# Konstruktion af robot til deltagelse i RoboCup

Erik Bidstrup
bidstrup@diku.dk

Rasmus Friis Kjeldsen
razzle@diku.dk

Peder Raatz-Pedersen
yoda@diku.dk

10. november 2002

# Indhold

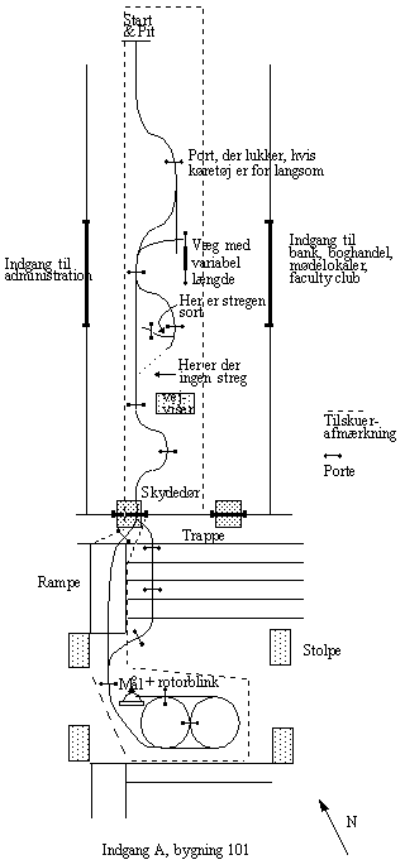
# Kapitel 1

# Indledning

## 1.1 Om RoboCup

På DTU afholdes årligt konkurrencen RoboCup. RoboCup er et orienteringsløb for autonome robotter. Banen består grundlæggende af 18 porte. Den robot, der kører igennem flest porte på kortest tid, vinder. Portene kan findes ved at følge en streg lavet med hvid tape på gulvet. Nogle steder er der særlige forhindringer: tapen skifter farve fra hvid til sort, tapen bliver erstattet med en væg, som robotten skal køre langs med, eller tapen forsvinder og robotten skal køre lige ud, endelig går ruten ned ad en trappe eller rampe.

Figur 1.1 viser det kort over banen, vi har brugt, som udgangspunkt ved design af robotten. Banen viste sig at være stort set identisk ved RoboCup 2002. Konkurrencen afholdes på ujævnt underlag bestående af fliser med store fuger imellem. Det udelukker, at nogen af DIKUs eksisterende robotter kan anvendes, og er den vigtigste motivation for at bygge en robot op fra grunden. Derudover er der forhindringer på banen i form af rampe og trappe, som stiller krav til robottens styrke.




Figur 1.1: Banens layout ved RoboCup 2001. (Figuren er kopieret fra [2].)

## 1.2 Overordnet strategi

Historisk har alle deltagere i RoboCup baseret deres robotter på såkaldte *linjesensorer*. Betegnelsen linjesensor dækker typisk over to eller flere lys-/fotodiode-par, som registrerer gulvets farve di-

rekte under eller foran bilen. Ud fra gulvets farve estimeres om fotodioden er over stregen.

Vores mål er at gennemføre banen udelukkende ved hjælp af input fra et fremadrettet kamera. Der er følgende åbenlyse fordele ved at anvende et kamera:

- Det er muligt at se frem på banen og derved opnå mere intelligent styring, der tager højde for banens forløb.
- Robotten bliver robust over for mindre defekter — f.eks. fodaftryk — på stregen, da den kan se et stort stykke af stregen.
- Hardwaren er ikke bygget specifikt til streggenkendelse, hvilket giver større fleksibilitet med hensyn til andre anvendelsesmuligheder.

Brug af kamera giver da også anledning til en del udfordringer, hvoraf kan nævnes:

- Billedbehandling kræver meget regnekraft, hvilket bevirker, at robotten bliver stor og tung og derved vanskelig at navigere.
- På grund af krav om vægt og størrelse af robotten er det ikke muligt at montere kraftigt lys på robotten. Den er derfor afhængig af lyset fra omgivelserne, hvilket netop under RoboCup er et stort problem.

## 1.3 Arbejdsfordeling

En af os, Rasmus Friis Kjelsen, fik dispensation til at skrive denne opgave sammen med de to andre kandidatstuderende, som lavede projektet som andendelsrapportopgave. Betingelsen for dette var dog, at det fremgår af rapporten, hvilken andel Rasmus har haft i arbejdet.

Dette er naturligvis svært at præcisere helt nøjagtigt, da vi i høj grad har samarbejdet om hele opgaven. Her nævnes derfor de dele af opgaven, som Rasmus har haft størst andel i: Afsnit 3.2 til og med afsnit 3.5, afsnit 6 og afsnit 7.

# Kapitel 2

# Hardware

## 2.1 Kravspecifikation

Udgangspunktet for vort design har været, at alle forhindringer på banen skal kunne gennemføres. Det betyder, at robotten skal være forholdsvis stabil og have gode køreegenskaber, da den dels skal køre på ujævnt underlag og dels skal kunne køre ned ad trapper og ramper. Et ufravigeligt krav er, at den skal være lille nok til at kunne passere portene.

Valget at bruge en komplet laptop som udgangspunkt for robotten frem for f.eks. en mindre palmtop har haft stor indflydelse på resten af valget af komponenter. Vægten og størrelsen af den bærbare computer sætter store krav til bilens mekaniske konstruktion. Derudover er den på grund af sin harddisk relativt følsom over for stød og slag, hvorfor det er nødvendigt med god mekanisk affjedring. Til gengæld har vi kunnet bruge standard pc-dele hvilket har givet adgang til et stort udvalg af bl.a. kameraer.

## 2.2 Komponentbeskrivelse

Til konstruktion af bilen har vi brugt følgende komponenter, de er alle valgt, som det billigste vi har kunne finde der formodes at være "godt nok" til formålet.

- Sony Vaio Z-600:
  700 MHz Pentium III. Er valgt på grund af sin relativt lave vægt (ca. 2 kg) og lille størrelse.
- Tamaiya "Mad Bull":
  Fjernstyret bil der bruges som robottens basiskonstruktion. (Figur 2.1)
- 7.2V DC motor fra boremaskine:
  Bruges til fremdrift af robotten. Motoren viste sig at have marginalt bedre egenskaber, end den som blev leveret med den fjenstyrede bil.
- Servo:
  Bruges til at kontrollere styretøjet. Servoen er efterfølgende blevet udstyret med et bedre kugleleje.
- Philips TuCam Pro:
  Webcam fra Philips dette webkamera udskiller sig ved at være udstyret med CCD-chip i stedet for CMOS. Kameraet er valgt fordi det er relativt lysfølsomt og dermed kan tage billeder uden al for meget bevægelsesløring selv under svagt lys. Desværre har kameraets optik en relativt stor brændvidde (tele), hvilket er et problem da bilen som resultat får tunnelsyn. (Figur 2.3)
- Teak Babymouse:
  Indmaden bruges som takometer. Musen er valgt på grund af sin lave pris og ringe størrelse.
- MacGyver servo- og motorstyring:
  Styreelektronik der er specialbygget til robotten. Motorstyringen udemærker sig ved, at den kan give meget høj strøm ($\sim$ 30A) til motoren. Det har vist sig at være nødvendigt

for at trække bilen med udstyr. MacGyver-styringen har endvidere interface til 3 infrarøde afstandsmålere, som vi dog ikke bruger. MacGyverkortet kommunikerer med pc-en via et serielinterface. (Figur 2.4)

- Kraftige fjedre:
  Stivere fjedre end de der blev leveret med bilen forhindrer konstruktionen i at gå knæ ved fuld vægtbelastning. (Figur 2.2)
- Zinkplade (22×27 cm):
  Pladen er monteret oven på den fjernstyrede bil og understøtter den bærbare computer og holder kameratårnet og MacGyverkortene. (Figur 2.4 og 2.3)
- Skumgummi:
  Er dels sat fast oven på zinkpladen for at absorbere stød for den bærbare computer. Dels er der fyldt skumgummi i bilens dæk for gøre dem hårdere og derved nedsætte rullemodstanden og forbedre styreegenskaberne. (Figur 2.5)
- 1/2l Faxe Kondi flaske.:
  Er brugt som kameratårn fordi den er let og stiv. (Figur 2.3)
- Strips
- Tape:
  Holder sammen på de løse dele der ellers ikke umiddelbart lader sig fæstne ordentligt.
- 2 blyakkumulatorer 12V, 1.2Ah:
  Leverer strøm til motor og MacGyver-elektronikken. Den bærbare computer får naturligvis strøm fra sit eget batteri.

## 2.3 Konstruktion af robottens mekanik

Udgangpunktet for robotten er, som nævnt, den fjernstyrede bil. Bilen vi købte bestod af fire hjul, hjulophæng, affjedring, styretøj, differentiale og motor. Mange af disse har det efterfølgende vist sig at være nødvendigt at udskifte eller forbedre, da bilen ikke var designet til at køre med den ekstra vægt fra laptoppen.

I dette afsnit vil vi kortfattet beskrive nogle af de vigtigste ændringer vi har lavet.

### 2.3.1 Affjedring

Det viste sig hurtigt at bilens affjedring var alt for blød. Fjedrene blev presset i bund under påvirkningen af ekstraudstyrets vægt. Først forsøgte vi at gøre fjedringen hårdere ved at komprimere fjedrene med 1-kroner, som det kan anes på figur 2.1.

Da det sidste udstyr blev monteret, var fjedrene alligevel presset i bund, hvorfor vi så os nødsaget til at købe nogle kraftigere fjedre. Disse kunne monteres uden yderligere problemer efter tilslibning af fjederholderne (figur 2.2).

