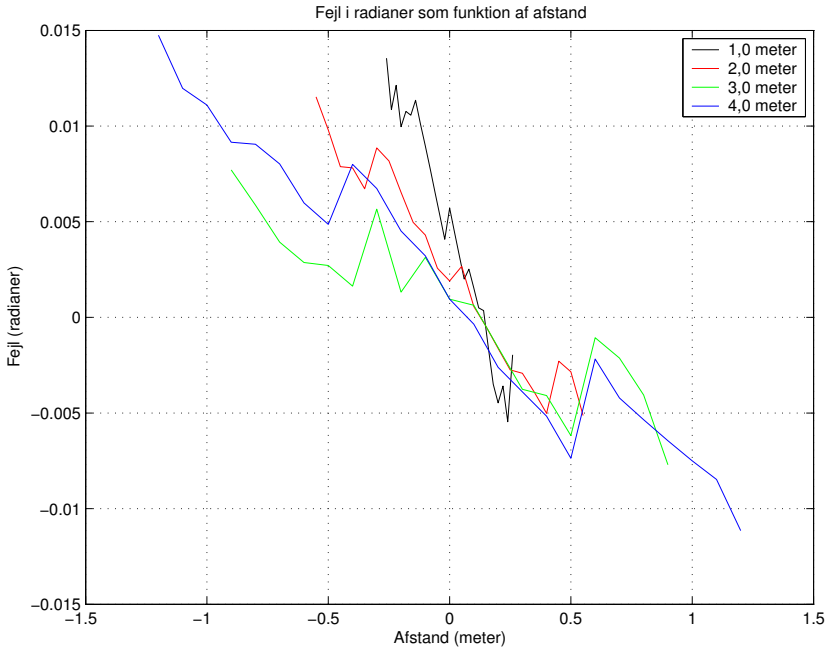
I forsøgsopstillingen blev vinklen θ ikke målt direkte, da det er forholdsvist svært at måle en sådan vinkel præcist. Da det rent praktisk er lettere at måle en afstand præcist, blev θ beregnet ud fra de med et målebånd målte afstande d og x:

$$\theta = \arctan\left(\frac{x}{d}\right) \tag{3.40}$$

Som det ses af figur 3.13 er der en tilnærmelsesvis lineær sammenhæng mellem placeringen x og vinkelfejlen. Fejlen afhænger også en smule af afstanden d.

Vinkelfejlen holder sig dog inden for $\pm 1^\circ$ for samtlige målinger. Dette vurderes at være så præcist, at der ikke er behov for at korrigere for vinkelfejlen, da det næpp kan retfærdiggøres med den valgte forsøgsopstillings generelle præcision. Middelværdi og spredning for vinkelfejlen er beregnet til $\mu_\theta = 0,0023$ og $\sigma_\theta = 0,0060$. Dette giver et 95 %konfidensinterval på $[-0,6^\circ;\ 0,9^\circ]$.

I det ovenstående blev det forudsat, at de med målebånd opmålte afstande er helt fejlfrie. Det er de naturligvis ikke i praksis. Der vil ikke her blive taget yderligere hensyn til dette. Det kan dog formodentlig forklare den forholdsvis store μ_d: Målebåndet er måske ikke lagt præcis midt i det optiske center for kameraet.



Figur 3.13: Afprøvning af beregningen af vinklen θ til et landmærke. Fejlen i radianer som funktion af længden x langs den vinkelrette linje. Med fejlen menes differencen $\theta - \hat{\theta}$ mellem den vha. kameraet beregnede vinkel og den målte vinkel.

3.2.12 Delkonklusion og mulige forbedringer

I denne sektion er det blevet vist, hvordan kunstige landmærker kan konstrueres, og hvordan afstand og vinkel til disse kan beregnes vha. et kamera og det udviklede programmel. Sidst blev kvaliteten af de beregnede størrelser vurderet statistisk. Genkendelsesprocenten blev dog ikke vurderet pga. problemet med at kontrollere lysforholdene og med at definere, hvad der er rimelige lysforhold. Dog noteres det, at under perfekte lysforhold vil genkendelsesprocenten ligge tæt på 100 %.

Som mulige forbedringer kan her nævnes: Et mere vidvinklet kamera, hvilket vil gøre det muligt at se flere landmærker ad gangen. Større kameraopløsning, hvilket vil føre til, at landmærker kan genkendes på større afstand end den nuværende maksimale afstand på ca. 4 meter, og som samtidig vil give større præcision på afstandsmålingen. Dog vil dette kræve mere processorkraft end den, som er tilgængelig på robotten i dens nuværende konfiguration. Sidst vil en mindre følsomhed over for lysforhold være ønskelig. Dette afhænger både af det udviklede program og af det valgte kamera.

3.3 Monte Carlo Lokalisering

Hovedideen i dette afsnit er, at repræsentationen af en robots position ikke skal betragtes som en enkelt endegyldig position. Den skal derimod betragtes som en sandsynlighedsfordeling over hele konfigurationsrummet, eller tilstandsrummet, som litteraturen inden for området kalder det, hvorfor det også vil blive kaldt tilstandsrummet i dette afsnit. I Murphys tilfælde vil det sige over planet inkl. orientering, dvs. $\mathbb{R}^2 \times [0;\ 2\pi[$. Ligeledes skal usikkerheden på robottens sensorer tages i betragtning, når deres målinger anvendes. Når robotten udfører en bevægelse, vil resultatet ikke blive præcis det samme hver gang. Denne usikkerhed skal også tages i betragtning.

Først vil teorien blive udledt under antagelsen, at sandsynlighedsfordelingen er kontinuert i afsnit 3.3.1. Afsnit 3.3.2 og afsnit 3.3.3 udleder modeller for, hvordan robotten bevæger sig, samt for de observationer robotten foretager med dens sensorer. Bevægelsesmodellen vil tage hensyn til usikkerheder i beregning af robottens bevægelse, og observationsmodellen vil tage hensyn til usikkerheder i observationerne. Til sidst vil afsnit 3.3.4 beskrive, hvordan man kan bruge de fundne resultater til at estimere robottens position.

3.3.1 Statistisk lokalisering

Teorien i dette afsnit tager udgangspunkt i artiklerne [Thrun et al., 2000] og [Fox et al., 1999], og kaldes ofte "(recursive) bayes filtering", jf. [Thrun et al., 2000], [Arulampalam et al., 2002] og [Carpenter et al., 1997]. Det noteres, at i denne sektion anvendes bogstavet p som betegnelse for sandsynlighedsfordelinger, og ikke for positioner, i overensstemmelse med den anvendte litteratur. x bruges som betegnelse for tilstande, som specielt i Murphys tilfælde svarer til positioner.

Formålet er at estimere en sandsynlighedsfordeling over tilstandsrummet betinget af data. Denne a posteriori-fordeling kaldes typisk tiltroen (eng: belief), idet den beskriver robottens tiltro til, at den er i en given tilstand x_t. Den skrives:

$$Bel(x_t) = p(x_t \mid d_{0\ldots t}) \tag{3.41}$$

Her betegner x tilstanden, x_t er tilstanden til tiden t, og $d_{0\ldots t}$ betegner data startende fra tiden 0 op til t. For mobile robotter skelnes mellem to typer af data. Perceptuelle data og bevægelsesdata. Perceptuelle data betegnes med o (for observation) og bevægelsesdata med a (for aktion). Med perceptuelle data menes de observationer, robotten gør af verden, altså hvordan robotten opfatter eller percepterer

verden. Med bevægelsesdata menes, hvordan robotten påvirker verden – og specielt sig selv.

Heraf får vi:

$$Bel(x_t) = p(x_t \mid o_t, a_{t-1}, o_{t-1}, a_{t-2}, \ldots, o_0) \tag{3.42}$$

Her betegner a_{t-1} robottens bevægelse i tidsintervallet $[t-1;\ t]$.

Det ønskes nu at definere $Bel(x_t)$ rekursivt, således at haves en tidligere tilstand $Bel(x_{t-1})$, samt nye data a_{t-1} og o_t, så kan den nye tilstand $Bel(x_t)$ beregnes på grundlag af disse. Dvs. så $Bel(x_t) = f(Bel(x_{t-1}), a_{t-1}, o_t)$. Det er funktionen f, der implicit bliver udledt i det følgende.

For at udlede en rekursiv opdateringsligning observeres det, at udtryk 3.42 kan transformeres vha. Bayes' regel for tæthedsfunktioner:

$$f(x \mid y) = \frac{f(y \mid x) f(x)}{f(y)} \tag{3.43}$$

Dette giver:

$$Bel(x_t) = \frac{p(o_t \mid x_t, a_{t-1}, \ldots, o_0) p(x_t \mid a_{t-1}, \ldots, o_0)}{p(o_t \mid a_{t-1}, \ldots, o_0)} \tag{3.44}$$

Fordi nævneren er en konstant i forhold til variablen x_t, kan dette omskrives til:

$$Bel(x_t) = \eta p(o_t \mid x_t, a_{t-1}, \ldots, o_0) p(x_t \mid a_{t-1}, \ldots, o_0) \tag{3.45}$$

Her er η en normaliseringskonstant, altså uafhængig af x_t, givet ved:

$$\eta = p(o_t \mid a_{t-1}, \ldots, o_0)^{-1} \tag{3.46}$$

Nu gøres den antagelse, at fremtidige data er uafhængige af tidligere data, givet den nuværende tilstand. Eller sagt på en anden måde: Idet det antages, at verden ud over robotten er statisk, og at robotten ikke ændrer på verden, så vil fremtidige observationer være uafhængige af, hvordan den nuværende tilstand er opnået. Denne antagelse kaldes Markov-antagelsen:

$$p(o_t \mid x_t, a_{t-1}, \ldots, o_0) = p(o_t \mid x_t) \tag{3.47}$$

Ud fra Markov-antagelsen kan formel 3.45 nu skrives som:

$$Bel(x_t) = \eta p(o_t \mid x_t) p(x_t \mid a_{t-1}, \ldots, o_0) \tag{3.48}$$

Leddet længst til højre kan nu udvides ved brug af sætningen om total sandsynlighed, jf. [Gut, 2005, p. 80]:

$$f(y) = \int f(y \mid x) f(x) dx \tag{3.49}$$

Dette giver:

$$Bel(x_t) = \eta p(o_t \mid x_t) \int p(x_t \mid x_{t-1}, a_{t-1}, \ldots, o_0) p(x_{t-1} \mid a_{t-1}, \ldots, o_0) dx_{t-1} \tag{3.50}$$

Igen kan Markov-antagelsen bruges:

$$p(x_t \mid x_{t-1}, a_{t-1}, \ldots, o_0) = p(x_t \mid x_{t-1}, a_{t-1}) \tag{3.51}$$

Udtrykket kan så simplificeres til:

$$Bel(x_t) = \eta p(o_t \mid x_t) \int p(x_t \mid x_{t-1}, a_{t-1}) p(x_{t-1} \mid a_{t-1}, \ldots, o_0) dx_{t-1} \tag{3.52}$$

Til sidst kan definitionen af $Bel(x_{t-1})$ indsættes, jf. formel 3.42, og følgende rekursive ligning findes:

$$Bel(x_t) = \eta p(o_t \mid x_t) \int p(x_t \mid x_{t-1}, a_{t-1}) Bel(x_{t-1}) dx_{t-1} \tag{3.53}$$

Formel 3.53 er den ønskede ligning, som beskriver, hvordan tilstanden $Bel(x_t)$ kan findes ud fra $Bel(x_{t-1})$, når der kommer nye bevægelses- og observationsdata a_{t-1} og o_t.

Formlen giver anledning til en række spørgsmål, nemlig: Hvordan defineres starttilstanden $Bel(x_0)$, da formlen jo er rekursiv? Hvordan evalueres $p(x_t \mid x_{t-1}, a_{t-1})$ og $p(o_t \mid x_t)$? Endelig er der spørgsmålet om, hvordan tilstanden $Bel(x_t)$ repræsenteres og tæt knyttet hertil, hvordan formel 3.53 løses.

Starttilstanden $Bel(x_0)$ er et udtryk for, hvad robotten på forhånd ved om sin position, når positionsestimeringen starter. Ved den intet, må $Bel(x_0)$ være en ligefordeling over hele tilstandsrummet. Dette er det såkaldte "wake-up robot problem", som iøvrigt er et specialtilfælde af "the kidnapped robot problem", jf. [Fox et al., 1999]. Haves et kort over det område, robotten bevæger sig inden for, kan umulige positioner udelukkes. Dette kan fx være placeringen af vægge og andre forhindringer, hvor robotten kan vides ikke at kunne befinde sig. Placeres robotten derimod på en udmålt startposition, og gives denne position til robotten, kan $Bel(x_0)$ fx modelleres som en normalfordeling med parametrene μ og σ svarende til præcisionen af udmålingen af startpositionen.

Tæthedsfunktionerne $p(x_t \mid x_{t-1},\, a_{t-1})$ og $p(o_t \mid x_t)$ vil blive kaldt robottens bevægelsesmodel hhv. observationsmodel, og disse vil blive gennemgået i afsnit 3.3.2 og afsnit 3.3.3 herunder. Til sidst vil afsnit 3.3.4 vise, hvordan $Bel(x_t)$ kan repræsenteres effektivt, og hvordan formel 3.53 løses.

3.3.2 Bevægelsesmodel

I dette afsnit skal bevægelsesmodellen for Murphy opstilles. a_{t-1} er den bevægelse, der sker fra tidspunktet $t-1$ til t, og det er følgende sandsynlighedsfordeling, der skal udledes på grundlag af modellen:

$$p(x_t \mid x_{t-1}, a_{t-1}) \tag{3.54}$$

Sandsynlighedsfordelingen i formel 3.54 angiver sandsynligheden for, at robotten har positionen x_t, efter at være startet i positionen x_{t-1} og have udført bevægelsen a_{t-1}.