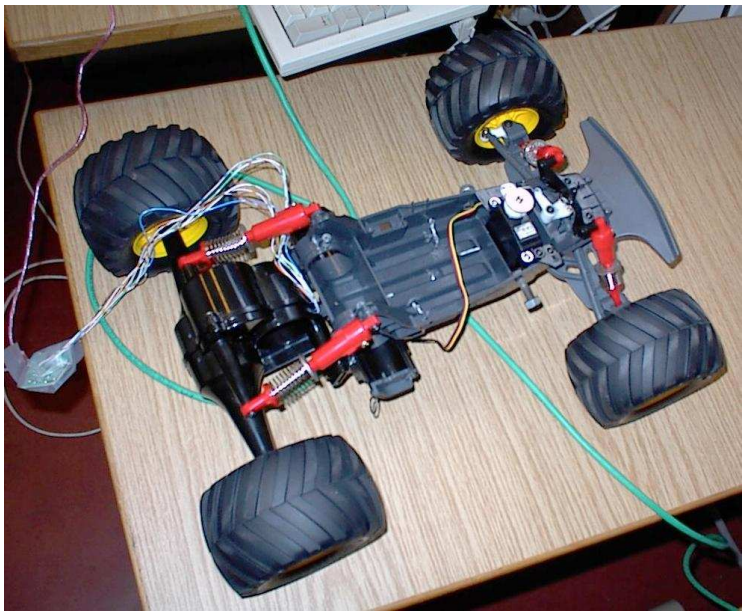
Selv efter udskiftningen af fjedrene lå robotten lavt. Dette skyldes at plastikken som chassiet var støbt af, begyndte at give efter. Vi har forsøgt at opstramme det med nogle strips dog uden fuldt tilfredstillende resultat.

### 2.3.2 Takometre

Da der sidder to takometre i en standard pc-mus, der kan sluttes direkte til laptoppen, var det oplagt at bruge disse til at måle bilens kørte afstand. Vi overvejede også at bruge en optisk mus til målingerne, men vi forkastede idéen, da vi ikke var sikre på, hvordan den skulle monteres på bilen, og hvordan den ville fungere med skiftende underlag.

Takometrene i den valgte mus består, meget simplificeret, af en skive med huller (hulskive), en lysdiode og en fotodiode. Når hulskiven roterer, vil lysstrålen mellem lysdioden og fotodioden blive periodisk brudt. Musens elektronik vil omsætte signalet fra fotodioden til en vinkel og retning.

Ved hjælp at en boremaskine og en fil blev hulskiverne modificeret, så de kunne monteres på



Figur 2.1: Bilen hvor alt er afmonteret undtagen styreservo og takometrene. Yderst til venstre i billedet ses resten af musens elektronik pakket ind i tape. Fjedrene er spændt ved hjælp af 1-kroner.

bilens baghjulsaksel på hver side af differentialet.

Lys- og fotodioderne var loddet fast på musens styreprint. Dette blev savet over, så printene med de to lys-/fotodiode-par (optogafler) blev separeret fra styreelektronikken. De printbaner som blev skåret over, er blevet reetableret med lange ledninger. De to optogafler blev monteret med en klat lim i differentialehuset sammen med hulskiverne, og ledningerne blev ført ud til styreelektronikken.

Konstruktionen har vist sig ikke at være helt problemfri. Den ene optogaffel har løsnet sig, hvorfor der ikke kan måles afstand på det ene hjul. Derudover er styreelektronikken, hvor printet er blevet delt, følsomt over for elektromagnetisk støj. Vi kunne observere, at musen holdt op med at sende afstandsinformation adskillige sekunder, hvis motoren, som udsender kraftig elektromagnetisk støj, satte hårdt igang. Dette er beskrevet yderligere i afsnit 5.1.

### 2.3.3 Zinkskjoldet

Zinkskjoldet er, som navnet antyder, skåret ud af en zinkplade med nogenlunde samme dimensioner som laptoppen. Der er med en pladesaks blevet klippet en slids i hver side, som er blevet bukket ned og skruet fast til bilen. Resten af pladen hviler på bilen. (Se figur 2.3.) Yderligere er zinkpladen blevet falset langs randen for at øge stivheden.

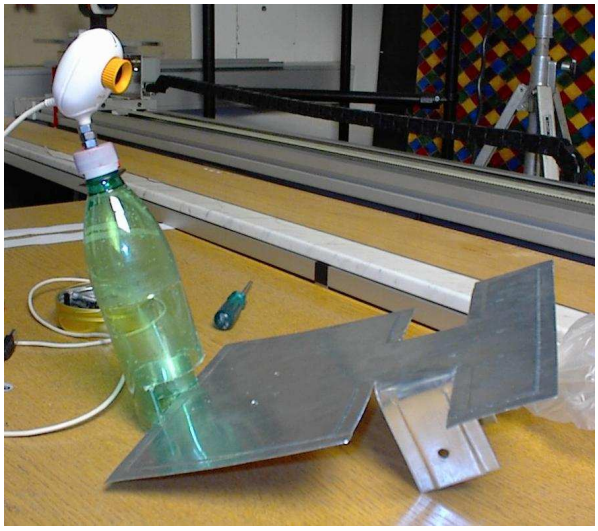
På undersiden af zinkskjoldet er MacGyverkortene blevet monteret med afstandskruer. (Se figur 2.4.)

### 2.3.4 Dæk

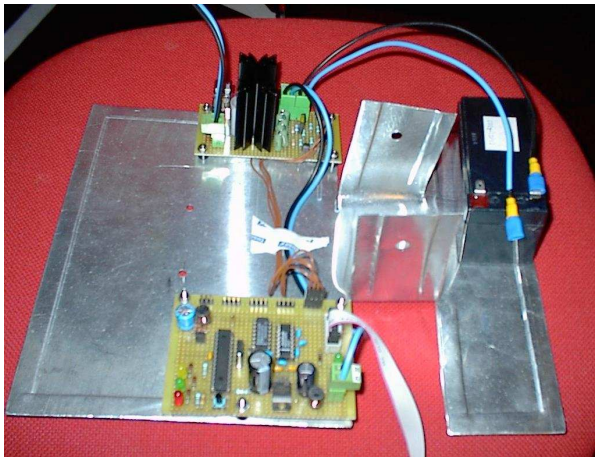
Da alle komponenter var blevet monteret på robotten, kunne den ikke køre. Det viste sig, at bilens dæk, under den ekstra vægt, blev komprimeret så meget at rullemodstanden var for høj.

Optimalt skulle vi have monteret nogle andre hjul

Figur 2.2: Robottens fjedre udskiftes med nogle betydeligt hårdere for at undgå at bilen ligger for lavt, hvilket går ud over bilens styreegenskaber. På billedet ses to af de oprindelige fjedre og en af de hårdere fjedre. De røde plastikstykker udgør fjederholderen.



Figur 2.3: Zinkskjoldet som bærer al elektronikken, og kameraet monteret på kameratårnet.



Figur 2.4: To af de tre printplader som udgør MacGyver-controleren monteret på undersiden af zinkskjoldet. Til højre i billedet ses en blyakkumulator.

Figur 2.5: Hulrummet i robottens dæk fyldes med skumgummi for at gøre dækkene hårdere og dermed reducere rullemodstanden.

på robotten, men da vi var under tidpres, løste vi i stedet problemet ved at afstive dækkene. Vi klippede en tyk skumgummimåtte ud i strimler svarende til dækkets bredde. Strimlerne rullede vi sammen og puttede ind i dækkene. (Se figur 2.5.)

Et andet problem med hjulene er, at de på grund af deres store kontaktflade med underlaget, ikke kan drejes af styreservoen når bilen holder stille. Det er et problem, da den optimale styrestrategi (afs. 4.4) forudsætter at hjulene kan skifte retning i et punkt. Løsningen er at montere hårde smalle hjul på bilen. Smallere hjul vil også bevirke at bilen bliver smallere hvilket yderligere er en fordel.

### 2.3.5 Den færdige robot

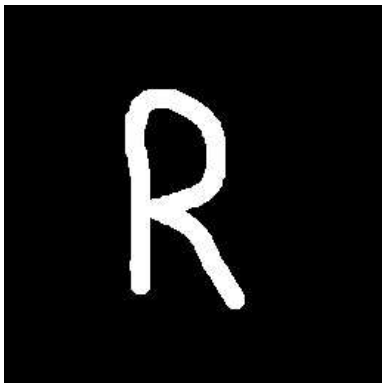
På figur 2.6 ses et billede af robotten taget under konkurrencen. Her er alle dele inklusive laptoppen samlet.

Figur 2.6: Robotten som den så ud ved starten af anden finalerunde. (Billedet er kopieret fra [2])

# Kapitel 3

# Billedbehandling

For at finde tapestregen, i de billeder vores robot optager (se eks. i figur 3.4), skal vi bestemme, hvilken type feature vi vil identificere stregen ved. Da stregen ikke er en tynd streg og dermed ikke vil fremstå som en enkelt streg i et billede af diskontinuiteterne i intensitetsbilledet, vil brug af et kantdetektionsfilter kræve en hel del videre processering for at kunne identificere, hvilke to streger i diskontinuitetsbilledet, der svarer til højre og venstre side af tapen. Dette vil yderligere blive besværliggjort, da RoboCup-banen bliver lavet oven på et område, hvor der vil være en hel del diskontinuiteter bl.a. pga. af flisernes ujævnheder og fugerne mellem flisерne.

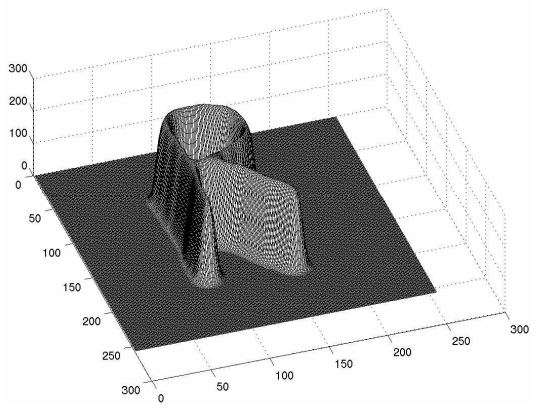


Figur 3.1: Billede af et R

## 3.1 Ridgedetektion

Vi har derfor valgt at lede efter højderygge (ridges) i intensitetsbilledet, kraftigt inspireret af [6]. Det kræver bl.a., at vi har en skalarumsrepræsentation, med en passende fast skala, af de billeder som robotten optager.

En illustration af højderygge kan ses i figur 3.1 og 3.2, der henholdsvis viser et billede af et R samt et plot, hvor højderyggene fra intensitetsbilledet fremgår.



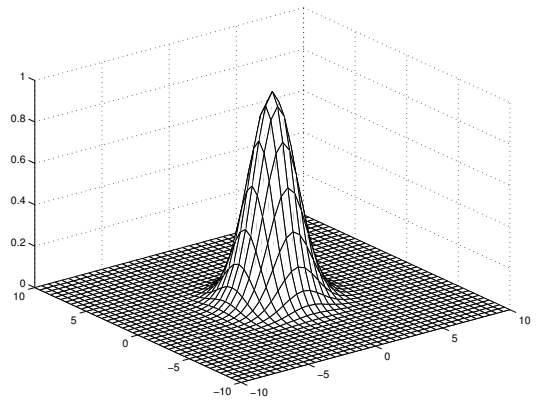

Figur 3.2: Plot af et R, hvor en højderyg tydeligt ses

### 3.1.1 Skalarumsrepræsentation

Når man ønsker at detektere strukturer i billeder (f.eks. kanter), støder man ofte på det problem, at de søgte strukturer kun kan detekteres over et begrænset område af opløsninger. Det kan derfor være nyttigt at bruge en mekanisme, der gradvist kan fjerne mere og mere af den mest højfrekvente information i et billede. Denne mekanisme skal være skalerbar, for at den kan give billedet på forskellige skalaniveauer. Dermed vil de resulterende billeder indeholde forskellig information om deres strukturer. Hvert af disse billeder er knyttet til en skalaparameter, der beskriver, hvilken skala billedet er frembragt af.

Uden at dykke for langt ned i skalarumsteori er det, vi ønsker, en foldningskerne, der ved foldning med et billede, kan give en skalarumsrepræsentation af billedet. Samtidig skal en højere skalaparameter resultere i grovere skalaniveauer. I takt med at skalaniveauet stiger, skal billedet udglattes og dermed indeholde mindre information, end det gør på et lavere skalaniveau. Ifølge [6] kan det vises, at netop en Gausskerne kan opfylde disse krav. Ved at bruge en Gausskerne til foldningen kan man så vælge at bruge spredingen $\sigma$ som skalaparameter.

I [6], som vi har brugt som grundlag for vores ridgedetektion, argumenteres der for, at det er nødvendigt at have en mekanisme, der udover at kunne lave en skalarumsrepræsentation desuden kan vælge skalaparameteren dynamisk. Dermed vil en feature i billedet undersøges på den bedst egnede skala. Denne automatiske skalaselektion er dog mere, end vi har brug for, da det vi ønsker er at finde en tapestreg, der vil have nogenlunde samme udseende og størrelse, og derfor skala, i billedfeltet over hele løbsdistancen. Vi kan dermed nøjes med eksperimentelt at bestemme en spredning (skalaparameter), hvilket gør os i stand til effektivt at detektere tapen. Hvordan vi gør dette, er beskrevet i afsnit 3.4.




Figur 3.3: Plot af 2D Gauss-funktion

### 3.1.2 Gaussfiltrering

Vi betragter et 2-dimensionelt Gaussfilter givet ved:

$$g[i,j] = e^{-\frac{i^2+j^2}{2\sigma^2}}$$

Et plot af en todimensionel Gaussfunktion kan ses i figur 3.3.

I plottet ses det intuitivt, hvordan Gaussfilteret fungerer som en udglatter. Der bliver lagt størst vægt på punktet i midten, og der bliver så lagt mindre og mindre vægt på nabopunkterne, jo længere væk de er. Det er værd at bemærke, at funktionen aftager lige meget i alle retninger og dermed opnår en symmetri, der samtidig gør, at der bliver udglattet lige meget i alle retninger. Filterets spredning (angivet ved $\sigma$) bestemmer, hvor bredt det bliver og dermed også, hvor stor en udjævningseffekt det har. Jo større $\sigma$ bliver, jo bredere bliver filteret, og jo flere punkter vil få en mærkbar effekt på det filtrerede billede.

Når $\sigma$ vokser, vil der dog ske det, at størrelsen på filteret — og dermed køretiden af programmet — vokser med $O(n^2)$, da filteret er kvadratisk. Det todimensionelle Gaussfilter har dog den egenskab, at det kan skilles i to endimensionelle filtre. Det

ses ved:

$$g[i,j] * f[i,j] = \sum_{k=1}^{m}\sum_{l=1}^{n} g[k,l]f[i-k,j-l]$$
$$= \sum_{k=1}^{m}\sum_{l=1}^{n} e^{-\frac{k^2+l^2}{2\sigma^2}} f[i-k,j-l]$$
$$= \sum_{k=1}^{m} e^{-\frac{k^2}{2\sigma^2}} \left\{ \sum_{l=1}^{n} e^{-\frac{l^2}{2\sigma^2}} f[i-k,j-l] \right\}$$

Dermed kan man altså folde med et endimensionalt Gaussfilter først vandret og siden lodret. Om man starter med at folde vandret eller lodret er ikke vigtigt, da foldning både er kommutativ og associativ. Filteret, der foldes med, er det samme, dog skal det transponeres mellem foldningerne.

Alt i alt betyder det, at når $\sigma$ vokser, vil filterets køretid kun vokse med $O(n)$.

### 3.1.3 Konstruktion af Gaussfilter

Når vi skal konstruere et Gaussfilter, kan vi altså nøjes med at konstruere et endimensionelt filter, hvor vægtene beregnes direkte ud fra:

$$g(x) = e^{-\frac{x^2}{2\sigma^2}}$$

Hvis filtervægtene ønskes som heltal, kan det gøres ved f.eks. at dividere den midterste vægt i filteret (den største) med den yderste (den mindste) og derefter skalere alle vægtene med resultatet. Efter foldningen med filteret skal man huske at normalisere outputtet med summen af vægtene. Hvis $g$ er Gaussfilteret gående fra -5 til 5, $f$ er billedet og $h$ er outputtet, gælder der efter skalering af filteret:

$$\sum_{g=-5}^{5} g[x] = y$$

og

$$h[i,j] = \frac{1}{y}(f[i,j] * g[i,j])$$

hvor $(f[i,j] * g[i,j])$ skal forstås som billedet foldet lodret og vandret med Gaussfilteret.

Værdien af vægtene i Gaussfilteret vil aftage og gå mod nul, jo længere man kommer væk fra centret i Gaussfilteret. Bredden af Gaussfilteret kan sættes efter, hvornår man synes, vægtene er så små, at de ikke længere vil have den store indflydelse. Det vil sige, at $\sigma$ skal tages med i beregningerne, når man i praksis vil bestemme, hvor bredt et givet filter skal være. En mulighed kunne være at:

$$Bredde = 1 + 2\lceil c\sigma \rceil$$

Hvor $c$ vælges, så de vigtigste vægte kommer med. Der multipliceres med 2, så samme bredde bliver lagt til på begge sider af Gaussfilterets centrum, og der lægges 1 til for centret.

### 3.1.4 Ridgefilter

Vi definerer først skalarumsrepræsentationen $L(\sigma)$ af billedet $f$ som værende

$$L(\sigma) = g(\sigma) \star f$$

hvor $\sigma$ er skalaen samt spredningen i Gaussfilteret $g$ og $\star$ er foldningsoperatoren. De skalarumsafledte kan så defineres som

$$L_{x^\alpha y^\beta}(\sigma) = \partial_{x^\alpha y^\beta} L(\sigma)$$

hvor $\alpha$ og $\beta$ angiver, i hvilken rækkefølge der differentieres. Dermed bliver f.eks. billedet med de førsteafledte i x-retningen $L_x$, mens billedet med de andenafledte i x-retningen bliver $L_{xx}$.

Vi indfører nu, for hvert punkt $(x,y)$, et lokalt koordinatsystem $(p,q)$ som ligger "linet" op med retningen af den største krumning i intensitetsbilledet. D.v.s. at den ene akse i $(p,q)$-systemet peger langs den største krumning og den anden akse er ortogonal på. For at udtrykke de afledte i dette koordinatsystem, kan vi rotere med en vinkel $\beta$ givet ved (ifølge [6]):

$$\cos\beta = \sqrt{\frac{1}{2}\left(1 + \frac{L_{xx} - L_{yy}}{\sqrt{(L_{xx} - L_{yy})^2 + 4L_{xy}^2}}\right)}$$

$$\sin\beta = \alpha\sqrt{\frac{1}{2}\left(1 - \frac{L_{xx} - L_{yy}}{\sqrt{(L_{xx} - L_{yy})^2 + 4L_{xy}^2}}\right)}$$

Her er $\alpha$ lig med fortegnet af $L_{xy}$. Enhedsvektorene i $(p, q)$-systemet bliver

$$\begin{aligned} e_p &= (\sin\beta, -\cos\beta) \\ e_q &= (\cos\beta, \sin\beta) \end{aligned}$$

og de afledte i retningerne $p$ og $q$ bliver

$$\begin{aligned} \partial_p &= \sin\beta\partial_x - \cos\beta\partial_y \\ \partial_q &= \cos\beta\partial_x + \sin\beta\partial_y \end{aligned}$$

Dermed kan vi udregne $L_p$,$L_q$,$L_{pp}$ og $L_{qq}$ som

$$\begin{aligned} L_p &= \partial_p L \\ &= (\sin\beta\partial_x - \cos\beta\partial_y)L \\ &= \sin\beta\mathrm{L}_x - \cos\beta\mathrm{L}_y \end{aligned}$$

$$\begin{aligned} L_q &= \partial_q L \\ &= (\cos\beta\partial_x + \sin\beta\partial_y)L \\ &= \cos\beta\mathrm{L}_x + \cos\beta\mathrm{L}_y \end{aligned}$$

$$\begin{aligned} L_{pp} &= \partial_p\partial_p L \\ &= (\sin\beta\partial_x - \cos\beta\partial_y)^2 L \\ &= ((\sin\beta\partial_x)^2 + (\cos\beta\partial_y)^2 \\ &\quad -2\sin\beta\partial_x\cos\beta\partial_y)L \\ &= (\sin\beta)^2 L_{xx} + (\cos\beta)^2 L_{yy} \\ &\quad -2\sin\beta\cos\beta L_{xy} \end{aligned}$$

$$\begin{aligned} L_{qq} &= \partial_q\partial_q L \\ &= (\cos\beta\partial_x + \sin\beta\partial_y)^2 L \\ &= ((\cos\beta\partial_x)^2 + (\sin\beta\partial_y)^2 \\ &\quad +2\cos\beta\partial_x\sin\beta\partial_y)L \\ &= (\cos\beta)^2 L_{xx} + (\sin\beta)^2 L_{yy} \\ &\quad +2\cos\beta\sin\beta L_{xy} \end{aligned}$$

Her kan $L_x$, $L_y$, $L_{xx}$, $L_{yy}$ og $L_{xy}$ tilnærmes ved at folde intensitetsbilledet med f.eks. et vertikalt og/eller et horisontalt sobelfilter et passende antal gange. F.eks. foldes to gange med et horisontalt sobelfilter for at finde $L_{yy}$.