Faktisk vil det senere vise sig, jf. afsnit 3.3.4, at der ikke er brug for et udtryk for selve sandsynlighedsfordelingen. Der vil kun blive brug for at kunne generere udfald fra den, dvs. kunne generere en mængde positioner x_t^i, så $p(x_t \mid x_{t-1}, a_{t-1})$ er fordelingen af x_t^i.

Der er mindst to måder at definere bevægelsen a på, jf. [Thrun et al., 2000, p. 105].

Den første metode går ud på, at man ud fra de kontrolkommandoer, robotten har sendt til sine aktuatorer i tidsrummet $[t-1;\, t]$, kan beregne positionsændringen. På Murphy-robotten vil en kontrolkommando være en motorspænding eller en forhjulsvinkel. Denne metode er uhensigtsmæssigt på Murphy-robotten, fordi den nøjagtige motorspænding ikke kendes, og fordi forhjulsvinklen er noget usikker pga. slør. Derudover vil kinematikken være besværlig at beregne, fordi en motorspænding skal omsættes til en acceleration, karakteristikken for servomotoren, der styrer forhjulene, skal kendes og modelleres, etc.

Den anden metode går ud på, at man bruger odometridata, dvs. uddata fra robottens tachometre, til at estimere robottens bevægelse. Dette var, som tidligere nævnt, den anvendte metode i [Rode and Kjeldsen, 2004].

I afsnit 3.1 blev sammenhængen mellem tachometermålingerne og robottens bevægelse, dvs. positionsændring, grundigt analyseret. Herudfra vil en bevægelsesmodel i det følgende blive opstillet.

Der er mange parametre, der spiller ind, når man skal opstille en bevægelsesmodel. Modellen vil blive opstillet med samme parametre som afprøvningen i afsnit 3.1.2. Dvs. det antages, at hastigheden og underlaget osv. er det samme som under afprøvningen.

Hvis tachometrene måler længderne s_h og s_v i tidsrummet fra $t-1$ til t, dvs. $a_{t-1} = \{s_h, s_v\}$, så vil robottens bevægelse blive modelleret som følgende:

$$x_t = x_{t-1} + \texttt{deadreckoning}(s_h + w_h,\ s_v + w_v) \tag{3.55}$$

Her svarer `deadreckoning`()-funktionen til formel 3.10. Som s_h og s_v skal her naturligvis anvendes de korrigerede længder $\tilde{s}_h$ og $\tilde{s}_v$, men tilderne er her udeladt for overskuelighedens skyld. w_h og w_v er usikkerheden eller støjen forbundet med bevægelsen. w_h og w_v vil blive betragtet som stokastiske udfald af en todimensionel normalfordeling. At valget er faldet på en normalfordeling, er en tilnærmelse, som virker rimelig jf. afsnit 3.1.2. Det vil ikke her blive testet, om afprøvningens resultater ligner andre forbedringer bedre.

w_h og w_v afhænger begge af den kørte længde. Jo længere robotten kørte i tidsrummet $t-1$ til t, jo større er usikkerheden. I afsnit 3.1.2 fandtes både gennemsnit og spredning for usikkerheden, når robotten havde kørt $1,0$ meter. Desuden blev det konstateret, at usikkerheden var proportional med den kørte længde. Skal spredningen σ findes for en vilkårlig kørt længde ud fra spredningen σ_1 for en kørt længde på $1,0$ meter, anvendes det at varianser er lineære:

$$\begin{aligned} \sigma^2 &= s\sigma_1^2 \quad \Rightarrow \\ \sigma &= \sqrt{s}\sigma_1 \end{aligned} \tag{3.56}$$

w_h og w_v kan altså findes som udfaldet af følgende todimensionelle normalfordeling:

$$W \sim N_2(\mu, \sqrt{s}\Sigma_1) \tag{3.57}$$

Her er $W = [w_h, w_v]$, s den kørte strækning, jf. formel 3.11, $\mu = [\mu_h, \mu_v]$ og Σ_1 er en varians-kovarians matrix. Både μ_h, μ_v og Σ_1 blev fundet i afsnit 3.1.2.

For at finde en mængde udfald $x_t^i \sim p(x_t|x_{t-1}, a_{t-1})$ findes altså først en mængde udfald $W^i \sim N_2(\mu, \sqrt{s}\Sigma_1)$, og derefter bruges formel 3.55 til at finde x_t^i ud fra x_{t-1}, $a_{t-1} = \{s_h, s_v\}$ og W^i.

Hvordan genereres $W \sim N_2(\mu, \sqrt{s}\Sigma_1)$ i praksis? I API'et for det anvendte programmeringssprog "C" findes kun en funktion til at generere (pseudo-tilfældige) udfald af en ligefordeling $U = U(a,b)$. Ved Box-Muller-metoden, jf. [Gentle, 2003, pp. 171–173], kan to uafhængige udfald af standardnormalfordelingen $X_1 \sim N(0,1)$ og $X_2 \sim N(0,1)$ findes ud fra to uafhængige udfald $U_1 \sim U(0,1)$ og $U_2 \sim U(0,1)$ af en ligefordeling som:

$$\begin{aligned} X_1 &= \sqrt{-2\log U_1}\cos(2\pi U_2) \\ X_2 &= \sqrt{-2\log U_1}\sin(2\pi U_2) \end{aligned} \tag{3.58}$$

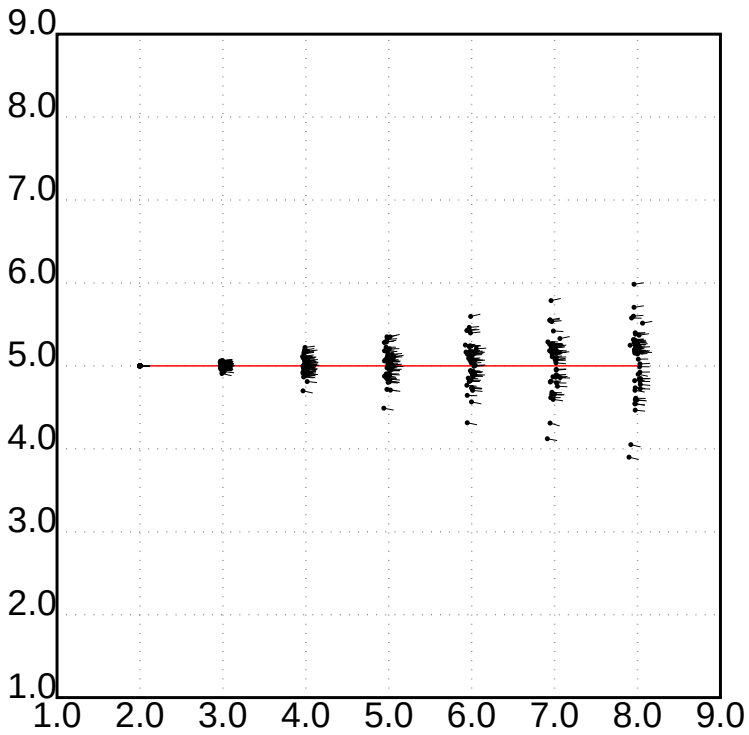
Sæt $X = [X_1, X_2]$. W kan nu jf. [Gentle, 2003, p. 197] findes som:

$$W = \sqrt{s}T^T X + \mu \tag{3.59}$$

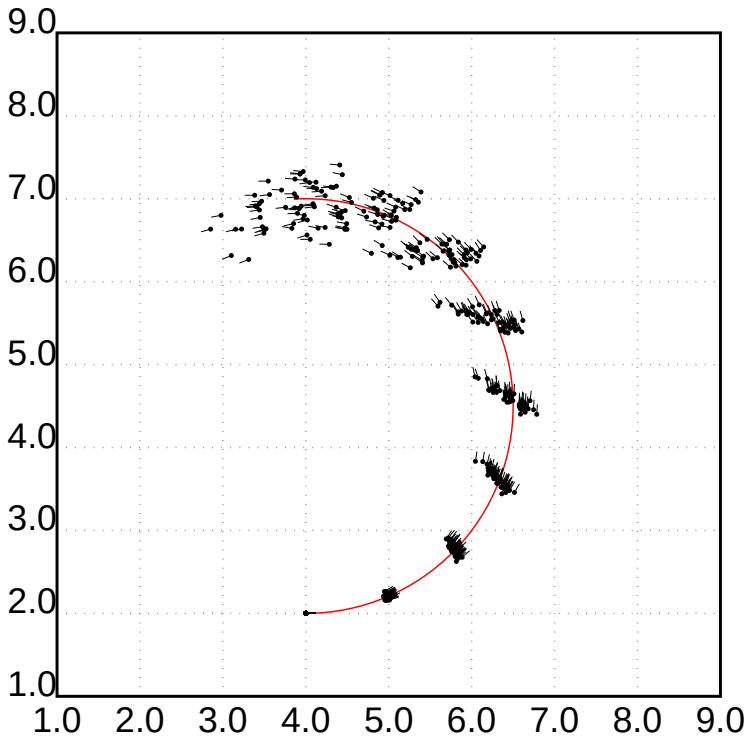
I det ovenstående skal T findes, så $T^T T = \Sigma_1$. T kan fx findes ved Cholesky-faktorisering. En sådan funktion findes i Matlab. For Σ_1 som fundet i formel 3.27 giver dette:

$$T = \begin{bmatrix} 0,0171 & 0,0145 \\ 0,0000 & 0,0066 \end{bmatrix} \tag{3.60}$$

For et eksempel på anvendelse af bevægelsesmodellen, se figur 3.14 og 3.15.



Figur 3.14: Eksempel på anvendelse af bevægelsesmodellen ved kørsel ligeud. Startpositionen initialiseres til $x_0 = (2,\ 5,\ 0)$. Formel 3.57 og formel 3.55 bruges til at opdatere positionen, som er vist for hver meter, indtil 6 meter er kørt. Dette er gentaget 50 gange. Spredningen er givet ved Σ_1.



Figur 3.15: Eksempel på kørsel langs cirkelbue. Startpositionen initialiseres til $x_0 = (4,\ 2,\ 0)$. Den estimerede position er vist for hver kørt meter.

3.3.3 Observationsmodel

I dette afsnit skal observationsmodellen opstilles. Den skal modellere robottens sensorer, i Murphys tilfælde uddata fra landmærkegenkendelsen. Idet $o = \{d,\ \theta\}$ betegner en observation, og x betegner robottens tilstand, dvs. position for Murphy, er det følgende tæthedsfunktion, der skal udledes:

$$p(o_t \mid x_t) \tag{3.61}$$

Denne giver sandsynligheden for observationen o_t, når robotten har tilstanden x_t. Da observationerne kan antages at være uafhængige af tiden, kan tæthedsfunktionen blot skrives som $p(o \mid x)$.

Som model for robottens genkendelse af landmærkerne, anvendes følgende model, som [Dissanayake et al., 2001, p. 236] også anvender:

$$\begin{aligned} d &= \sqrt{(l_x - p_x)^2 + (l_y - p_y)^2} + w_d \\ \theta &= \texttt{atan2}(l_y - p_y,\ l_x - p_x) - p_\theta + w_\theta \end{aligned} \tag{3.62}$$

Her er $x = (p_x,\ p_y,\ p_\theta)$ robottens position, $l = (l_x,\ l_y)$ er landmærkets koordinater, og w_d og w_θ er støjen for afstanden d, henholdsvis vinklen θ, hvor w_d er afhængig af afstanden d. Modellen er baseret på resultatet af afprøvningen af landmærkegenkendelsen, jf. afsnit 3.2.11. I modellen i formel 3.62 bruges funktionen $\texttt{atan2}()$. Det er en "C"-funktion, som svarer helt til brugen af $\arctan()$ i det nedenstående, men uden problemet med division med 0 ved $l_x - p_x = 0$:

$$\theta = \arctan\left(\frac{l_y - p_y}{l_x - p_x}\right) - p_\theta + w_\theta \tag{3.63}$$

Antag nu, at robotten på positionen x har gjort observationen $o = \{d,\ \theta\}$, jf. modellen i formel 3.62. Lad $\{\hat{d},\ \hat{\theta}\}$ betegne den forventede (støjfri) observation, hvor $\hat{d}$ og $\hat{\theta}$ er fuldstændig bestemt ud fra positionen x og landmærkets koordinater l:

$$\begin{aligned} \hat{d} &= \sqrt{(l_x - p_x)^2 + (l_y - p_y)^2} \\ \hat{\theta} &= \texttt{atan2}(l_y - p_y,\ l_x - p_x) - p_\theta \end{aligned} \tag{3.64}$$

Nu kan sandsynligheden for $o = \{d,\ \theta\}$ givet x dvs. $\{\hat{d},\ \hat{\theta}\}$ udregnes. Lad e_d og e_θ være givet ved:

$$\begin{aligned} e_d &= d - \hat{d} \\ e_\theta &= \theta - \hat{\theta} \end{aligned} \tag{3.65}$$

Hvis observationen følger modellen i formel 3.62 præcist, haves altså $e_d = w_d$ og $e_\theta = w_\theta$.

I det e_d og e_θ modelleres som uafhængige normalfordelte stokastiske variable, kan sandsynlighedsfordelingen $p(o \mid x)$ beregnes som:

$$\begin{aligned} p(o \mid x) &= p(e_d \mid x)p(e_\theta \mid x) \\ &= \frac{1}{\sqrt{2\pi\sigma_d^2}} e^{-\frac{1}{2}\frac{(e_d-\mu_d)^2}{\sigma_d^2}} \frac{1}{\sqrt{2\pi\sigma_\theta^2}} e^{-\frac{1}{2}\frac{(e_\theta-\mu_\theta)^2}{\sigma_\theta^2}} \\ &= \frac{1}{2\pi\sigma_d\sigma_\theta} e^{-\frac{1}{2}\left(\frac{(e_d-\mu_d)^2}{\sigma_d^2}+\frac{(e_\theta-\mu_\theta)^2}{\sigma_\theta^2}\right)} \end{aligned} \tag{3.66}$$

Her er det brugt, at tæthedsfunktionen for en normalfordelt stokastisk variabel er givet ved:

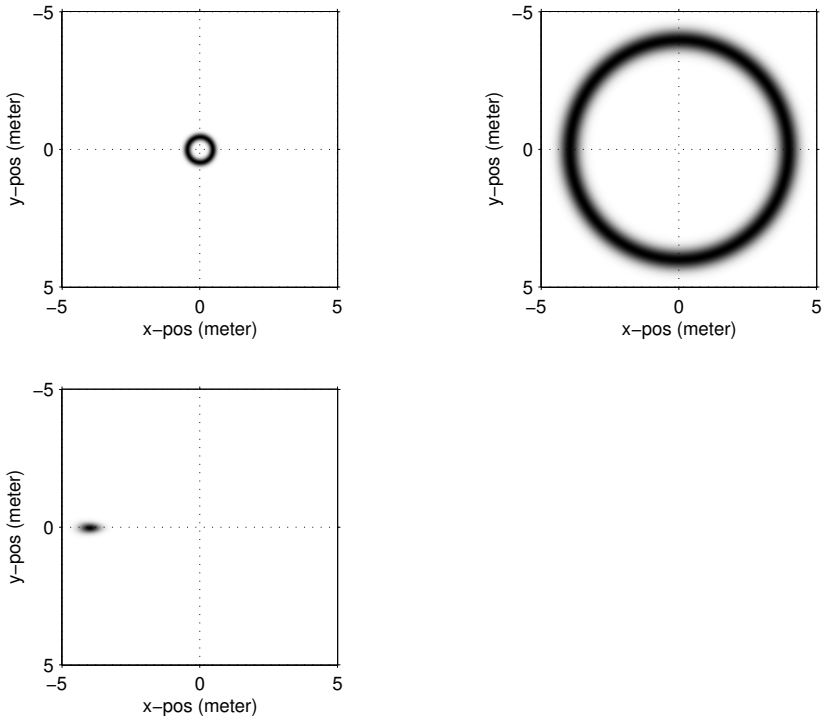
$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} \tag{3.67}$$

Som nævnt i afsnit 3.2.11, er fejlen e_d proportional med afstanden d. Kendes standardafvigelsen σ_{d1} for $d = 1{,}0$ meter, kan standardafvigelsen σ_d for en vilkålig afstand d udregnes som vist i formel 3.56.

Indsættes dette i formel 3.66, haves den endelige observationsmodel:

$$p(o \mid x) = \frac{1}{2\pi\sqrt{d}\sigma_{d1}\sigma_\theta} e^{-\frac{1}{2}\left(\frac{(e_d-\mu_d)^2}{d\sigma_{d1}^2}+\frac{(e_\theta-\mu_\theta)^2}{\sigma_\theta^2}\right)} \tag{3.68}$$

En visualisering af 3.68 kan ses på figur 3.16.



Figur 3.16: Sandsynlighedsfordelingen $p(o \mid x)$ jf. formel 3.68 for ét landmærke placeret i $l = (0,\ 0)$. p_x og p_y varieres i intervallet $[-5;\ 5]$. Mørkt svarer til stor sandsynlighed og lyst svarer til lille sandsynlighed.
Øverst til venstre: $d = 0,5$, vinklen ignoreres og spredningen ved $1,0$ meter er sat til $10\sigma_{d1}$ for illustrationens skyld.
Øverst til højre: $d = 4,0$, ellers samme parametre. Læg mærke til afstandens indflydelse på usikkerheden.
Nederst til venstre: $d = 4,0$ og $\theta = 0,0$. Spredning for afstand ved $1,0$ meter er $10\sigma_{d1}$, og spredning for vinklen er $10\sigma_{\theta}$.

Hvis flere observationer o^i er tilgængelige på samme tid, dvs. hvis flere landmærker er synlige i samme billede, betragtes disse som uafhængige hændelser. $p(o \mid x)$ findes så som produktet af sandsynligheden udregnet for hvert af de n synlige landmærker:

$$p(o \mid x) = \prod_{i=1}^{n} p(o^i \mid x) \tag{3.69}$$

Hver $p(o^i \mid x)$ udregnes efter formel 3.68.

3.3.4 Partikelfilter

At løse formel 3.53 er ikke helt enkelt, især ikke, hvis det skal være effektivt. Løsningen hænger nøje sammen med repræsentationen af $Bel(x)$. Traditionelt har Kalman-filtrering, jf. [Kalman, 1960] og [Maybeck, 1979], været anvendt. Kalmanfiltrering repræsenterer $Bel(x)$ som en normalfordeling og kræver, at estimeringsproblemet er lineært, og at fejlen er normalfordelt. Udviddet (extended) Kalmanfiltrering kan håndtere ikke-lineære problemer ved en lineærisering af estimeringsproblemet, jf. [Gordon et al., 1993].

For at løse generelle ikke-lineære, ikke-normalfordelte estimeringsproblemer, kan $Bel(x)$ ikke repræsenteres analytisk ved en normalfordeling. Grid-baserede metoder, jf. [Fox et al., 1999], repræsenterer $Bel(x)$ ved en diskretisering af hele tilstandsrummet. Dette kræver en nøje afbalancering: En meget finkornet diskretisering vil kræve store mængder hukommelse, og det vil tage lang tid at opdatere $Bel(x)$. Omvendt vil en meget grovmasket diskretisering naturligvis føre til en dårlig repræsentation af $Bel(x)$.

Partikelfiltrering, også kaldet “the condensation algorithm”, jf. [Isard and Blake, 1998], “the bootstrap filter”, jf. [Gordon et al., 1993] og diverse tilsvarende algoritmer med andre navngivninger, jf. [Arulampalam et al., 2002], løser dette problem ved at repræsentere $Bel(x)$ med en mængde af N vægtede stikprøver (eller udfald), som var de fordelt ud fra $Bel(x)$:

$$Bel(x) \approx \left\{x^i,\, w^i\right\}_{i=1}^{N} \tag{3.70}$$

Hvert x^i er et udfald, og w^i er udfaldets vægt. Disse vægtede udfald kaldes også partikler, heraf navnet partikelfilter. Til at løse formel 3.53 findes flere forskellige

algoritmer. Her er valget faldet på SIR-algoritmen (Sample Importance Resampling). Dette er også algoritmen [Thrun et al., 2000] anvender. De kalder kombinationen af rekursiv bayes-filtrering og partikelfiltrering for "Monte Carlo Localization", heraf dette hovedafsnits navn. Af følgende grunde anvendes SIR-algoritmen til at løse estimeringsproblemet i formel 3.53: Algoritmen kan løse ikke-lineære, ikke-normalfordelte estimeringsproblemer, er let at forstå og implementere og giver præcise resultater. For en sammenligning af SIR-filtret med en række andre partikelfiltre samt med det udviddede Kalman-filter og et grid-baseret filter, se [Arulampalam et al., 2002]. SIR-algoritmen er vist i pseudokode i tabel 3.4 og forklares i det følgende.

$$\left[\left\{x_t^i,\, w_t^i\right\}_{i=1}^N\right] \;=\; \text{SIR}\;\left[\left\{x_{t-1}^i,\, w_{t-1}^i\right\}_{i=1}^N,\, a_{t-1},\, o_t\right]$$

- FOR i = 1 : N
  - Træk $x_t^i \;\sim\; p(x_t^i \mid x_{t-1}^i,\, a_{t-1})$
  - Beregn $w_t^i \;=\; p(o_t \mid x_t^i)$
- END FOR
- Beregn total vægt: $s \;=\; \text{SUM}\;\left[\left\{w_t^i\right\}_{i=1}^N\right]$
- FOR $i \;=\; 1 \;:\; N$
  - Normaliser: $w_t^i \;=\; \frac{w_t^i}{s}$
- END FOR
- Resample:
  - $y_t \;=\; x_t$
  - FOR i = 1 : N
    - Udtag med tilbagelægning x_t^i fra y_t med sandsynlighed w_t^i
  - END FOR

Tabel 3.4: Pseudokode for SIR-filteret.

Først anvendes bevægelsesmodellen til at forudsige, eller give et bud på, den nye tilstand x_t ud fra den gamle tilstand x_{t-1} og bevægelsen a_{t-1}:

$$\left\{x_t^i \sim p(x_t^i \mid x_{t-1}^i,\, a_{t-1})\right\}_{i=1}^N \tag{3.71}$$

Herefter anvendes observationsmodellen til at vægte hver x_t^i ud fra, hvor sandsynlig x_t^i er, givet den aktuelle observation o_t:

$$\left\{w_t^i = p(o_t \mid x_t^i)\right\}_{i=1}^N \tag{3.72}$$

Disse to skridt svarer til det noget simplere SIS-partikelfilter, jf. [Arulampalam et al., 2002].

Dette filter har dog et degenereringsproblem, idet efter nogle få iterationer vil de fleste partikler have en meget lille vægt. Derfor foretages nu en genudtrækning (eng: resample).

Først normeres vægtene, så $\sum_{i=1}^N w_t^i = 1$. Dernæst foretages en vægtet genudtrækning med tilbagelægning af N partikler fra den gamle population $\left\{x_t^i,\ w_t^i\right\}$. Sandsynligheden for at x_t^i udtages er givet ved vægten w_t^i. Det noteres, at efter genudtrækningen sættes alle vægte $w_t^i = N^{-1}$, da fordelingen af x_t^i efter genudtrækningen afspejler den vægtede fordeling før genudtrækningen.

SIR-algoritmen og udledningen af formel 3.53 kan give det indtryk, at det kræves, at bevægelsesdata a og observationsdata o kommer i alternerende rækkefølge $o_0,\ a_0,\ o_1,\ a_1$, etc. Dette er dog ikke tilfældet. Kommer der fx flere bevægelsesdata $a_{t-2},\ a_{t-1}$ mellem to observationer $o_{t-2},\ o_t$, anvendes bevægelsesmodellen blot flere gange i træk:

$$\begin{aligned}&\left\{x_{t-1}^i \sim p(x_{t-1}^i \mid x_{t-2}^i,\ a_{t-2})\right\}_{i=1}^N\ ,\\ &\left\{x_t^i \sim p(x_t^i \mid x_{t-1}^i,\ a_{t-1})\right\}_{i=1}^N\end{aligned} \tag{3.73}$$

Dette er essentielt. Kører robotten nemlig for langt mellem opdateringen af positionen ud fra tachometermålingerne, vil tilnærmelsen at robotten har kørt på en cirkelbue mellem hver opdatering blive meget grov og dermed unøjagtig, jf. afsnit 3.1.1. Landmærkegenkendelsen kan ikke forventes at levere data oftere end omtrent en gang i sekundet, da det tager så lang tid at processere hvert billede.

Om end mindre aktuelt i Murphys tilfælde, kan flere observationsdata mellem hver bevægelsesdata også håndteres. Disse kan nemlig så betragtes som uafhængige hændelser, og anvendes som vist i formel 3.69.

Bemærk at genudtrækningen således kun foretages i de tilfælde, hvor nye bevægelsesdata modtages og de sidst modtagne data var observationsdata.

I pseudokoden i tabel 3.4 er det ikke nærmere specificeret, hvorledes genudtrækningen foretages. En metode til at finde udfald x^i med sandsynlighed $p(x^i)$,

hvor $\sum p(x^i) = 1$ er den inverse kumulerede sandsynlighedsfunktions metode, jf. [Gentle, 2003, pp. 102–109]. Denne involverer dog en søgning, hvilket giver metoden en køretid på $O(N \log N)$. Dette er uhensigtsmæssigt, da den øvrige SIR-algoritme kører i $O(N)$ tid. Derfor anvendes den systematiske genudtrækningsmetode, jf. [Arulampalam et al., 2002, algorithm 2]. Pseudokode for denne findes i tabel 3.5, dog er her rettet en indeksfejl.

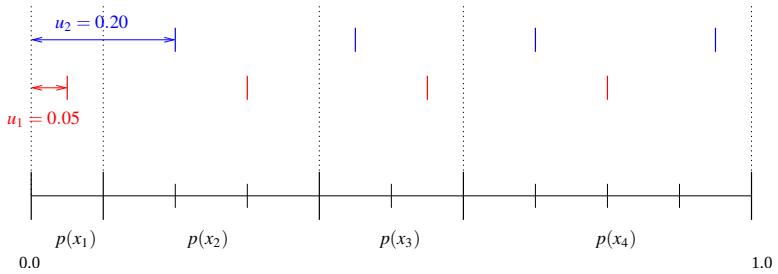
$\left[\left\{x_t^i,\, w_t^i\right\}_{i=1}^N\right] = \text{RESAMPLE}\ \left[\left\{x_t^i,\, w_t^i\right\}_{i=1}^N\right]$

- $y_t = x_t$
- Initialiser cdf: $c^1 = w_t^1$
- FOR i = 2 : N
  - Konstruer cdf: $c^i = c^{i-1} + w_t^i$
- END FOR
- $i = 1$
- Udtræk startpunkt: $u \sim U(0,\, N^{-1})$
- FOR $j = 1 : N$
  - WHILE $u > c^i$
    - $i = i + 1$
  - END WHILE
  - $x_t^j = y_t^i$
  - $w_t^j = N^{-1}$
  - $u = u + N^{-1}$
- END FOR

Tabel 3.5: Pseudokode for systematisk genudtrækning.

Algoritmen virker på følgende måde: Først findes de kumulerede sandsynligheder $c^i = \sum_{j=1}^{i} p(x^j)$. Så findes et udfald $u \sim U(0,\, N^{-1})$ af en ligefordeling. Nu gentages følgende så længe $u \leq 1$: Hvis $c^{i-1} < u \leq c^i$, er x^i et udfald, og N^{-1} adderes til u.

For et eksempel på algoritmen, se figur 3.17. Algoritmen svarer dog ikke helt nøjagtigt til vægtet udtagning med tilbagelægning. Dette ses let af eksemplet i figur 3.17. Udfaldet x_4 vil nemlig altid blive udtrukket mindst én gang og højst to gange. I praksis fungerer algoritmen dog godt jf. [Douc et al., 2005], som sammenligner en række genudtrækningsalgoritmer.