Ridgepunkterne kan så defineres som de punkter, der i intensitetsbilledet antager et lokalt maksimum i retningen af den største krumning. Det kan udtrykkes ved at den førsteafledte skal være tæt på 0

$$\begin{aligned} L_p &> minD, \\ L_p &< maxD, \end{aligned}$$

og den andenafledte skal være stor ortogonalt på ridgen

$$L_{pp} < -Minimum\ krumning,$$

men lille parallelt med ridgen

$$|L_{pp}| \geq |L_{qq}|$$

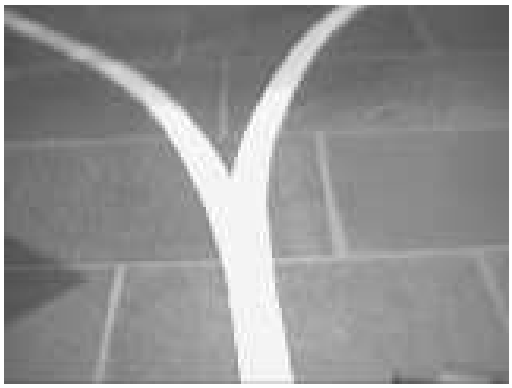
for hvide ridgepunkter og modsat

$$\begin{aligned} L_q &> minD, \\ L_q &< maxD, \\ L_{qq} &> Minimum\ krumning, \\ |L_{qq}| &\geq |L_{pp}| \end{aligned}$$

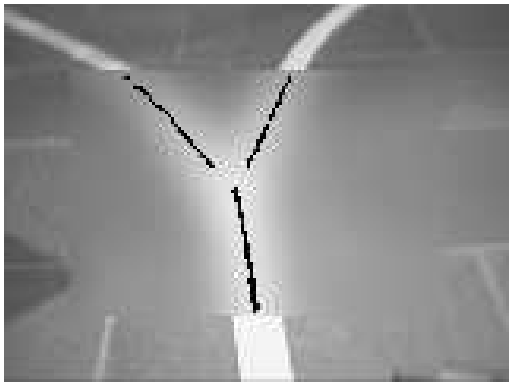
for sorte ridge punkter. Der skal bemærkes at det lokale koordinatsystem akser er ombyttet ved sorte ridgepunkter, det skyldes at fortegnene skiter i (3.1) og (3.1), men har ikke nogen yderligere konsekvenser.

Konstanterne $minD$, $maxD$ og $Minimum\ krumning$ kan findes eksperimentielt, dog skal $minD$ og $maxD$ ideelt ligge i nærheden af 0.

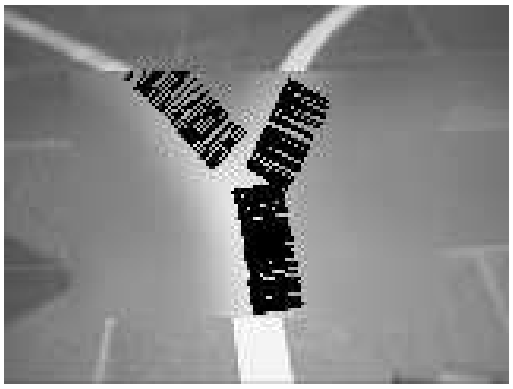
Se figur 3.4, figur 3.5 og figur 3.6 for en illustration af ridgefilterets virkemåde.



Figur 3.4: Originalbillede, som er taget af robotten under første finalerunde i RoboCup 2002. Det viser banens første forgrening.



Figur 3.5: Det Gaussfiltrerede billede lagt oven i originalbilledet, for illustrationens skyld. Detekterede ridgepunkter er tegnet som sorte prikker.



Figur 3.6: Som figur 3.5 men, i stedet for sorte punkter, med sorte streger som starter i ridgepunkterne og som peger vinkeltret på deres orienteringer.

## 3.2 Streggenkendelse

Streggenkendelsens funktion er ud fra uddata fra ridgefilteret at segmentere billedet. Sorte eller hvide streger i billedet identificeres og repræsenteres som pile, der angiver position og retning af stregerne.

### 3.2.1 Ekspansion

Ridgefilteret giver som nævnt punkter, som er kandidater til streger i billedet samt for hvert punkt en orientering. Kandidatpunkter til hvide streger er repræsenteret ved værdien 1, sorte ved værdien 2, og 0 repræsenterer punkter, som ikke er kandidater (baggrund). Ved den morfologiske operation *ekspansion* (eng: dilation) ekspanderes ridgepunktbilledet, idet ridgepunkterne ikke altid er sammenhængende. Som strukturelement anvendes et $3 \times 3$ element S:

$$S = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Ekspansionen foretages binært. Dog har vi både 1- og 2-taller, så 1-taller ekspanderes med 1-taller og

2-taller ekspanderes med 2-taller (uafhængigt af at S består af 1-taller, de angiver blot "her skal der ekspanderes"). En ekspansion med S får punkter hvis afstand er 2 pixel eller mindre, til at blive sammenhængende.

### 3.2.2 Sammenhængende komponenter

Ud fra det ekspanderede ridgepunktbillede findes nu sammenhængende komponenter (eng: connected components) af hhv. 1-taller og 2-taller. Der kigges på 8-sammenhængende punkter, og kun hvis punktet og nabopunktets værdi er ens (begge 1 eller begge 2), regnes de som værende sammenhængende. For hver komponent udregnes: Størrelse (antal pixel), massecentrum, bounding box, farve og gennemsnitlig orientering.

Det ekspanderede ridgepunktbillede bruges kun til at bestemme hvilke ridgepunkter, der hænger sammen. Til udregning af de ovennævnte komponentegenskaber bruges kun ridgepunkterne, ikke de ekspanderede ridgepunkter. Når der i det følgende refereres til *komponenter*, menes de udregnede komponentegenskaber.

Den gennemsnitlige orientering udregnes som gennemsnittet af orienteringen af hver af ridgepunkterne. Dog opstår der et problem, da orienteringen er givet ved en retningsvektor $v$:

$$v = \begin{pmatrix} dx \\ dy \end{pmatrix}$$

Orienteringen angiver nemlig hældningen af en linje langs stregen (ridgen), og $v$ og $-v$ angiver samme hældning, se figur 3.7. Tages gennemsnittet af en mængde af vektorer, hvoraf nogle peger ca. i retning $v$, og andre peger ca. i retning $-v$, bliver resultatet mere eller mindre vilkårligt. For at undgå dette udtages før udregning af gennemsnitsorienteringen af hver komponent en retningsrepræsentant $v_r$ bland vektorerne. Ved gennemsnitsudregningen summeres over hver orienteringsvektor $v_i$, men for hver vektor tjekkes dens prikprodukt med $v_r$. Er dette mindre

Figur 3.7: Retningsproblemet. Her starter de sorte streger i ridgepunkterne og peger (for klarhedens skyld) vinkelret på ridgen.

end 0 (svarende til at vinklen mellem $v_i$ og $v_r$ er større end 90 grader), adderes $-v_i$ til summen. Til sidst multipliceres summen med $\frac{1}{n}$, hvor $n$ er antallet af orienteringsvektorer i komponenten:

$$\frac{1}{n}\left(\sum_{i=1, v_i \cdot v_r \geq 0}^{n} v_i - \sum_{i=1, v_i \cdot v_r < 0}^{n} v_i\right)$$

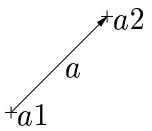
Dette kan selvfølgelig gå galt, hvis $v_r$ vælges uheldigt (nemlig i en retning vinkelret på gennemsnitsretningen). Vi vælger en vilkårlig orienteringsvektor, nemlig den første vi møder i summeringen $v_1$, som repræsentant, og det virker fint i praksis.

For at fjerne uvedkommende støj, frasorteres meget små komponenter (som har et lille antal ridgepunkter) ud fra en tærskelværdi.

### 3.2.3 Pile

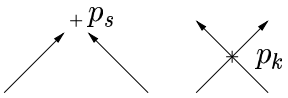
En pil definerer vi som et afgrænset linjestykke med en valgt retning, repræsenteret som:

$$a = \left( \begin{pmatrix} a_{1_x} \\ a_{1_y} \end{pmatrix}, \begin{pmatrix} a_{2_x} \\ a_{2_y} \end{pmatrix} \right)$$




Figur 3.8: Definition af en pil.

$a1$ og $a2$ er punkter i planen. Retningen ligger implicit i repræsentationen, som værende fra $a1$ mod $a2$, se fig. 3.8. I det efterfølgende kaldes $a2$ for pilens hoved og $a1$ for pilens hale. Med *skæringen* mellem to pile, vil vi i det efterfølgende mene skæringen mellem linjerne som de to pile udspænder, og *krydsningen* af to pile betegner tilfælde hvor de to linjestykker, pilene selv udgør, skærer hinanden. Se fig. 3.9.



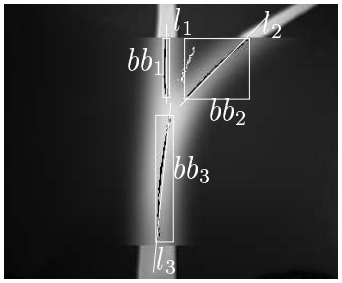

Figur 3.9: Definition af skæring hhv. krydsning af to pile.