Figur 3.17: Eksempel på systematisk genudtrækning. Generering af 4 udfald af den stokastiske variabel X med udfaldsrum $\{x_1,\ x_2,\ x_3,\ x_4\}$ og med sandsynlighedsfordeling $p(x_1) = 0,1$, $p(x_2) = 0,3$, $p(x_3) = 0,2$ og $p(x_4) = 0,4$. Først findes et udfald af $U(0,\ \frac{1}{4})$, her illustreret ved de to tilfælde $u_1 = 0,05$ og $u_2 = 0,20$. For u_1 giver dette udfaldene $\{x_1,\ x_2,\ x_3,\ x_4\}$. For u_2 giver dette udfaldene $\{x_2,\ x_3,\ x_4,\ x_4\}$.

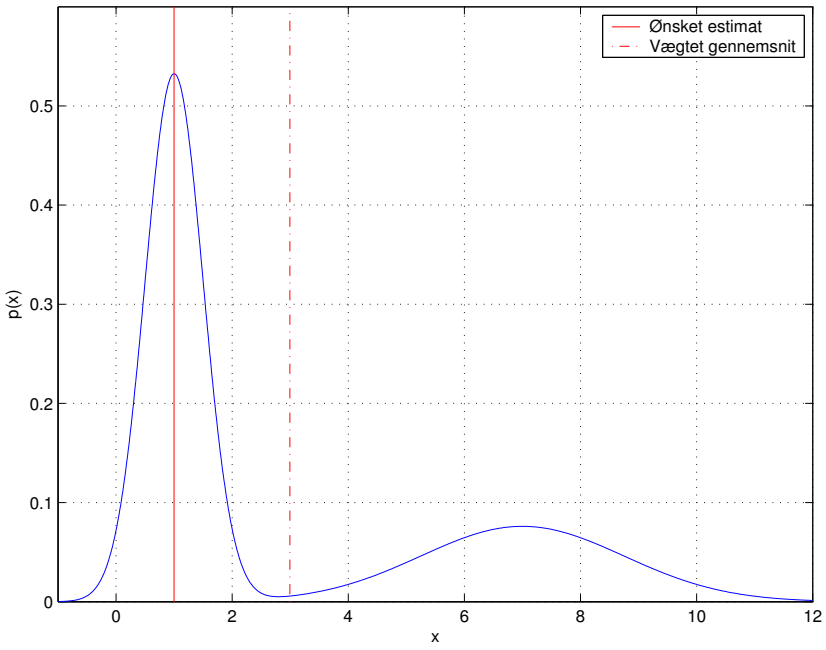
Partikelfiltret giver løbende som uddata en opdateret sandsynlighedsfordeling $Bel(x_t)$ repræsenteret som en mængde af partikler (vægtede stikprøver). Ud fra denne sandsynlighedsfordeling ønskes det, at finde den mest sandsynlige position x_t, dvs. det bedste estimat af den nuværende position.

Det virker her umiddelbart helt oplagt at bruge den vægtede middelværdi $\bar{x}_t$, der udregnes som:

$$\bar{x}_t = \sum_{i=1}^{N} w_t^i x_t^i \tag{3.74}$$

Dette kan dog give temmelig uønskede resultater. Er fordelingen multimodal, vil det vægtede gennemsnit give et gennemsnit af hele sandsynlighedsfordelingen, se figur 3.18.

Det vurderes her, at det ønskede positionsestimat må være, hvor den vægtede sandsynlighedstæthed er størst. [Rekleitis, 2003] giver fire metoder til at finde bedste estimat, deriblandt det "robuste gennemsnit", som er det vægtede gennemsnit af de partikler, der ligger mindre end en vis afstand ε fra partiklen med størst vægt. [Röfer and Jüngel, 2003] bruger gennemsnittet af den største klynge af partikler, hvor klyngerne er defineret ud fra en diskretisering af udfaldsrummet. Disse to



Figur 3.18: Eksempel på en bimodal sandsynlighedsfordeling. Her er det vægtede gennemsnit samt det ønskede estimat indtegnet.

nævnte metoder giver begge bud på, hvordan positionsestimatet ud fra hvor den vægtede sandsynlighedstæthed er størst, kan udregnes.

For at undgå at skulle afprøve en række estimatorer for at finde den bedste, vil det vægtede gennemsnit blot blive anvendt her. Det vægtede gennemsnit har dog en komplikation, nemlig problemet med at tage gennemsnittet af en mængde vinkler, pga. vinklernes cirkularitet. Derfor anvendes følgende formel for at finde gennemsnitsvinklen, jf. [Röfer and Jüngel, 2003]:

$$\bar{p}_\theta = \texttt{atan2}\left(\sum_{i=1}^{N} \sin p_\theta^i,\ \sum_{i=1}^{N} \cos p_\theta^i\right) \tag{3.75}$$

3.3.5 Samlet løsning

I det følgende vil en samlet løsning på estimeringsproblemet blive vist. I afsnit 3.3.4 blev problemet løst vha. partikelfiltret. Løsningen har dog to problemer.

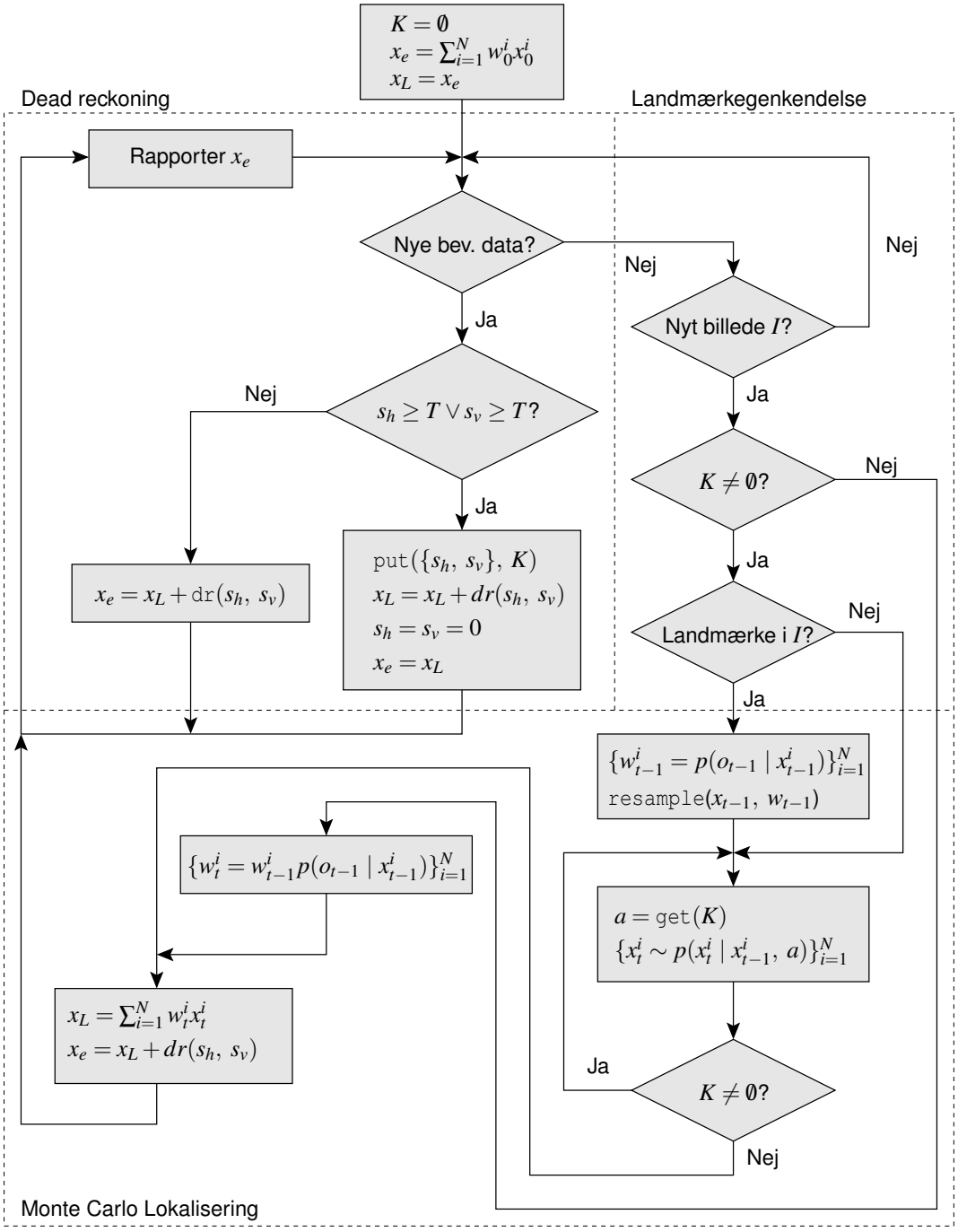
For det første ønskes en positionsestimering x_e opdateret så ofte som muligt. Hver eneste gang et tachometer inkrementeres, skal et nyt estimat gives, modsat det i afsnit 3.3.4 beskrevne, hvor positionsestimeringen opdateres omtrent hver $10.$ cm. Denne meget hyppige opdatering bliver nemlig nødvendig, når robotten skal følge en given rute, hvilket vil blive beskrevet i kapitel 4.2.

For det andet er landmærkegenkendelsen et halvt til et helt sekund om at bearbejde hvert billede. Kører robotten med en fart på en halv meter pr. sekund, når robotten altså at køre 0,25 – 0,50 meter fra et billede er taget og til landmærkegenkendelsen kan give observationsdata ud fra billedet.

Det første problem løses ved at opdatere en mellemposition x_L hver $T = 10$ cm, så $x_L = x_L + \texttt{deadreckoning}(s_h,\ s_v)$ og $s_h = s_v = 0$ og ud fra x_L at give et nyt positionsestimat x_e hver gang et tachometer opdateres dvs. så $x_e = x_L + \texttt{deadreckoning}(s_h,\ s_v)$ uden nulstilling af tachometrene.

Løsningen på det andet problem er at gemme bevægelsesdata $\{s_h,\ s_v\}$ i en kø K fra et billede I_{t-1} er taget og indtil landmærkegenkendelsen giver observationsdata o_{t-1} eller konstaterer at intet landmærke findes i I_{t-1}. Nu opdateres vægtene w_{t-1}^i som hører til tilstanden x_{t-1} da billedet blev taget, og en genudtrækning foretages. Bevægelsesdata i køen K bruges så til at opdatere tilstanden x_{t-1} indtil den svarer til den aktuelle tilstand x_t.

Flowdiagrammet i figur 3.19 giver et overblik over den samlede løsning som fungerer i realtid, hvor partikelfiltret er kombineret med hensyntagen til forsinkelsen i observationsdata, samt kravet om hyppig opdatering af positionsestimatet.



Figur 3.19: Flowdiagram over den samlede løsning. x_e er det ønskede, hyppigt opdaterede positionsestimat, x_L er en mellemposition, som opdateres hver $T = 10$ cm, og `dr`() er `deadreckoning`()-funktionen, svarende til formel 3.10. K er en kø indeholdende nul eller flere bevægelsesdatapar $\{s_h,\ s_v\}$. I er det behandlede billede og $\{x_t^i,\ w_t^i\}$ er tilstanden til tiden t.

3.3.6 Delkonklusion og mulige forbedringer

I dette hovedafsnit blev anvendt en statistisk tilgang til positionsestimering. En rekursiv opdateringsformel blev udledt, som givet bevægelses- og observationsdata kan opdatere en sandsynlighedsfordeling, som beskriver tiltroen til en given position. Herefter blev en bevægelsesmodel og en observationsmodel opstillet for robotten Murphy, på grundlag af afprøvningen af dead reckoningen og landmærkegenkendelsen. Til sidst blev vist, hvordan opdateringsformlen i praksis kan anvendes.

Der kan nævnes en mængde mere eller mindre omfattende forslag til forbedringer i forhold til indholdet i dette hovedafsnit, i det følgende opsummeret i vilkårlig rækkefølge. Bevægelsesmodellen kan forbedres, så den tager hensyn til acceleration, andre hastigheder end $0,5$ meter pr. sekund og andre former for underlag. Partikelfiltreringsalgoritmen kan også forbedres, fx med en mere avanceret algoritme end SIR-algoritmen, eller med udvidelse til mixture-mcl, jf. [Thrun et al., 2000]. Brugen af vægtet gennemsnit som beregning af det bedste estimat af positionen er heller ikke optimal, som før nævnt. Markov-antagelsen, jf. afsnit 3.3.1 er også meget restriktiv. Det ville være ønskværdigt, hvis robotten kunne navigere i i hvert fald moderat dynamiske omgivelser, se [Fox et al., 1999] for forslag til en løsning på dette.

3.4 Delkonklusion og mulige forbedringer

Dette kapitel startede med at udlede et positionsestimat alene baseret på dead reckoning ud fra robottens tachometre. Det blev konstateret, at et sådant positionsestimat vil have en akkumulerende fejl. Dernæst blev det vist, hvordan kunstige landmærker kan genkendes, og hvordan afstand og vinkel til disse kan beregnes. I sidste del af kapitlet blev resultatet af dead reckoningen og landmærkegenkendelsen kombineret, hvilket gav et forbedret positionsestimat.

Den beskrevne landmærkegenkendelse virker som en såkaldt “feature extraction”, dvs. ud fra et helt billede udtages en feature-vektor, $o = \{d,\ \theta\}$ indeholdende kun to tal. Dette er en kolosal datareduktion af en størrelsesorden 100.000:1. Det er klart, at en del vigtige data må gå tabt i processen. Dette inkluderer bl.a. naturligt forekommende landmærker, som også kan give information om robottens position. [Thrun, 1998] foreslår fx en selvlærende metode til genkendelse af landmærker, som kan give en bedre udnyttelse af informationerne i billedet. For en diskussion af forskellige metoder til positionsestimering baseret på landmærker, se [Sim and Dudek, 1998].

Dead reckoningen ud fra tachometrene anvender heller ikke alle tilgængelige data, som det også kort blev diskuteret i afsnit 3.3.2, hvor bevægelsesmodellen blev opstillet. Informationen om forhjulsvinklen og motorspændingen er i hvert fald delvis tilgængelig, og kunne sammen med informationen om fx robottens vægt, luft- og rullemodstand, mængde af slør i styretøjet og underlagets beskaffenhed, bruges til at opstille en model for robottens kinematik. Denne model kunne sammen med tachometermålingerne bruges til at give en endnu bedre bevægelsesmodel. Det må dog påregnes, at opstillingen af en bevægelsesmodel på grundlag af alle tilgængelige informationer om robottens bevægelse kan blive temmelig kompliceret. Det kan så forslås, at en selvlærende algoritme anvendes, som fx et neuralt netværk eller en genetisk algoritme.

I det Murphy allerede har påmonteret en række afstandssensorer, nemlig tre IR-afstandssensorer og en ultralydssensor, kan disse anvendes sammen med kortet over omgivelserne til at give en bedre positionsestimering, som fx [Fox et al., 1999] viser det. Kortet kan også anvendes til at udelukke "umulige" positioner i $Bel(x_t)$, hvor robotten ikke rent fysisk kan befinde sig, fx hvis robotten vil være delvist inde i en væg.

Faktisk kan afstandssensorerne anvendes til på samme tid at bestemme robottens position *og* opbygge et kort over den verden robotten kører rundt i. Dette er det såkaldte SLAM-problem (Simultaneous Localization And Mapping), se [Montemerlo and Thrun, 2003], [Hu et al., 2004] og [Dissanayake et al., 2001], hvilket Monte Carlo lokalisering faktisk er meget velegnet til at løse. I denne opgave er ønsket at kunne planlægge en rute ud fra et kendt kort. I stedet for manuelt at opmåle de omgivelser robotten bevæger sig rundt i, til brug for kortet, kan SLAM altså bruges til automatisk at generere et kort. En sådan løsning er dog fravalgt her, for at begrænse opgavens omfang.

Sidst kan det jo diskuteres, om den statistiske lokalisering, jf. afsnit 3.3.1 er den bedste måde at gribe estimeringsproblemet an på. Det har ikke været muligt at finde alternative metoder som giver et tilsvarende eller bedre resultat, og søger man indenfor litteraturen på området ser det ud til at det, på nuværende tidspunkt, er den bedste metode.

Kapitel 4

Styring

Kapitel 2 om ruteplanlægning beskrev, hvordan en rute γ bestående af linjestykker og cirkelbuer kan findes, givet en start- og en målposition, som det er muligt for en bil-agtig robot at følge. Kapitel 3 om positionsestimering beskrev, hvordan robotten kan estimere sin aktuelle position x_e.

Givet ruten γ og den aktuelle position x_e, ønskes det nu at få den bil-agtige robot Murphy til at følge γ fra start til mål. Det er underforstået, at Murphy starter med at være placeret på startpositionen.

Murphy har to aktuatorer, som den kan bevæge sig ved hjælp af, nemlig to dc-motorer som driver baghjulene og en servomotor, der styrer forhjulsvinklen.

Afsnit 4.1 herunder beskriver, hvordan robottens hastighed styres. Afsnit 4.2 beskriver, hvordan forhjulsvinklen styres.

4.1 Styring af robottens hastighed

Robottens hastighed ønskes styret på to måder. For det første ønskes det at kunne få Murphy til at køre med en given hastighed. Dette beskrives i afsnit 4.1.1. For det andet ønskes det at kunne få robotten til at stoppe op på et givet sted på en rute. Når en rute skal følges, skal robotten nemlig stoppe op, når der sker et skift mellem forlæns og baglæns kørsel på ruten. Desuden skal robotten stoppe op på målpositionen. Dette beskrives i afsnit 4.1.2.

4.1.1 Hastighedsregulering

Når hastigheden af robotten skal reguleres, må det først konstateres, hvad der kan reguleres på robotten, ud fra hvilke informationer reguleringen kan foretages, og hvad målet for reguleringen er.

Målet for hastighedsreguleringen er at holde robottens hastighed v_R, jf. figur 2.1, så tæt som muligt på en given hastighed v_w.

Robottens hastighed kan reguleres vha. de to dc-motorer, som driver robottens baghjul. Mere specifikt kan motorernes fælles forsyningsspænding reguleres vha. pulsbreddemodulering, jf. [Kjeldsen, 2003, afsnit 6.3.2]. Dette svarer til at kunne regulere motorspændingen V_m ud fra en faktor c som:

$$V_m = cV_{bat}, \qquad 0 \leq c \leq 1 \tag{4.1}$$

Her betegner V_{bat} batterispændingen, og faktisk er c diskretiseret, men vil her blive betragtet som et reelt tal.

Informationen, der kan reguleres ud fra, er robottens aktuelle hastighed v_R, den ønskede hastighed v_w og batterispændingen V_{bat}. Denne kan Murphy måle med en sensor. Hastigheden v_R kan findes ud fra positionsestimatet x_e eller blot ud fra tachometermålingerne s_h og s_v, som er rigeligt præcise til formålet.

Der skal altså findes en funktion u, der beregner faktoren c ud fra v_w, v_R og V_{bat}. Da det ønskes at minimere forskellen mellem v_w og v_R, ønskes altså en funktion $u(e, V_{bat})$ fundet, som minimerer $e = v_w - v_R$. Det antages først, at $v_w \geq 0$ og $v_R \geq 0$, hvilket der rettes op på senere.

Motorspændingen reguleres, jf. formel 4.1, så som:

$$V_m = u(e, V_{bat})V_{bat} \tag{4.2}$$

Til løsning af problemet anvendes en PID-regulering (PID: Proportional, Integral, Differential), jf. [Ole Jannerup, 2000, afsnit 6.3] og [Kjeldsen, 2003, afsnit 8.1]. Denne vides at fungere godt på Murphy, da den blev anvendt ved RoboCup 2004 og RoboCup 2005, jf. [rob, 2004]. u defineres som den vægtede sum af fejlen, integralet af fejlen og den afledede af fejlen:

$$u(e) = k_p e + k_i \int e + k_d \dot{e} \tag{4.3}$$

Formlen kan diskretiseres, så den kan implementeres i en løkke, som jævnligt aflæser tachometrenes værdi og opdaterer pulsbreddemoduleringen af motorspændingen:

$$u(e(j)) = k_p e(j) + k_i \sum_{x=0}^{j} e(x)\Delta t + k_d \frac{e(j) - e(j-1)}{\Delta t} \tag{4.4}$$

Her betegner j iterationsnummeret, og Δt betegner tiden siden sidste opdatering. Summen af fejlen kan udregnes som en løbende opdateret sum. For at kompensere for variationer i batterispændingen, divideres med denne, jf. formel 4.2:

$$u(e(j), v_{bat}(j)) = \frac{1}{V_{bat}(j)} u(e(j)) \tag{4.5}$$

Yderligere ønskes der kompenseret for forskellen mellem den statiske friktion og rullemodstanden. Det har vist sig, at der er en stor forskel mellem dem på Murphy. Reguleringen bliver mere stabil, hvis der tages hensyn til denne forskel. Kompenseringen foretages som følger:

$$u_f(e(j), v_{bat}(j)) = u(e(j), V_{bat}(j)) + \begin{cases} \frac{k_s}{V_{bat}} & \text{hvis } v_R = 0 \\ \frac{k_r}{V_{bat}} & \text{hvis } v_R > 0 \end{cases} \tag{4.6}$$

Her er k_s en konstant svarende til den statiske friktion, og k_r er en konstant svarende til rullemodstanden.

Hvis robotten kører meget for hurtigt, dvs. hvis $v_R \gg v_w$, bliver værdien af u_f mindre end 0, hvilket ikke kan bruges direkte i reguleringen, jf. formel 4.1. Tilfældet svarer til, at reguleringen gerne vil bremse robotten op. Murphy har ingen bremser, men motorerne kan bruges som motorbremser, ved at kortslutte dem. Kortslutningen kan også pulsbreddemoduleres. Pulsbreddemoduleringen af kortslutningen afhænger naturligvis ikke af V_{bat}. Giver $u(e(j)) < 0$, kan motorbremsen anvendes som:

$$M_b = k_b |u(e(j))| \tag{4.7}$$

Her er k_b en faktor, som kompenserer for, at motorernes bremsekraft er mindre end motorernes kraft ved samme værdi af $|u(e(j))|$, og M_b betegner moduleringen af motorbremsen.

For at understøtte at robotten også skal kunne bakke, bemærkes at motorernes retning styres uafhængigt af pulsbreddemoduleringen. Hvis v_w er mindre end 0, svarende til at robotten skal bakke, sættes motorernes retning til baglæns, og $|v_w|$ og $|v_R|$ anvendes i beregningen af u. Det er rimeligt at antage at v_w og v_R altid har samme fortegn, hvis der blot bremses helt op, så retningsskift på motorerne kun foretages, når robottten holder stille.

Som en udviddet motorbremse kan motorerne også sættes til at køre i modsat retning af kørselsretningen. Dette er mere effektivt end den simple motorbremse, men det komplicerer motorstyringen.

Størrelsen af de anvendte konstanter k_p, k_i, k_d, k_s, k_r og k_b kan findes empirisk. Specielt kan det for k_p, k_i og k_d betale sig først at finde en god værdi for k_p, som bestemmer stigtiden, med $k_i = k_d = 0$. Derefter findes k_d, som begrænser oversvinget, med $k_i = 0$. Sidst findes k_i, som begrænser den stationære fejl. For mere om regulering, se [Ole Jannerup, 2000] og [of Michigan, 1997].

4.1.2 Stopproblemet

Det ønskes her at finde en metode, så robotten kan stoppe op på et bestemt sted på en given rute γ. Afsnit 4.1.1 beskrev, hvordan robottens hastighed v_R kan styres ud fra en ønsket hastighed v_w. Afstanden s_s langs γ til stoppunktet kan udregnes som afstanden fra en position p_s på γ langs γ til stoppunktet. For en definition af p_s, se afsnit 4.2.3. Desuden kan den ønskede bremseacceleration a_w bestemmes, enten ved at måle den maksimale bremseacceleration for Murphy, eller hvis denne er for voldsom, ved at vælge en noget mindre a_w.

Først udregnes bremseafstanden s_b, som angiver, hvor langt et stykke Murphy vil køre, før den stopper, hvis den bremser op fra hastigheden v_R med accelerationen a_w, jf. [Gieck, 1979, formel I 19]:

$$s_b = \frac{v_R^2}{2a_w} \tag{4.8}$$

Vha. denne formel kan bremsehastigheden også udregnes, ud fra afstanden til stoppunktet s_s og a_w:

$$v_b = \sqrt{2a_w s_s} \tag{4.9}$$

For at få hastighedsreguleringen til at få robotten til at stoppe op på et givent punkt på γ, indbygges et tjek:

$$v'_w = \begin{cases} v_w & \text{hvis } s_s > s_b \\ v_b & \text{hvis } s_s \leq s_b \end{cases} \tag{4.10}$$

v'_w bruges nu i stedet for v_w i hastighedsreguleringen.

4.2 Ruteføglning

Når en given rute ønskes fulgt, ønskes det, at punktet R, nemlig midtpunktet af robottens bagaksel, følger ruten med så lille en afvigelse som muligt, jf. figur 2.1. Grunden til dette er, at ruteplanlægningen tager udgangspunkt i bevægelsen af punktet R. I [Rode and Kjeldsen, 2004, afsnit 4.5] foreslåes en kurveføglningsalgoritme, som ved simulation har vist sig at være robust og stabil. Den algoritme søger dog at få midtpunktet af forhjulsakslen F til at følge ruten. I det følgende vil en algoritme blive beskrevet, som tager udgangspunkt i kurveføglningsalgoritmen fra [Rode and Kjeldsen, 2004, afsnit 4.5], men som søger at få punktet R til at følge ruten.

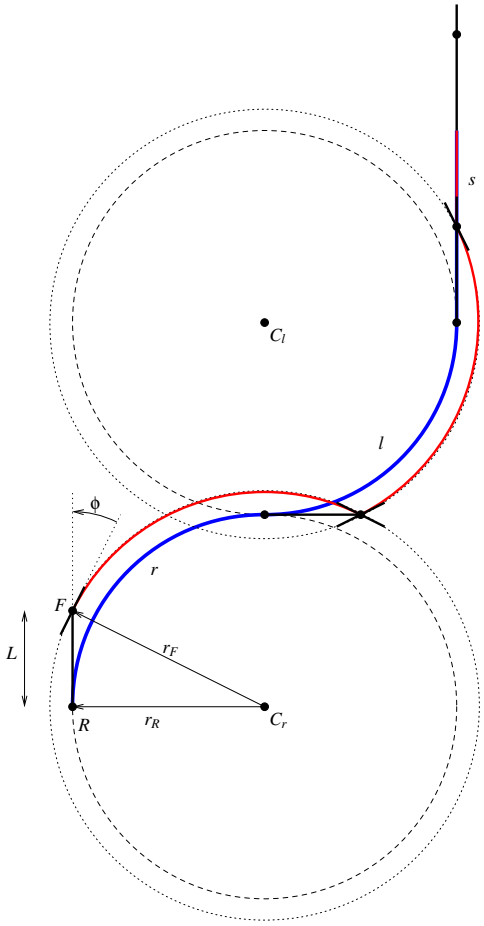
Overordnet virker algoritmen ved, ud fra den ønskede rute γ, som det ønskes at midtpunktet af bagakslen R skal følge, at generere en rute γ_F, som midten af forhjulsaksen F skal følge. Dette grunder i, at det kun er vinklen af forhjulene på robotten, som kan styres. Figur 4.1 opridser situationen. Jeg vil herunder udlede, hvordan segmenterne af γ_F beregnes ud fra segmenterne af γ. Udledningen er opdelt i to tilfælde, da et segment enten kan være en cirkelbue eller et linjestykke.