Den gennemsnitlige orientering, massecentrum samt bounding box af ridgepunkterne i hver komponent bruges til at beregne pile, som ligger i samme retning som den oprindelige streg, samt har samme længde og placering. Dette gøres ud fra linien $l$, der har orientering som den gennemsnitlige orientering, og som går gennem massecentrum samt ud fra bounding box'en $bb$. Se fig. 3.10.

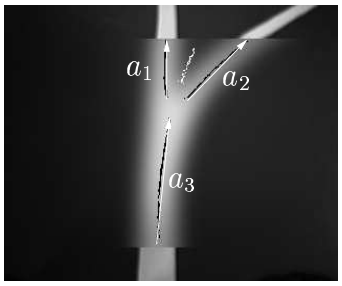
Skæringerne mellem $l$ og $bb$ findes. Dette giver pilens to endepunkter. Dens retning vælges, så den peger væk fra robotten (svarer til "op" i billedet) ved at bytte om på endepunkterne, hvis pilen peger mere nedad end opad. Se fig. 3.11.

### 3.2.4 Sammenlægning af komponenter

Selv efter den morfologiske ekspansion, kan der stadig findes komponenter som ikke er sammenhæn-




Figur 3.10: Her er indtegnet $l$ (afkortede for klarhedens skyld) og $bb$ for de tre hvide komponenter.




Figur 3.11: Her er indtegnet de tre resulterende pile.

gende, men hvis tilhørende pile ligger i forlængelse af hinanden og har samme retning. Disse sammenlægges til een pil på følgende måde:

Enhver pil $a_i$ i pilemængden sammenlignes vha. en sammenligningsfunktion med enhver anden pil $a_j$. Hvis sammenligningsfunktionen vurderer at $a_i$ og $a_j$ kan sammenlægges til een pil, gør den det, og sammenligningen af $a_i$ starter forfra, dvs. sammenlignes igen med enhver $a_j$.

Sammenligningsfunktionen sammenligner to pile $a_i$ og $a_j$ og består af to dele:

1. Sammenligning. Det undersøges om:

   (a) Den relative afstand (i forhold til pilenes samlede længde) mellem de modstående endepunkter for $a_i$ og $a_j$ er tilstrækkelig lille, altså om $a_i$ og $a_j$ ligger i forlængelse af hinanden:

   $$\frac{\|a_i 1 - a_j 2\|}{\|a_i\| + \|a_j\|} < 0.3 \quad \vee \quad \frac{\|a_i 2 - a_j 1\|}{\|a_i\| + \|a_j\|} < 0.3$$

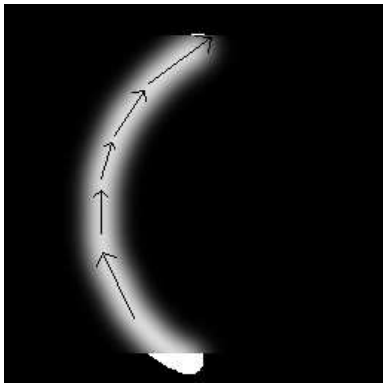
   (b) Farven af $a_i$ og $a_j$ er ens.

   (c) Vinklen mellem $a_i$ og $a_j$ er tilstrækkelig lille, højest $\frac{\pi}{6}$.

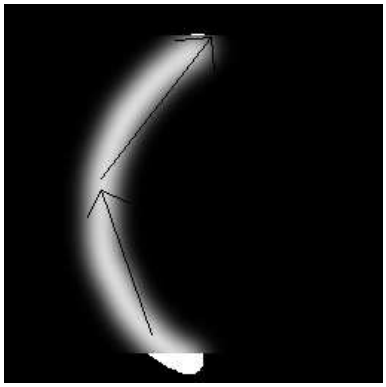
   De anvendte tærskelværdier, 30% og $\frac{\pi}{6}$, er fundet eksperimentelt.

2. Hvis alle punkter i sammenligningen er opfyldt, sammenlægges $a_i$ og $a_j$ og deres tilhørende komponenter. Dvs. ny bounding box og massecentrum udregnes for den samlede komponent og en ny samlet pil $a_{ij}$ udregnes. Endepunkterne for $a_{ij}$ findes som de to endepunkter (ud af de i alt 4) fra $a_i$ og $a_j$, som ligger længst fra hinanden, og igen sørges for, at pilen kommer til at pege væk fra robotten, altså "op" i billedet.

Se figur 3.12 og figur 3.13 for en illustration af sammenlægning af pile. Sammenlægningen kan virke som en lidt grov måde at smide data væk på, men det gør implementeringen af højniveaustyrefunktionerne væsentlig lettere.



Figur 3.12: Før sammenlægning af pile



Figur 3.13: Efter sammenlægning af pile

## 3.3 Højniveaustyring

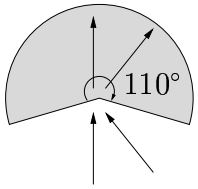
Højniveaustyringen sørger for den overordnede styrestrategi. Det er her, beslutninger, om til hvilken side robotten skal køre ved forgreninger eller kryds, tages. Dette gøres ved at kalde en sekvens af navigeringsfunktioner som hver, ud fra pilene fundet under streggenkendelsen, udvælger en pil, som skal køres efter.

### 3.3.1 Styresekvens

Vores hensigt med højniveaustyringen er, at det skal være let at indprogrammere en rute, som robotten skal tage på banen. Her er et eksempel (hver funktion returnerer, når betingelsen er opfyldt):

```
/* Følg en sort streg, og kør til venstre
når der kommer en forgrening med en hvid
streg: */

findFork(DIR_LEFT,
         WHITE_BLACK,
         WHITE_LINE);

/* Følg en hvid streg, og kør ligeud når
der kommer et kryds med hvide
forgreninger: */

findIntersection(DIR_STRAIGHT,
                 WHITE_LINE,
                 WHITE_LINE,
                 WHITE_LINE,
                 WHITE_LINE);

/* Følg en hvid streg ind til den
slutter: */

findEndOfLine(WHITE_LINE);
```

Herefter følger en beskrivelse af hver af navigeringsfunktionerne.

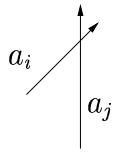



Figur 3.14: Pile vendes væk fra skæringspunktet, hvis deres vinkel er mindre end 110°. Er vinklen større, regnes pilen som værende en del af en samling.

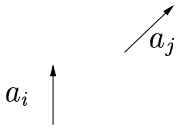
### 3.3.2 Funktion til at finde forgreninger

Funktionen følger en linie, indtil den møder en forgrening, og returnerer når den er passeret. Det overføres til funktionen, om robotten skal dreje til højre eller til venstre i forgreningen, samt stregfarverne i forgreningen. Funktionen udfører følgende punkter i en løkke, indtil forgreningen er fundet og passeret:

1. Tag nyt billede, og udfør streggenkendelse på det. Denne giver en liste af pile $l_a$.
2. For hver kombination af par af pile $(a_i, a_j)$ fra $l_a$ vendes pilene, så de vender væk fra deres skæringspunkt, hvis de har et sådant, og det ligger inden for billedet, altså inden for robottens synsfelt. En pil vendes dog kun, hvis dens vinkel fra lodret op er mindre end 110°. Se fig. 3.14.
3. Nu undersøges det, om der findes en forgrening i billedet. En række kriterier skal være opfyldt, før der er tale om en forgrening. For hver mulig kombination af par af pile $(a_i, a_j)$ undersøges, om de udgør en forgrening:
   (a) Vinklen mellem $a_i$ og $a_j$ skal være mindst 10° (hvilket samtidig sikrer, at linjerne udspændt af $a_i$ og $a_j$ skærer hinanden).
   (b) Skæringspunktet $p_s$ findes. For både $a_i$ og $a_j$ skal gælde, at afstanden mellem deres




Figur 3.15: Selv om pilene begge peger væk fra deres skæringspunkt, er der tale om en samling og ikke en forgrening.




Figur 3.16: Ikke en god kandidat til en forgrening, der er for langt mellem pilene.

pilehoved og $p_s$ skal være større end 20% af pilenes samlede længde:

$$\frac{\|p_s - a_{i_{p2}}\|}{\|a_i\| + \|a_j\|} > 0.2 \quad \wedge \quad \frac{\|p_s - a_{j_{p2}}\|}{\|a_i\| + \|a_j\|} > 0.2$$

Dette sikrer, at tilfælde som illustreret på fig. 3.15 ikke bliver opfattet som en forgrening.

(c) Afstanden mellem pilenes endepunkter og $p_s$ må ikke afvige for meget:

$$\frac{|\|p_s - a_{i_{p2}}\| - \|p_s - a_{j_{p2}}\||}{\|p_s - a_{i_{p2}}\| + \|p_s - a_{j_{p2}}\|} < 0.33$$

Dette sikrer, at tilfælde som illustreret på fig. 3.16 ikke bliver opfattet som en forgrening.

(d) Pilenes længde skal være omtrent lige store:

$$\frac{|\|a_i\| - \|a_j\||}{\|a_i\| + \|a_j\|} < 0.70$$

(e) Pilenes farve skal matche farveparametrene til funktionen.

Hvis alle ovenstående betingelser er opfyldt, er en forgrening fundet. I tilfælde, hvor der findes mere end een forgrening i samme billede, returneres den største forgreningspil.

4. Hjulstyringsfunktionen kaldes med forgreningspilen som parameter. Hvis der ikke blev fundet en forgreningspil, så køres efter den største pil i billedet.

5. Når en forgrening er bekræftet (se afsnit 3.3.5) og passeret, returneres.

Konstanterne i det ovenstående er fundet eksperimentielt ved at køre funktionen på et træningssæt af billeder og justere konstanterne, indtil en korrekt identifikation er opnået.

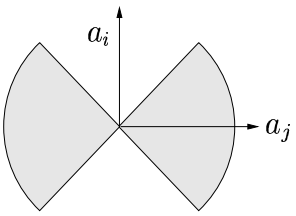
Et problem i den implementerede algoritme er, at den først gennemløber hele $l_a$ for at vende pile væk fra skæringspunkter og derefter igen gennemløber $l_a$ for at finde forgreningspar. En pil risikerer nemlig først at blive vendt væk fra et skæringspunkt og derefter blive vendt væk fra et andet skæringspunkt, så den alt i alt ikke er blevet vendt. Optimalt burde løkken kun gennemløbes en enkelt gang, idet hvert pilepar så først vendes væk fra skæringspunktet og derefter undersøges om det er et forgreningspar. I praksis opstår problemet dog sjældent.