4.2.1 Cirkelbue

Et segment af den ønskede rute γ antages at være en cirkelbue c. Skal R følge cirkelbuen med radius r_R, må F følge en cirkelbue c_F med større radius, kald denne r_F. r_F og forhjulvinklen ϕ, som får F til at følge r_F, kan udregnes, jf. 4.1, som følger:

$$\tan\phi = \frac{L}{r_R} \qquad \Rightarrow \qquad \phi = \arctan\left(\frac{L}{r_R}\right) \tag{4.11}$$

$$r_F = \sqrt{{r_R}^2 + L^2} \tag{4.12}$$



Figur 4.1: En rute af formen $r^+l^+s^+$. Med blåt: Ruten, som midtpunktet af robottens bagaksel skal følge. Med rødt: Kurven, som midten af robottens foraksel F skal følge, hvis R skal følge ruten med blåt. For kørsel langs lige linjer er de to kurver delvist sammenfaldne. Robotten er vist som en bicykel.

Cirkelbuen c_F er naturligvis også længere end c, dens længde beregnes som:

$$\| c_F \| = \frac{r_F}{rR} \| c \| \tag{4.13}$$

Er startpositionen for det oprindelige segment c givet ved $p = (x, y, \theta)$, kan startpositionen p_c for c_F beregnes ud fra ϕ og L som:

$$p_c = p + (L\cos\theta, L\sin\theta, \phi) \tag{4.14}$$

Cirkelbuen c_F, som F skal følge, er nu helt bestemt, nemlig ud fra startpositionen p_c, dens radius r_F, dens længde $\| c_F \|$ og dens rotationsretning, som er den samme som for c.

4.2.2 Linjestykke

Et segment af den ønskede rute γ antages at være et linjestykke l. Linjestykket l_F, som midtpunktet af forakslen F skal følge, har samme længde og orientering som l, det er blot translateret et stykke L frem i forhold til l. Dvs. startpositionen p_l for l_F er givet ved:

$$p_l = p + (L\cos\theta, L\sin\theta, 0) \tag{4.15}$$

Linjestykket l_F, som F skal følge, er nu helt bestemt, nemlig ud fra startpositionen p_l og længden $\| l_F \| = \| l \|$.

4.2.3 Regulering af forhjulsvinklen ϕ

Forhjulsvinklen ϕ reguleres nu ud fra fejlen v_e (e for error) på den aktuelle position x_e. Med fejlen på den aktuelle position menes forskellen mellem den aktuelle position x_e, og hvor robotten burde være på γ_F. Kald positionen, hvor robotten burde være på γ_F, for p_s. p_s defineres som den position på γ_F, der ligger euklidisk nærmest punktet F på robotten, dog med den begrænsning, at hvis en position på γ_F er fundet som nærmest, og positionen ligger på et segment σ_i af ruten, vil segmenter s_j, hvor $j < i$ efterfølgende blive ignoreret, når der skal findes nærmeste positioner på γ_F. Dette forhindrer, at robotten ved en fejl kommer til at følge segmenter, der ligger tidligere på ruten, fx hvis dele af ruten ligger meget tæt på hinanden, hvis ruten ligefrem krydser sig selv, eller hvis robotten er kommet langt væk fra ruten. For en

udregning af, hvordan positionen p_s på ruten findes, se [Rode and Kjeldsen, 2004, afsnit 4.5.2].

v_e defineres altså som:

$$v_e = x_e - p_s \tag{4.16}$$

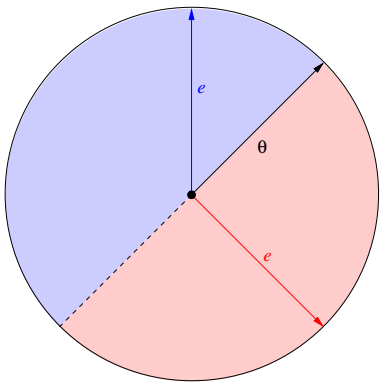
Ud fra vektoren v_e kan en skalar fejl nu udregnes:

$$e = \text{sign}(v_e, \theta_r) \parallel v_e \parallel \tag{4.17}$$

Her er θ_r robottens nuværende orientering, og $\text{sign}(v_e, \theta_r)$ er en funktion defineret som:

$$\text{sign}(v, \theta) = \begin{cases} 1 & \text{hvis vinklen af } v \text{ er mindre end } \theta \\ 0 & \text{hvis vinklen af } v \text{ er den samme som } \theta \\ -1 & \text{hvis vinklen af } v \text{ er større end } \theta \end{cases}$$

Da vinkler regnes modulo 2π, er definitionen af "større end" og "mindre end" ikke helt oplagt, men se figur 4.2, som anskueliggører det grafisk.



Figur 4.2: Med rødt: Vinklen af e er mindre end θ, $\text{sign}(e, \theta) = 1$, og der skal drejes til højre. Med blåt: Vinklen af e er større end θ, $\text{sign}(e, \theta) = -1$, og der skal drejes til venstre. Efter [Rode and Kjeldsen, 2004, figur 16].

Hjulvinklen ϕ_w beregnes nu ud fra en PID-regulering, ligesom formel 4.3:

$$\phi = \phi_0 + k_p e + k_i \int e + k_d \dot{e} \tag{4.18}$$

Her betegner ϕ_0 den forhjulsvinkel, som blev fundet under udregningen af c_F, hvis det aktuelle segment er en cirkelbue. Hvis det aktuelle segment er en linje, er $\phi_0 = 0$.

Formel 4.18 kan også diskretiseres, analogt til formel 4.4.

4.2.4 Diskontinuitet af rutens krumning

Som det ses på figur 4.1 har rutens krumning diskontinuiteter, hvor rutens segmenter mødes, hvis det da ikke er to cirkler med samme krumning eller to linjer, som mødes. Robottens forhjul skal groft sagt være to steder på samme tid. Da robotten kun kan bevæge forhjulene med en omtrent konstant, begrænset vinkelhastighed, kan robotten kun følge en rute helt nøjagtigt, hvis den stopper op i hvert punkt, hvor to segmenter mødes. Dette er naturligvis uhensigtsmæssigt.

Dette problem blev allerede konstateret i afsnit 2.6.1. Problemet er dog ikke større for Murphy, end at det kan løses på en pragmatisk måde. Robottens hastighed begrænses blot til $0,5$ meter pr. sekund, hvilket alligevel er nødvendigt, for at positionsestimeringen og specielt landmærkegenkendelsen kan følge med. Dette begrænser samtidig den motion blur der vil opstå, hvis robotten kører meget hurtigt. Robottens topfart er i forvejen begrænset af motorernes styrke, og ligger på ca. en meter pr. sekund. Samtidig vælges en mindste drejeradius ρ_{min}, som er lidt større end Murphys reelle mindste drejeradius. Sammenholdt med styrevinklens maksimale hastighed på ca. to radianer pr. sekund, giver denne løsning et rimeligt resultat.

Der tages dog også hensyn til problemet på en anden måde. Skift i styrevinklen "forudses" nemlig. Når robotten nærmer sig et sted på ruten, hvor rutens krumning ændrer sig, drejes forhjulene mod den nye hjulvinkel, lidt før stedet nås. Dette gøres rent praktisk ved at ændre ϕ_0 i formel 4.18 til et vægtet gennemsnit af den nuværende ϕ_0 og ϕ_0 for det næste segment af ruten, vægtet med den tid t, det vil tage robotten at nå stedet, hvor det nuværende og det næste segment mødes:

$$t = \frac{l}{v_R} \tag{4.19}$$

Her er l afstanden til stedet, hvor segmenterne mødes. Skift forudses dog ikke, når segmenterne, der mødes, har forskellig kørselsretning. Her skal robotten nemlig alligevel stoppe op for at skifte kørselsretning, og der er rigelig tid til at ændre forhjulsvinklen samtidig.

4.3 Delkonklusion og mulige udvidelser

I dette kapitel er det blevet beskrevet, hvorledes robotten Murphys hastighed kan reguleres, og hvordan den bringes til at stoppe op på et givent punkt på en given rute. Desuden blev det beskrevet, hvordan robottens forhjul kan styres, så robotten følger en given rute.

Ruteføgningen er implementeret simpelt og tager fx ikke højde for robottens orientering i forhold til kurvens tangent. I det hele taget fungerer ruteføgningen kun, hvis afstanden mellem robotten og ruten ikke er for stor, og hvis forskellen mellem robottens orientering og kurvens tangent i positionen p_s heller ikke er for stor. Der findes adskillige andre ruteføgningsalgoritmer, jf. [Balluchi et al., 1996] og [Luca et al., 1999, kapitel 4]. Disse er dog alle temmelig komplicerede. [Egerstedt et al., 1997] beskriver en robust og simpel metode. Denne blev afprøvet i [Rode and Kjeldsen, 2004], hvor det nok viste sig, at metoden er robust, til gengæld følges ruten ikke særlig nøjagtigt. Metoden ville dog være god at anvende i de tilfælde, hvor robotten kommer langt væk fra ruten.

Et problem ved ruteføgningen i afsnit 4.2 er, at PID-regulatoren afhænger af tiden. Hvis robotten holder stille eller kører meget langsomt, og $e > 0$, vil integralleddet blive større og større, og robotten vil dermed dreje skarpere og skarpere. Dette resulterer i ustabilitet i kørslen. Det ville være hensigtsmæssigt, hvis reguleringen tog højde for robottens hastighed.

Kapitel 5

Implementering

I dette kapitel vil programmets implementering kort blive beskrevet. Afsnit 5.1 giver en oversigt over de implementerede programbiblioteker, og afsnit 5.2 beskriver, hvorledes hovedprogrammet skulle have været implementeret. Afsnit 5.3 beskriver, hvilke andre programdele der mangler at blive implementeret.

5.1 Programbiblioteker

Af tekniske årsager blev "C" det valgte programmeringssprog. Alligevel er programbibliotekerne opbygget så objektorienteret som muligt for at lette brugen af dem. Fremgangsmåden er inspireret af måden, Gnome-bibliotekerne er opbygget på, jf. [Diverse, 2006]. En `struct` definerer et objekts data, og objektets metoder er almindelige funktioner, som har objektets navn som præfiks, og som tager en pointer til objektets `struct` som første argument. På denne måde er det let at gennemskue, hvilke metoder som hører til hvilke data.

I tabel 5.1 findes en oversigt over de implementerede programbiblioteker, en kort beskrivelse af "klasserne" i hvert bibliotek, samt referencer til de afsnit, hvor biblioteket anvendes. Programbibliotekerne ligger i mappen `src` på den vedlagte cd-rom, og i mappen `src/html` findes Doxygen-genereret dokumentation for programkoden. Doxygen er et dokumenteringssystem, jf. [van Heesch, 2006].

Tabel 5.1: Oversigt over programbibliotekerne.

Pakke	Beskrivelse	Afsnit
`2dgeometry`	`line`, `polygon`, `rectangle` og `vector` implementerer hhv. et linjesegment, en polygon, et rektangel og en vektor (eller et punkt) i planet. `position` implementerer en position i konfigurationsrummet $\mathbb{R}^2 \times [0; 2\pi[$.	2.2, 2.3
`datastructures`	`graph`, `heap` og `list` implementerer datastrukturerne hhv. en graf, en hop og en liste. `bbtree` implementerer et balanceret binært søgetræ implementeret som et AVL-træ. `rtree` og `stree` udvider `bbtree` og implementerer hhv. et range-træ og et segment-træ.	2.3.3, 2.5, 3.2.7
`landmark`	`camera` håndterer kommunikationen med kameraet, `image` og `image_io` implementerer diverse billedbehandlingsfunktioner. `landmark` implementerer landmærkegenkendelsen.	3.2
`motion_planning`	`map` og `map_read` håndterer kortet over forhindringerne. `rmp` og `rmp_io` implementerer Švestkas statistiske ruteplanlægning. `motion_planner` kan planlægge en rute mellem en given start- og målposition på et givet kort og giver resultatet både som tekst og som en postscriptfil.	2.5
`murphy`	`murphy` håndterer kommunikationen med Atmel-processoren, som styrer robottens elektronik, og definerer samtidig en række fysiske konstanter for Murphy.	-
`navigation`	`deadreckoning`, `routefollow` og `speedregulate` implementerer hhv. dead reckoning, ruteføgning og hastighedsregulering.	3.1, 4
`particle_filter`	`particle_filter` implementerer partikelfilteret, inkl. bevægelses- og observationsmodellen.	3.3
`psdoc`	`psdoc` håndterer genereringen af postscriptfiler, men tegning af diverse primitiver er uddelegeret til de respektive biblioteker. `position` har fx en funktion, som kan tegne en position. `psdoc` bruges hovedsaglig til fejlfinding og til generering af specialets figurer.	-

`route`	`segment` bruges af `segment_line` og `segment_arc` til at implementere klasserne hhv. S og C, d-vs. linjestykker hhv. cirkelbuer. `route` definerer en rute bestående af linjestykker og cirkelbuer. `routemanager` finder ruter inden for klasserne CCC, CSC og SCS, og finder den korteste rute inden for klasserne CCC og CSC.	2.1, 2.3, 2.5
`util`	`angle`, `object`, `random` og `timer` implementerer diverse hjælpefunktioner.	-
`avr/murphy`	Her implementeres den interruptstyrede kerne, som kører på Atmel-processoren, der kontrollerer styre-elektronikken.	A
`avr/speciale`	`slave` implementerer en proces, som kører på Atmel-processorens kerne, og som kommunikerer med XScale-processoren.	A

En del af bibliotekerne implementerer ganske basale funktioner som fx datastrukturer og 2d-geometri. Normalt vil det være mest hensigtsmæssigt at anvende eksisterende biblioteker i stedet for at implementere disse fra bunden. Dette har dog ikke været muligt, da programmet både skulle kunne oversættes og køres på en standard-pc og kunne krydsoversættes til at køre på den indlejrede XScale-processor, som styrer Murphy. Diverse open source biblioteker blev undersøgt, men problemer med krydsoversættelsen og afhængigheder mellem bibliotekerne forhindrede anvendelsen af disse. Intels IPP-bibliotek bliver dog anvendt, se [Corporation, 2004].

5.2 Hovedprogrammet

I dette afsnit vil det blive beskrevet, hvordan hovedprogrammet skulle have virket. Programmet er ikke blevet implementeret pga. det i indledningens afsnit 1.2.1 nævnte problem.

Programmet implementeres som et standard Linux kommandolinjeprogram. Programmet tager et kort, samt en start- og en målposition som inddata. Programmet beregner først en kollisionsfri rute fra start til mål og styrer dernæst robotten langs ruten til målpositionen. Programmet antager, at robotten er placeret på startpositionen ved programmets start. Pseudokode for det tænkte program kan ses i tabel 5.2.

NAVIGATE $[M,\ s,\ g]$

- Planlæg en rute fra start s til mål g vha. kortet M
- Initialiser $Bel(x_0)$ til en spids normalfordeling med top ved s
- Start højprioritetstråd, som kører dead reckoning, hastighedsregulering og ruteføgning
- Start lavprioritetstråd, som kører landmærkegenkendelse og Monte Carlo Lokalisering
- Afslut program, når målet g er nået

Tabel 5.2: Pseudokode for hovedprogrammet.

Grunden, til at landmærkegenkendelsen og Monte Carlo Lokaliseringen kører i en lavprioritetstråd, er, at landmærkegenkendelsen tager meget cpu-tid, men behøves ikke køres i streng sand tid, og Monte Carlo Lokaliseringen behøver kun at køre en opdatering af partikelfilteret, når et nyt billede er færdigbehandlet. Det er derimod nødvendigt at køre hastighedsreguleringen og ruteføgningen i sand tid for at holde robotten på ruten. Derfor skal disse køres i en højprioritetstråd. Se også figur 3.19.

En kørsel med robotten foretages som følger. Et kort over de omgivelser, robotten skal køre i, optegnes, og det bestemmes, hvor en række landmærker skal placeres. Denne information medtages på kortet, og landmærkerne placeres på deres respektive pladser. Robotten placeres i den ønskede startposition. Navigationsprogrammet startes på robotten ved at logge ind på Murphy over en trådløs ssh-forbindelse og indtaste en kommandolinje som fx:

```
> navigate kort.map 0 0 0 10 0 180
```

Dette vil få robotten til, vha. kortet i filen `kort.map`, at køre fra startpositionen $s = (0,0,0)$ til målpositionen $g = (10,0,180)$, hvor vinklerne er opgivet i grader.

5.3 Manglende implementering

I dette afsnit bliver det kort beskrevet, hvilke programdele der mangler at blive implementeret, før hovedprogrammet beskrevet i afsnit 5.2 kan komme til at fungere.

Hovedsaglig mangler selve hovedprogrammet samt den samlede løsning på posi-

tionsestimeringen beskrevet på flowdiagrammet på figur 3.19 at blive implementeret. Følgende programbiblioteker fungerer: ruteplanlægningen, dead reckoningen, landmærkegenkendelsen, bevægelses- og observationsmodellen, partikelfilteret, hastighedsreguleringen, stopproblemet og ruteføl gningen. Dog må det forventes, at diverse småting som finindstilling og forbedring af mindre uhensigtsmæssigheder i implementeringen af programbibliotekerne vil blive nødvendig, når hovedprogrammet implementeres og afprøves og derved synliggører disse.

Kapitel 6

Konklusion

Målet for dette speciale var at få den bil-agtige robot Murphy til, vha. et kort, at køre fra en startposition til en målposition uden at støde ind i forhindringer.

Teorien bag opgavens problemstilling er blevet gennemgået.

Kapitel 2 gennemgik ruteplanlægning først uden forhindringer. Det blev bestemt, om en given rute vil føre til kollision, og efter en diskussion af forskellige ruteplanlægningsalgoritmer blev den valgte løsning fremlagt, og det blev vist, at den implementerede løsning fungerer.

Kapitel 3 viste, hvordan dead reckoning implementeres, og løsningen blev afprøvet på Murphy. Hernæst blev en metode til landmærkegenkendelse både vist og afprøvet. Efter en diskussion af positionsestimeringsalgoritmer, blev Monte Carlo Lokaliseringen beskrevet. Her blev en bevægelsesmodel og en observationsmodel opstillet på grundlag af afprøvningen af dead reckoningen og landmærkegenkendelsen. Den samlede løsning blev dog ikke implementeret og derfor ikke afprøvet.

Kapitel 4 gennemgik styringen af robottens aktuatorer, hvilket vil sige regulering af robottens hastighed, samt en metode til at følge en give rute.

Kapitel 5 beskrev kort strukturen af det implementerede program.

Desværre er det ikke lykkedes at få det endelige mål opfyldt, pga. det i indledningens afsnit 1.2.1 nævnte problem. Alt i alt er specialet dog kommet meget tæt på målet. Jeg vil anslå, at en fuldt fungerende løsning vil kunne implementeres på mellem to og fire uger, hvis jeg ikke havde problemer med armene.

Litteratur

[rob, 2004] (2004). *DTU RoboCup: Robotternes tumleplads*. Danmarks Tekniske Universitet. http://www.robocup.dtu.dk/.

[Aarno et al., 2004] Aarno, D., Kragic, D., and Christensen, H. I. (2004). Artificial potential biased probabilistic roadmap method. *ICRA-04*, pages 461–469. http://www.nada.kth.se/~hic/hic-papers/arno-icra2004.pdf.

[Arulampalam et al., 2002] Arulampalam, S., Maskell, S., Gordon, N., and Clapp, T. (2002). A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188. http://citeseer.ist.psu.edu/article/arulampalam01tutorial.html.

[Atramentov and LaValle, 2002] Atramentov, A. and LaValle, S. M. (2002). Efficient nearest neighbor searching for motion planning. citeseer.ist.psu.edu/atramentov02efficient.html.

[Balkcom and Mason, 2002] Balkcom, D. J. and Mason, M. T. (2002). Time Optimal Trajectories for Bounded Velocity Differential Drive Vehicles. *The International Journal of Robotics Research*, 21(3):199–217. http://ijr.sagepub.com/cgi/content/abstract/21/3/199.

[Balluchi et al., 1996] Balluchi, A., Bicchi, A., Balestrino, A., and Casalino, G. (1996). Path tracking control for dubin's car. citeseer.ist.psu.edu/balluchi96path.html.

[Bentley and Ottmann, 1979] Bentley, J. L. and Ottmann, T. A. (1979). *Algorithms for reporting and counting geometric intersections*.

[Bidstrup et al., 2002] Bidstrup, E., Kjeldsen, R. F., and Raatz-Pedersen, P. (2002). *Konstruktion af robot til deltagelse i RoboCup*. http://reblag.dk/robot/overkill.pdf.

[Borenstein et al., 1996] Borenstein, J., Everett, H. R., and Feng, L. (1996). *"Where am I?" - Systems and Methods for Mobile Robot Positioning*. http://www-personal.engin.umich.edu/~johannb/position.htm.

[Bourke, 1987] Bourke, P. (1987). *Determining if a point lies on the interior of a polygon*. http://astronomy.swin.edu.au/~pbourke/geometry/insidepoly/.

[Carpenter et al., 1997] Carpenter, J., Clifford, P., and Fernhead, P. (1997). An improved particle filter for non-linear problems. http://citeseer.ist.psu.edu/carpenter04improved.html.

[Chan, 1994] Chan, T. M. (1994). *A Simple Trapezoid Sweep Algoritm for Reporting Red/Blue Segment Intersections*. http://www.cs.uwaterloo.ca/~tmchan/red_blue.ps.gz.

[Cohen et al., 1995] Cohen, J. D., Lin, M. C., Manocha, D., and Ponamgi, M. K. (1995). *I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments*. ftp://ftp.cs.unc.edu/pub/users/manocha/PAPERS/COLLISION/paper3dint.pdf.

[Committee, 2004] Committee, M. T. (2004). *RoboCup2004 Middle Size League - Rules and Regulations for 2004*. http://www.er.ams.eng.osaka-u.ac.jp/rc2004msl/index.cgi?page=Regulations+and+Rules.

[Corporation, 2006] Corporation, A. (2006). Atmel corporation. http://atmel.com/.

[Corporation, 2004] Corporation, I. (2004). *Intel Integrated Performance Primitives for Intel Architecture*. http://developer.intel.com.

[de Berg et al., 2000] de Berg, M., van Krevel, M., Overmars, M., and Schwarzkopf, O. (2000). *Computational Geometry - Algoritms and Applications*, second edition edition.

[Decker, 1989] Decker, R. (1989). *Datastructures*.

[Dissanayake et al., 2001] Dissanayake, G., Newman, P., Clark, S., Durrant-Whyte, H. F., and Csorba, M. (2001). *A Solution to the Simultaneous Localization and Map Building (SLAM) Problem*. http://robot.anu.edu.au/~david/slam2004/Basic%20Slam%20Paper.pdf.

[Diverse, 2006] Diverse (2006). Gnome. http://gnome.org/.

[Douc et al., 2005] Douc, R., Cappe, O., and Moulines, E. (2005). Comparison of resampling schemes for particle filtering. *ISPA 2005.PROCEEDINGS OF THE 4TH INTERNATIONAL SYMPOSIUM ON*, 2005:64. http://www.tsi.enst.fr/~cappe/papers/05ispa_dcm.pdf.

[Dubins, 1957] Dubins, L. E. (1957). *On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal position and tangents*.

[Egerstedt et al., 1997] Egerstedt, M., Hu, X., Rehbinder, H., and Stotsky, A. (1997). *Path Planning and Robust Tracking for a Car-Like Robot*.

[Fabricius-Bjerre, 1977] Fabricius-Bjerre, F. (1977). *Laerebog i geometri II - Differentialgeometri, kinematisk geometri*.

[Federation, 2005] Federation, T. R. (2005). *RoboCup*. http://www.robocup.org.

[Foley et al., 1996] Foley, van Dam, Feiner, and Hughes (1996). *Computer Graphics - Principles and Practice*.

[Fox et al., 1999] Fox, D., Burgard, W., and Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427. http://citeseer.ist.psu.edu/fox99markov.html.

[Fraichard and Scheuer, 2004] Fraichard, T. and Scheuer, A. (2004). From reeds and shepp's to continuous-curvature paths. *IEEE Trans. on Robotics*, 20(6):1025–1035. http://emotion.inrialpes.fr/bibemotion/2004/FS04.

[Gentle, 2003] Gentle, J. E. (2003). *Random Number Generation and Monte Carlo Methods*, second edition edition.

[Gieck, 1979] Gieck, K. (1979). *A Collection of Technical Formulae*. Gieck-Verlag.

[Gonzalez and Woods, 1993] Gonzalez, R. C. and Woods, R. E. (1993). *Digital Image Processing*.

[Gordon et al., 1993] Gordon, N., Salmond, D., and Smith, A. (1993). *Novel Approach to Non-linear/Non-Gaussian Bayesian State Estimation*.

[Gut, 2005] Gut, A. (2005). *Probability: A Graduate Course*. Springer Science+Business Media, Inc.

[Hu et al., 2004] Hu, W., Downs, T., Wyeth, G., Milford, M., and Prasser, D. (2004). *A Modified Particle Filter for Simultaneous Robot Localization and Landmark Tracking in an Indoor Environment*. http://www.araa.asn.au/acra/acra2004/papers/hu.pdf.

[Isard and Blake, 1998] Isard, M. and Blake, A. (1998). Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28. citeseer.ist.psu.edu/isard98condensation.html.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82:35–45. http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf.

[Kjeldsen, 2003] Kjeldsen, R. F. (2003). *Robotkonstruktion*. http://reblag.dk/robot/robotkonstruktion.pdf.

[Kjeldsen et al., 2001] Kjeldsen, R. F., Thulin, L. B., and Leopold, M. (2001). *Find the middle*. http://www.diku.dk/~leopold/.

[Kostov and Degtiariova-Kostova, 1995] Kostov, V. and Degtiariova-Kostova, E. (1995). *Some properties of clothoids*. INRIA.

[LaValle, 2004] LaValle, S. M. (2004). *Planning Algorithms*. http://msl.cs.uiuc.edu/planning.

[Liljedahl and Christensen, 1992] Liljedahl, E. and Christensen, O. K. (1992). *Grafalgoritmer og kompleksitet*, volume Bind 7. DIKU.

[Luca et al., 1999] Luca, A. D., Oriolo, G., and Samson, C. (1999). *Robot Motion Planning and Control*. http://www.laas.fr/~jpl/book.html.

[Lund, 2000a] Lund, J. (2000a). *Den store danske encyklopædi*.

[Lund, 2000b] Lund, J. (2000b). *Den store danske encyklopædi, Panama – Ranum*.

[Maybeck, 1979] Maybeck, P. S. (1979). *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press. http://www.cs.unc.edu/~welch/media/pdf/maybeck_ch1.pdf.

[McManis, 2003] McManis, C. (2003). *Servo-Motor 101*. http://www.repairfaq.org/filipg/RC/F_Servo101.html.

[Montemerlo and Thrun, 2003] Montemerlo, M. and Thrun, S. (2003). Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. citeseer.ist.psu.edu/586492.html.

[Nieuwenhuisen and Overmars, 2004] Nieuwenhuisen, D. and Overmars, M. H. (2004). Useful cycles in probabilistic roadmap graphs. http://www.give.nl/movie/publications/utrecht/cycles_icra2004.pdf.

[of Michigan, 1997] of Michigan, U. (1997). *PID Tutorial*. http://www.engin.umich.edu/group/ctm/PID/PID.html.