### 3.3.3 Funktion til at finde kryds

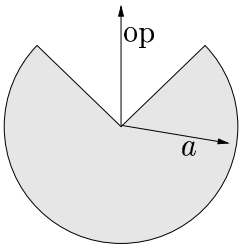
Funktionen følger en linje indtil den møder et kryds og returnerer, når det er passeret. Det overføres til funktionen, om robotten skal dreje til højre eller til venstre eller køre ligeud i krydset, samt pilefarverne krydset skal have. Funktionen udfører følgende punkter i en løkke, indtil krydset er fundet og passeret:

1. Tag nyt billede, og udfør streggenkendelse på det. Denne giver en liste af pile $l_a$.

2. Ud fra listen $l_a$ findes en liste $ls_a$, der indeholder de pile, som har et skæringspunkt med en anden pil i $l_a$, som ligger inden for billedet (dvs. i robottens synsfelt).




Figur 3.17: Vinklen mellem $a_i$ og $a_j$ skal være mellem 45° og 135°, dvs. i nærheden af 90°, for at være en del af et kryds.




Figur 3.18: Vinklen mellem $a$ og lodret op skal være mindst 45°, før $a$ bliver vendt væk fra skæringspunktet.

3. For hver kombination af par af pile $(a_i, a_j)$ fra $ls_a$ vendes pilene, så de vender væk fra deres skæringspunkt, hvis de har et sådant, hvis det ligger inden for billedet (dvs. ligger inden for robottens synsfelt) og hvis vinklen mellem pilene er mellem 45° og 135°. Se figur 3.17. Desuden vendes en pil kun, hvis vinklen mellem den og en pil lodret op er mindst 45°. Se fig. 3.18.

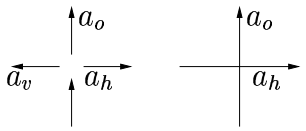
4. Den længste pil fra $ls_a$ i retningen hhv. venstre, op og højre findes, hvis de eksisterer: $a_v$, $a_o$, $a_h$. En pil i en retning skal her blot ligge inden for ±45° i forhold til retningen. Pilenes farve skal desuden matche farveparametrene til funktionen.

5. Findes både $a_v$, $a_o$ og $a_h$, er et kryds i billedet fundet.

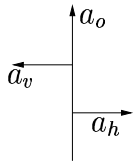
   Hvis $a_o$ findes og en og kun en af $a_v$ og $a_h$ findes, kan der stadig være tale om et kryds, idet den højre og venstre pil kan være blevet sammensat til een lang jf. afsnit 3.2.4. Se fig. 3.19. Det undersøges, om den fundne højre/venstrepil krydser oppilen. Hvis den gør det, er der også tale om et kryds. Hvis retningsparameteren til funktionen specificerer, at der skal drejes til højre eller venstre i et fundet kryds, vendes højre/venstrepilen, så den peger i den ønskede retning.

   Hvis der ikke blev fundet et kryds, vælges den største pil i billedet som køreretning.

6. Hjulstyringsfunktionen kaldes med den fundne retningspil.




Figur 3.19: Højre- og venstrepilene i et kryds kan være blevet sammensat til een.




Figur 3.20: To T-kryds efter hinanden i synsfeltet vil blive genkendt som et kryds.

Algoritmen tager ikke højde for tilfældet illustreret i fig. 3.20, men da situationen ikke findes på RoboCup-banen (se fig. 1.1), har det ikke været et problem. At udvide den til at tage højde for det, kan let implementeres.

### 3.3.4 Andre navigeringsfunktioner

Ud over funktionerne til at finde forgreninger og kryds, har vi yderligere et par navigeringsfunktioner:

1. En funktion til at finde enden af en linje. Denne finder blot den største pil i billedet og styrer efter denne.

2. En funktion, som blot kører ligeud, indtil den finder en streg.

### 3.3.5 Bekræftelsesfunktion

Samtlige navigeringsfunktioner bruger bekræftelsesfunktionen, som fungerer som følger.

For at undgå at tilfældig støj skal have for meget indflydelse på navigeringsfunktionerne, returneres der ikke fra dem, så snart de har fundet det første billede, som opfylder deres søgekriterie. Tag f.eks funktionen til at finde forgreninger. Hvis den på et enkelt billede har fundet en forgrening, og den så på det næste ikke ser en forgrening, betyder det ikke nødvendigvis, at den har fundet og passeret en forgrening. Lysforhold, bevægelsessløring (eng. motion blur) etc. får streggenkendelsen til at fejle ind imellem.

Bekræftelsesfunktionen består af to faser:

1. En tæller $t$ nulstilles. Når en navigationsfunktion har fundet sit søgekriterie i et billede, tælles $t$ op. Når søgekriteriet ikke findes i et billede, tælles $t$ ned (kun til og med 0). Når $t$ overstiger en tærskelværdi, markeres kriteriet som bekræftet, og næste fase udføres.

2. $t$ nulstilles igen. Når navigeringsfunktionen nu *ikke* finder kriteriet, tælles $t$ op, ellers tælles $t$ ned (igen kun til 0). Når $t$ overstiger en tærskelværdi, markeres kriteriet som bekræftet og passeret, og navigeringsfunktionen kan nu returnere.

Som tærskelværdi har vi brugt 3 (dvs. 3 billeder), fundet eksperimentielt ved at køre funktionen på nogle realistiske sekvenser af testbilleder.

### 3.3.6 Visualiseringen af navigationsfunktionernes beslutninger

Hver navigationsfunktion sørger for at gemme de behandlede billeder, og deres tilhørende original på disken under programmets udførsel. På de behandlede billeder indtegnes desuden de fundne pile, farvekodet ud fra navigationsfunktionernes beslutninger på følgende måde:

1. Øverst i venstre hjørne tegnes et lille ikon, som illustrerer, hvilken navigationsfunktion som behandlede billedet, samt hvad den var sat til at køre efter fremhævet i tyrkis. F.eks at køre til venstre i en forgrening.

2. Bliver en navigationsfunktion bekræftet i, hvad den søger efter, tegnes en gul firkant om ikonet. Bliver det søgte vurderet som bekræftet og passeret, tegnes en grøn firkant om ikonet. Se afsnit 3.3.5 for definitionen af hhv. bekræftet og passeret.

3. Finder en navigationsfunktion hvad den leder efter, tegnes den aktuelle pil i rød.

4. Finder en navigationsfunktion ikke hvad den leder efter, tegnes den pil, som så køres efter i grøn.

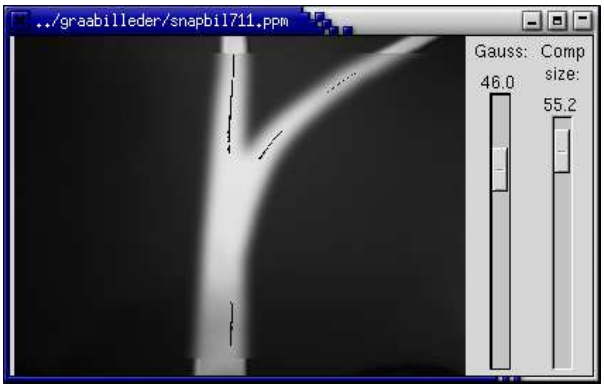
5. De resterende fundne pile i billedet tegnes i blå.

Billedsekvenserne optaget under konkurrencen, både i original og i behandlet form, kan ses som mpeg-film, vedlagt på CD i bilag B. Se også afsnit 7 for en beskrivelse af CDens indhold. I de behandlede sekvenser kan man således visuelt følge med i, hvad robotten (tror den) ser.

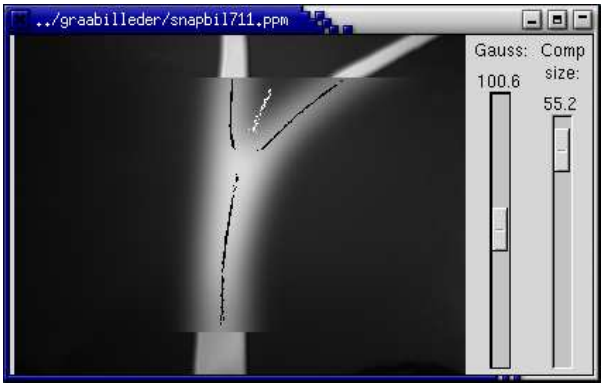
## 3.4 GUI

Under udviklingen af streggenkendelsen havde vi brug for en måde til at justere forskellige parametre på og hurtigt se resultatet på et testbillede. Til formålet blev udviklet et lille GUI-program i gtk[3].

Se fig. 3.21 og fig. 3.22. Vha. skydeknapperne i højre side, kan parametrene ændres interaktivt. På figuren er knapper til ændring af Gausskernens spredning (og størrelse) og mindste accepterede komponentstørrelse (se hhv. afsnit 3.1.2 og afsnit 3.2.2) medtaget.




Figur 3.21: Interaktivt program - Gausskernestørrelsen er for lille.




Figur 3.22: En passende Gausskernestørrelse.

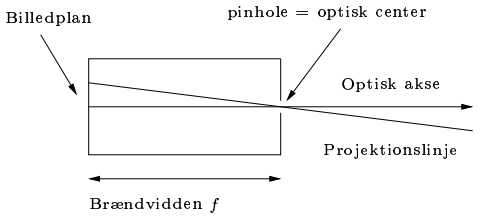
## 3.5 Koordinattransformation

### 3.5.1 Beregning af konstanter

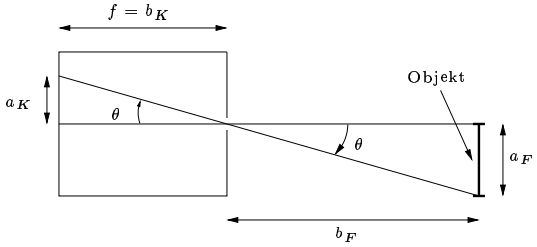
Vi antager, at kameraet tilnærmelsesvis er et pinholekamera. Se fig. 3.23. Kameraets brændvidde $f$ findes ud fra opstillingen i fig. 3.24. Her ses, at:

$$\frac{a_F}{b_F} = \frac{a_K}{f} \iff$$

$$f = \frac{a_K b_F}{a_F} \tag{3.1}$$




Figur 3.23: Pinholekamera - Efter [10].




Figur 3.24: Brændvidden $f$ findes ved denne opstilling.

I praksis behøver objektets ene ende ikke ligge lige ud for den optiske akse, som den gør på figuren. For at udregne $f$ ud fra 3.1 måles $a_K$ (objektets fysiske størrelse) og $b_K$ (længden fra kameraet til objektet). $a_K$ måles ved at tage et billede, og i et billedbehandlingsprogram måle objektets størrelse i billedet (i enheden pixel). Da $a_K$ måles i pixel, bliver også enheden af den udregnede $f$ i pixel.

Konstanten $\theta_K$, som er kameraets vinkel i forhold til kameratårnet, kan findes ved at opmåle $h_K$ og $Z_{oa}$ (se fig. 3.28) og indsætte værdierne i den trigonometriske formel:

$$\tan(\theta_K) = \frac{h_K}{z_{oa}} \quad \Longleftrightarrow$$

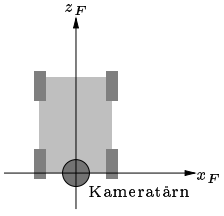
$$\theta_K = \tan^{-1}\left(\frac{h_K}{z_{oa}}\right) \tag{3.2}$$

Alternativt kan en vinkelmåler anvendes til at finde $\theta_K$, om end det sikkert vil give mindre præcision.

### 3.5.2 Definition af koordinatsystemer

Billedkoordinatsystemet er todimensionelt og har origo i øverste venstre hjørne. De to akser $(x_B, y_B)$ er defineret, så $x_B$ er positiv mod højre, og $y_B$ er positiv nedad. Se fig. 3.26.

Verdenskoordinatsystemet er også todimensionelt. Det ligger i gulvplanen og har origo i det gulvpunkt, som ligger lige lodret under kameraets optiske center. De to akser $(x_F, z_F)$ er defineret, så $x_F$ er positiv mod højre og $z_F$ positiv fremad (begge set i robottens køreretning). Se fig. 3.25.

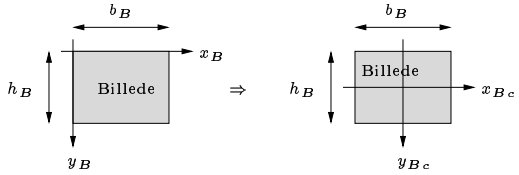



Figur 3.25: Verdenskoordinatsystemet.

### 3.5.3 Transformation fra billedkoordinater til verdenskoordinater

Før transformationen bliver billedkoordinaterne $(x_B, y_B)$ centreret. Se fig. 3.26:

$$x_{Bc} = x_B - \frac{w_B}{2} \quad , \quad y_{Bc} = y_B - \frac{h_B}{2} \tag{3.3}$$




Figur 3.26: Koordinatsystemets origo flyttes til centrum af billedet.

Nu transformeres fra de centrerede billedkoordinater $(x_{Bc}, y_{Bc})$ til verdenskoordinater (fysiske koordinater) $(x_F, z_F)$, igen antages kameraet at være et pinholekamera.

Først findes $\theta_{y_{Bc}}$ ud fra at betragte trekanter på figur 3.27:

$$\tan(\theta_{y_{Bc}}) = \frac{y_{Bc}}{f} \quad \Longleftrightarrow$$

$$\theta_{y_{Bc}} = \tan^{-1}\left(\frac{y_{Bc}}{f}\right) \tag{3.4}$$

Igen ud fra figur 3.27 ses:

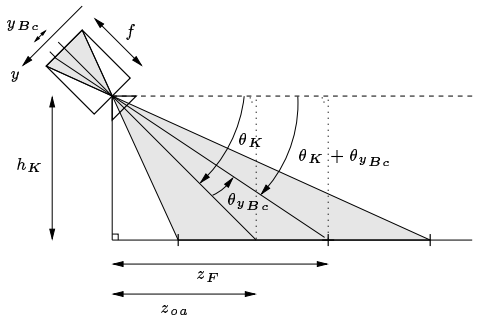
$$\tan(\theta_K + \theta_{y_{Bc}}) = \frac{h_K}{z_F} \quad \Longleftrightarrow$$

$$z_F = \frac{h_K}{\tan(\theta_K + \theta_{y_{Bc}})} \tag{3.5}$$

Indsættes (3.4) i (3.5) kan $z_F$ findes:

$$z_F = \frac{h_K}{\tan\left(\theta_K + \tan^{-1}\left(\frac{y_{Bc}}{f}\right)\right)} \tag{3.6}$$

Her er $h_K$, $\theta_K$ og $f$ som nævnt konstanter, og $y_{Bc}$ er den centrerede billedkoordinat i det aktuelle billede.




Figur 3.27: Kameraets synsfelt set fra siden.

Ud fra fig. 3.28 ses, at:

$$\frac{x_F}{b_F} = \frac{x_{Bc}}{f} \quad \Longleftrightarrow$$

$$x_F = \frac{x_{Bc} b_F}{f} \tag{3.7}$$

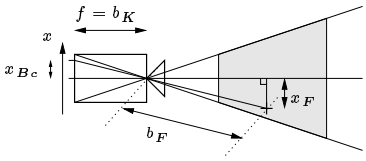
Længden $b_F$ findes ud fra Pythagoras sætning:

$$b_F = \sqrt{h_k^2 + z_F^2} \tag{3.8}$$

Indsættes (3.8) i (3.7) kan $x_F$ findes:

$$x_F = \frac{x_{Bc}\sqrt{h_k^2 + z_F^2}}{f} \tag{3.9}$$

Her er $h_K$ og $f$ konstanter, $z_F$ findes ud fra (3.6) og $x_{Bc}$ er den centrerede billedkoordinat i det aktuelle billede.




Figur 3.28: Kameraets synsfelt set ovenfra.

# Kapitel 4

# Styring af bilen

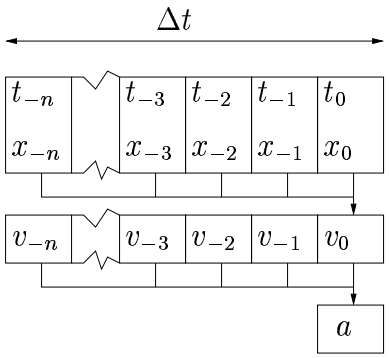
## 4.1 Motorstyring

Vi har under udvikling af robotten brugt megen tid på at finde en passende fremdrift. Grundliggende skyldes vores problemer, at bilen vi har brugt som udgangspunkt er bygget til at have en tophastighed på over 30 km/t, hvilket er alt for hurtigt til RoboCup. Den høje tophastighed medfører også, at bilen er højt gearet, hvilket giver ekstra problemer pga. udstyrets store masse. Problemet er, at når vi skal køre ved lave hastigheder, så skal motoren have lav spænding og høj strøm.

### 4.1.1 Måling af hastighed og acceleration

En åbenlys forudsætnig for at kunne justere robottens hastighed er, at robottens øjeblikkelige hastighed kan måles. Som nævnt har vi monteret to takometre på robotten. Via dem kan vi for hver kørte 3 mm aflæse robottens position. Hver gang vi modtager en afstandsmåling fra musen, gemmes den i en cyklisk buffer sammen med tidpunktet. Vi refererer i det følgende til den nyeste måling som $x_0, t_0$ og målingen for $n$ målinger før som $x_{-n}, t_{-n}$.

Hver gang der er fortaget en måling, ønskes den aktuelle hastighed og acceleration beregnet. Hastigheden er givet ved $v = \frac{dx}{dt}$ og kan i det diskrete tilfælde beregnes ved $v_n = \frac{x_n - x_{n-1}}{t_n - t_{n-1}}$. For at opnå mere stabil hastighedsmåling midles der over et tidsinterval $\Delta t$.




Figur 4.1: De to buffere hvor tid, sted og fart tripler gemmes. Bufferne opdateres for hver museevent hvor tid og den nye museposition gemmes. Ud fra events der er mindre end $\Delta t$ gamle beregnes den aktuelle hastighed, som også gemmes. Ud fra hastighedsberegninger der er mindre end $\Delta t$ gamle beregnes den aktuelle acceleration.

Jo større $\Delta t$ vælges, des mindre er sandsynligheden for fejlmålinger, men des langsommere reagerer robotten på hastighedsforandringer. I praksis fandt vi, at $\Delta t = 100$ms fungerede godt. $v$ findes da som:

$$v = \frac{1}{\sum_{t_n > t_0 - \Delta t}^{n} 1} \sum_{t_n > t_0 - \Delta t}^{n} \frac{x_n - x_{n-1}}{t_n - t_{n-1}} \qquad (4.1)$$

Den beregnede hastighed gemmes ligesom positionen i en cyklisk buffer som $v_0$. Accelerationen er defineret som $a = \frac{dv}{dt}$. Da accelerationen $a$ forholder

sig på samme måde til $v$, som $v$ forholder sig til $x$, er beregningen af $a$ helt analog til beregningen af $v$. Ud fra de gemte hastigheder beregnes accelerationen $a$ som:

$$a = \frac{1}{\sum_{t_n > t_0 - \Delta t}^{n} 1} \sum_{t_n > t_0 - \Delta t}^{n} \frac{v_n - v_{n-1}}{t_n - t_{n-1}} \qquad (4.2)$$

### 4.1.2 Jævnstrømsmotorstyring

I databladet på motoren [5] kan vi læse, at motorens moment (kraft) vokser lineært med, hvor meget strøm den får. Til gengæld er momentet ikke direkte afhængig omdrejningshastighed og spænding.