[Ole Jannerup, 2000] Ole Jannerup, P. H. S. (2000). *Introduktion til reguleringsteknik*.

[Olsen, 1995] Olsen, S. I. (1995). *Forelæsningsnoter til introduktion til billedbehandling*.

[PlatformX, 2006] PlatformX (2006). Platformx. http://platformx.sourceforge.net/.

[Procopiuc et al., 2002] Procopiuc, O., Agarwal, P., Arge, L., and Vitter, J. (2002). Bkd-tree: A dynamic scalable kd-tree. citeseer.ist.psu.edu/procopiuc02bkdtree.html.

[Reeds and Shepp, 1990] Reeds, J. and Shepp, R. (1990). *Optimal paths for a car that goes both forward and backwards*. http://www.dtc.umn.edu/~reedsj/car.ps.

[Rekleitis, 2003] Rekleitis, I. M. (2003). A particle filter tutorial for mobile robot localization. http://www.cim.mcgill.ca/~yiannis/ParticleTutorial.html.

[Rode and Kjeldsen, 2004] Rode, G. and Kjeldsen, R. F. (2004). *En simpel robotsimulator*. http://reblag.dk/robot/robotsim.pdf.

[Röfer and Jüngel, 2003] Röfer, T. and Jüngel, M. (2003). Vision-based fast and reactive monte-carlo localization. citeseer.ist.psu.edu/579510.html.

[Scheuer and Fraichard, 1997] Scheuer, A. and Fraichard, T. (1997). Continuous-curvature path planning for car-like vehicles. citeseer.csail.mit.edu/article/scheuer97continuouscurvature.html.

[Sim and Dudek, 1998] Sim, R. and Dudek, G. (1998). Mobile robot localization from learned landmarks. *Proc. IEEE/RSJ Conf. on Intelligent Robots and Systems (IROS)*. citeseer.ist.psu.edu/73261.html.

[Sunday, 2001] Sunday, D. (2001). *Intersections of Lines, Segments and Planes (2D and 3D)*. http://www.geometryalgorithms.com/Archive/algorithm_0104/algorithm_0104B.htm.

[Svestka, 1993] Svestka, P. (1993). *A Probabilistic Approach to Motion Planning for Car-like Robots*. http://archive.cs.uu.nl/pub/RUU/CS/techreps/CS-1993/1993-18.pdf.

[Thrun, 1998] Thrun, S. (1998). Bayesian landmark learning for mobile robot localization. *Machine Learning*, 33(1):41–76. http://robots.stanford.edu/papers/thrun.landmark.html.

[Thrun et al., 2000] Thrun, S., Fox, D., Burgard, W., and Dellaert, F. (2000). Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141. http://robots.stanford.edu/papers/thrun.robust-mcl.html.

[van Heesch, 2006] van Heesch, D. (2006). Doxygen. http://www.stack.nl/~dimitri/doxygen/.

[Vestergaard, 2004] Vestergaard, E. (2004). Vejgeometri. http://www.matematiksider.dk/vejgeometri.html.

[Zomorodian and Edelsbrunner, 2002] Zomorodian, A. and Edelsbrunner, H. (2002). *Fast Software for Box Intersections*. http://www.cs.duke.edu/~edels/GeoDS/Boxes.pdf.

Bilag A

Robotten Murphy

Dette appendiks beskriver karakteristika for robotten Murphy, som deltog i RoboCup-konkurrencen 2004 og 2005 på Danmarks Tekniske Universitet. Første afsnit omhandler dens udseende og hardware, andet afsnit består af en tabel over robottens fysiske dimensioner, og til sidst defineres robottens lokale koordinatsystem.

A.1 Hardware

Robotten er en firehjulet bil-agtig robot, dvs. den er forhjulsstyret. Chassiset er bygget i LEGO Technic, og hvert baghjul er drevet af hver sin LEGO-motor. På hvert baghjul sidder desuden et tachometer. Forhjulene bliver styret af en standard hobbyservomotor, for mere om servomotorer, se [McManis, 2003]. Robottens styreelektronik er selvbygget, og en Atmel AtMega32 indlejret processor håndterer kommunikationen med styreelektronikken, se [Corporation, 2006]. Atmel-processoren kører en interrupt-styret kerne udviklet til RoboCup 2004. Specialets programmel kører på et Stargate udviklerkit, se [PlatformX, 2006]. Udviklerkittet har omtrent samme dimensioner som et kreditkort og er 3 cm højt. Det indeholder en 400 mhz Intel XScale-processor, 64 Mb ram, bruger et 128 Mb Compact Flash-kort som "harddisk", et pc-card til trådløs kommunikation, flere serielporte og en usb-port. Udviklerkittet kører Linux som styresystem. Usb-porten bruges til web-kameraet og en serielport bruges til kommunikation med styreelektronikkens Atmel-processor.

Robottens øvrige udstyr inkluderer: tre IR-afstandssensorer, en syv-kanals stregsensor, en ultralydsafstandssensor, 80-tegns LCD-display, talesyntese, tastatur med fire taster, forlygter, bremselys, afviserblink samt diverse statuslysdioder.

A.2 Robottens dimensioner

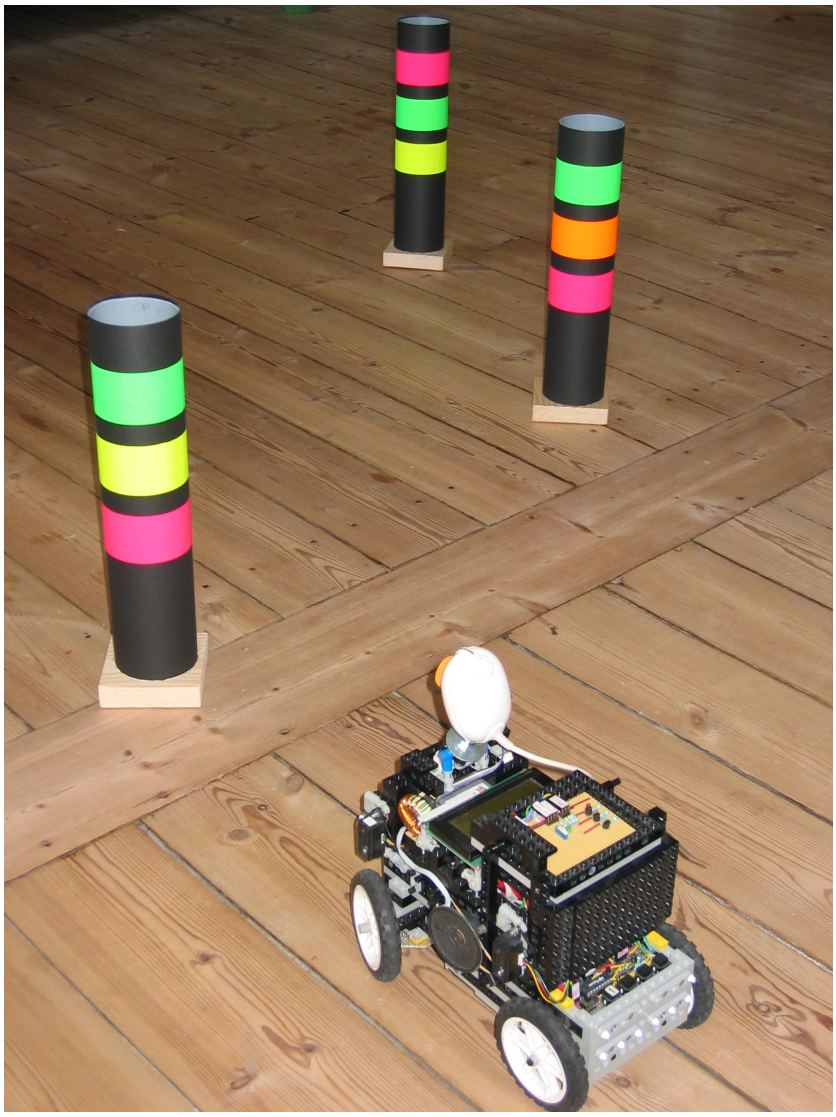
Enhed	Størrelse
Længde	260 mm
Bredde	170 mm
Hjulradius	40 mm
Længde mellem for- og bagaksel (L)	165 mm
Længde mellem de to baghjul (W)	125 mm
Tachometeropløsning	615 ticks pr. meter
Forhjulenes maksimale styrevinkel	ca. $\pm$ 0,54 radianer
Forhjulenes maksimale styrevinkelhastighed	ca. 2 radianer pr. sekund
Maksimal hastighed	Ukendt, men i omegnen af 1 m/s

Tabel A.1: Robottens Murphys dimensioner.

A.3 Robottens lokale koordinatsystem

Robottens lokale koordinatsystem er defineret som følger: x-aksen ligger parallelt med robottens længdeakse, fra bagenden mod forenden, y-aksen ligger vinkelret på x-aksen, så x- og y-aksen danner et højrehåndskoordinatsystem. Origo ligger midt på robottens bagaksel, på figur 2.1 med punktet R på figur 2.1. Vinkler regnes positive i positiv omløbsretning (dvs. mod uret), 0° ligger langs x-aksen.

For et billede af Murphy, se figur A.1.



Figur A.1: Murphy og tre landmærker.

Bilag B

Definitioner

I dette appendiks opsummeres og specificeres nogle definitioner, som de anvendes i dette speciale.

B.1 Punkt

Et *punkt* P defineres som tuplen bestående af de cartetiske koordinater (x, y) i planen.

B.2 Position

En bil-agtig robots *position* p defineres som triplen af bestående af de cartetiske koordinater (x, y) i planen samt robottens orientering θ, dvs. $p = (x, y, \theta)$.

B.3 Rotation

Operatoren $\mathrm{rot}(P, \alpha)$ defineres som rotationen af punktet P α radianer om origo, og operatoren $\mathrm{rot}(P, \alpha, C)$ defineres som rotationen af punktet p α radianer om punktet C.

B.4 Linjestykke

Et *linjestykke* l defineres som den rette linje mellem to punkter P_1 og P_2, og repræsenteres ved disse to punkter.

B.5 Polygon

En *polygon* $\mathcal{P}$ defineres som en lukket kurve i planen sammensat af linjestykker, som mødes i polygonens hjørner, jf. [Lund, 2000b, p. 366]. Disse linjestykker kaldes også polygonens *kanter*. $\mathcal{P}$ repræsenteres ved den ordnede liste $\{P_1, \ldots, P_{n+1}\}$ af dens n hjørner. I dette speciale vil det være underforstået at en polygon altid er *lukket*, dvs. at $P_{n+1} = P_1$, samt at en polygon altid er *simpel*, dvs. ingen af linjestykkerne af $\mathcal{P}$ krydser hinanden. Er antallet n af hjørner i $\mathcal{P}$ vigtigt for sammenhængen i teksten, nævnes det eksplicit som $\mathcal{P}^n$. Med den i'te kant af $\mathcal{P}$ menes linjestykket fra P_i til P_{i+1}, $i \leq n$.

B.6 Akseparallelt omgivende rektangel

Det *akseparallelle omgivne rektangel*, forkortet AABB, for en polygon $\mathcal{P}$ repræsenteret ved punkterne $P_i = (x_i, y_i)$ defineres som polygonet bestående af punkterne $P'_1 = (x'_1, y'_1)$, $P'_2 = (x'_2, y'_1)$, $P'_3 = (x'_2, y'_2)$ og $P'_4 = (x'_1, y'_2)$, hvor $x'_1 = \min\{x_i\}$, $x'_2 = \max\{x_i\}$, $y'_1 = \min\{y_i\}$ og $y'_2 = \max\{y_i\}$.

Bilag C

Fysisk udformning af landmærkerne

Landmærkerne er konstrueret af et 335 mm langt stykke plastiknedløbsrør med en diameter på 75 mm. Røret er limet på en $90 \times 90 \times 21$ mm træklods. Under træklodsen er monteret fire stk. 1 mm høje gummidutter, som gør landmærket stabilt og skridsikkert. Røret er påklistret sort karton, som sikrer en mat sort overflade. De tre 45 mm høje farvede bånd, placeret med 20 mm mellemrum i mellem, udgøres af "fluorescerende" papir udvalgt af de fire farver pink, orange, gul og grøn. Dette er nemlig de fire farver, det fluorescerende papir kan købes i. For et billede af tre af landmærkerne, se figur A.1. Det fluorescerende i papiret består af, at i dagslys (indeholdende ultraviolet stråling) fremstår farverne meget intenst, og dermed letgenkendeligt. Den ultraviolette stråling omdannes af papiret til synligt lys. Derfor blev netop dette papir valgt.

Bilag D

Robotnavigation - Arbejdsbeskrivelse

En bil-agtig robot skal køre fra en start position til en slut position uden at støde ind i forhindringer. Ved en position forstås såvel sted som retning. Robotten har i forvejen et kort over forhindringerne og over positionen af visse landmarks. For at navigere skal robotten løbende estimere sin position og orientering. Hertil vil jeg anvende dead-reckoning sammen med måling af retning og afstand til landmarks. Dead-reckoning giver akkumulerende fejl (ca 5% pr. kørte meter), men ved at sammenholde med målinger af landmarks kan man for eksempel ved Kalman-filtrering forbedre sit estimat af sin position.

En særlig vanskelighed er at robotten er ikke-holonom, hvilket medfører at robottens rute skal planlægges som sammensætning af segmenter af cirkelbuer og liniestykker. Et tidligere projekt har løst problemet i en simulator, men uden forhindringer, og kun for fremadrettet kørsel.

Til ruteplanlægningen regner jeg med at tage udgangspunkt i den vedhæftede artikel [Svestka, 1993]. I specialet vil jeg diskutere de forskellige metoder til ruteplanlægning for en bil- agtig robot i forhindringsfyldte omgivelser.

Jeg regner med at bruge min egen robot (den fra Robocup 2004), udvidet med et Xscale (arm) 400mhz processorboard, et webcam og forhåbentligt et trådløst netværkskort.