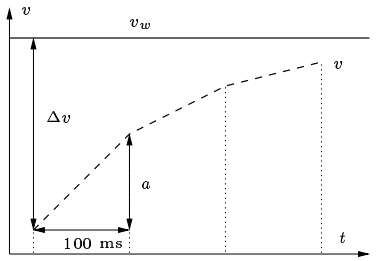
MacGyvercontrolleren giver en pulsbreddemoduleret 12V spænding. Det vil sige, at den sætter spændingen over motoren til skiftevis 0 og 12V ved 14KHz. En DC-motor er, ved passende høje frekvenser, ufølsom over for pulsbreddemoduleringen og reagerer på samme måde, som hvis den fik en DC-spænding på $U = 12V\frac{t_{\text{on}}}{t_{\text{off}}}$, hvor $t_{\text{on}}$ og $t_{\text{off}}$ er den del af perioden, hvor spændingen er 12V hhv. 0V.

Vi kan ikke sige noget om sammenhængen mellem spænding og strøm ud over, at de er proportionale. Da det kun er spændingen vi kan styre, og da momentet, som nævnt, er afhængig af strømmen, har vi altså ikke direkte kontrol over motorens moment. Det gør, at vi ikke kan programmere motorstyringen 100% effektivt.

### 4.1.3 Fartregulering

Fartreguleringen er implementeret i sin egen tråd og justerer motorspændigen 10 gange i sekundet.

Motorstyringen tager udgangspunkt i differensen mellem den ønskede hastighed $v_w$ og den aktuelle hastighed $v$. Denne benævnes $\Delta v$. Målet for hastighedsreguleringen for hver iteration (hvert $\Delta t = 100$ms) kørte $\Delta v$ $p$ gange mindre, hvor $p$ er dæmpningskonstanten. En illustration af dette




Figur 4.2: Skitse af hvordan den overdæmpede fartregulering med tiden vil bringe farten $v$ mod den ønskede fart $v_w$.

princip ses i figur 4.2, hvor $p = 2$. For at opnå dette udnyttes at $a = \frac{dv}{dt}$. Kombineres dette med vores ønske om at $v$ i den følgende iteration skal være $\frac{\Delta v}{p}$ større fås ligningen:

$$\frac{dv}{dt}\Delta t = \frac{\Delta v}{p} \iff \qquad (4.3)$$

$$a_w = \frac{\Delta v}{p\Delta t} \qquad (4.4)$$

$$= \frac{v_w - v}{p\Delta t} \qquad (4.5)$$

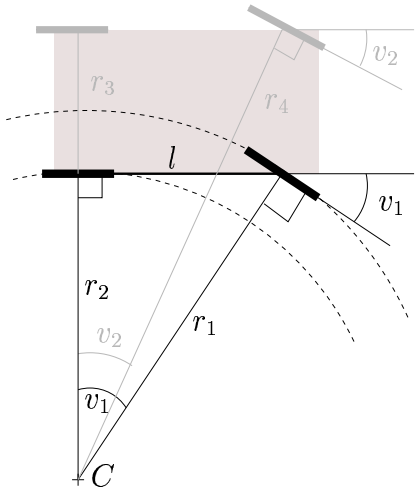
hvor $a_w$ er den acceleration, vi ønsker.

Som beskrevet i afsnit 4.1.2 er

$$a = \frac{F}{m} = kI, \qquad (4.6)$$

hvor $m$ er bilens masse, $I$ er strømmen gennem motoren og $k$ er en konstant, der indeholder motoregenskaber, hjulstørrelse og gearing. Havde vi kunnet sætte $I$ direkte er dette muligvis en optimal styrestrategi.

I stedet bruger vi inddirekte (4.6) til at estimere $I$ ud fra $a$, som vi har målt som beskrevet i afsnit 4.1.1. I praksis sammenligner vi den målte acceleration $a$ med den ønskede acceleration $a_w$. Er $\Delta a = a_w - a$ positiv, er $I$ for lille og motorspændingen øges proportionalt med $\Delta a$. Er $\Delta a$ negativ, er $I$ for stor og motorspændingen sænkes proportionalt med $\Delta a$.




Figur 4.3: Bilen vil ved konstant hjulposition følge en cirkel. Det er muligt at finde en sammenhæng mellem hjulenes vinkel $v$, afstanden mellem hjulene og diameteren på cirkelbuen, som bilen vil følge.

## 4.2 Styrestrategi

### 4.2.1 Bilens styreegenskaber

En forudsætning for at kunne lægge en styrestrategi er, at vi ved, hvad en given vinkel af de styrbare hjul (herefter benævnt forhjulene) betyder for den bane, bilen vil følge.

En vigtig observation er, at hvert hjul på bilen kun kan bevæge sig langs dets akse. Det medfører, at hjulets retning til enhver tid er tangent til den bane, hjulet følger.

Ved konstant hjulvinkel $v_2$ (se figur 4.3) vil hvert hjul følge randen af en cirkel. Da den indbyrdes afstand mellem bilens hjul er konstant, må cirklerne have samme centrum $C$. $C$ findes som skæringen af linjerne, der går ortogonalt på kørselsretningen gennem hjulene. Cirkelradius findes da som

$$\tan(v_1) = \frac{l}{r_2} \iff r_2 = \frac{l}{\tan(v_1)} \tag{4.7}$$

eller for $r_1$:

$$\sin(v_1) = \frac{l}{r_1} \iff r_1 = \frac{l}{\sin(v_1)}$$

Som det ses af figuren, er forskellen på $r_2$ og $r_3$ bilens bredde. Ved betragtning af $r_4$ kommer man frem til det — måske overraskende — resultat, at de to hjul ikke drejer lige meget. I praksis er forskellen på $v_1$ og $v_2$ langt mindre end den præcision vi, pga. slør i hjulstyringen, kan sætte vinklen ved. Vi vælger derfor at benytte (4.7) til at sætte hjulvinklen.

## 4.3 Kørsel mod punkt eller linje

I dette afsnit betragtes et manøvreprimitiv. Ved et manøvreprimitiv forstås, at hjulene sættes til en fast vinkel, og der køres $d$ mm.

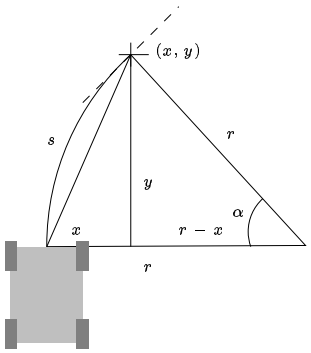
Ved et manøvreprimitiv bevæger bilen sig altså $d$ mm langs randen af en cirkel med radius $r$. Ved et manøvreprimitiv kan bilen bringes til at stå på en linje parallelt med denne dog uden, at man kan bestemme, hvor på linjen, bilen skal stå. En anden mulighed i et manøvreprimitiv er at køre bilen mod et punkt, dog uden at man kan bestemme bilens retning, når den når punktet.

### 4.3.1 Kørsel mod et punkt

Et grundlæggende primitiv for styringen er at få bilen til at køre mod et punkt.

I figur 4.4 er situationen, hvor bilen skal køre mod punktet $(x, y)$, skitseret. Ud fra figurens symboler kan drejningsradien $r$ findes ved at bruge Pytagoras sætning om retvinklede trekanter:

$$\begin{aligned} r^2 &= y^2 + (r - x)^2 \\ &= y^2 + r^2 + x^2 - 2rx \\ \iff & \\ r &= \frac{x^2 + y^2}{2x} \end{aligned} \tag{4.8}$$




Figur 4.4: Skitse over geometrien, der benyttes til at finde radien $r$, der medfører, at bilen kører igennem punktet $(x, y)$.

Indsættes $r$ i (4.7) fås den vinkel hjulene skal sættes til.

Afstanden bilen skal køre for at nå punktet er cirkelbuens længde.

$$s = 2r\pi\frac{\alpha}{2\pi} = r\alpha \tag{4.9}$$

Indsættes vinklens $\alpha$.

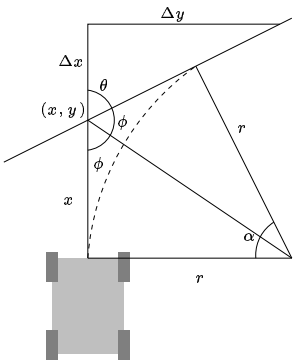
$$\sin\alpha = \frac{y}{r} \tag{4.10}$$

fås

$$s = r\sin^{-1}\frac{y}{r} \tag{4.11}$$

### 4.3.2 Kørsel mod en linje

Et andet primitiv er at få bilen til at køre hen til en linje og stille sig parallelt med denne. Vi forudsætter, at linjen er foran bilen. Vi lader linjen være defineret ved dens hældning $\frac{\Delta y}{\Delta x}$ og punktet $(x, y)$, som skal ligge på bilens længdeakse (se figur 4.5). Derved er $x$ givet som afstanden mellem bilen og stregen langs bilens længdeakse.




Figur 4.5: Skitse over geometrien, der benyttes til at finde radien $r$, der medfører, at bilen kører op på og parallelt med stregen, der går igennem punktet $(x, y)$ og har hældningen $\frac{\Delta y}{\Delta x}$.

Ud fra figur 4.5 findes nu trigonometrisk:

$$\tan\theta = \frac{\Delta y}{\Delta x} \tag{4.12}$$

$$2\phi = \pi - \theta \tag{4.13}$$

$$\tan\phi = \frac{r}{x} \tag{4.14}$$

Kombineres de tre ligninger fås

$$r = x\tan\frac{\pi - \tan^{-1}\frac{\Delta y}{\Delta x}}{2} \tag{4.15}$$

som kan indsættes i (4.7), for at få den vinkel, hjulene skal sættes til.

Afstanden bilen skal køre findes ud fra (4.9) hvor

$$\alpha = 2\left(\frac{\pi}{2} - \phi\right) = \pi - 2\phi \tag{4.16}$$

Indsættes $\alpha$ i (4.9) fås:

$$s = r(\pi - 2\phi) \tag{4.17}$$

## 4.4 At følge en streg

I afsnit 4.2.1 har vi vist, at bilen altid vil følge en cirkelbue, og hvordan vi ved at ændre hjulenes vinkel $v$ kan justere cirklens radius $r$. I afsnittene 4.3.1 og 4.3.2 har vi beskrevet to primitiver, som kan bringe bilen til hhv. et punkt og en linje. I dette afsnit vil vi sammensætte disse primitiver til en strategi for at følge en streg, og i de efterfølgende afsnit vil vi beskrive, hvorfor denne strategi er vanskelig at implementere robust i praksis.

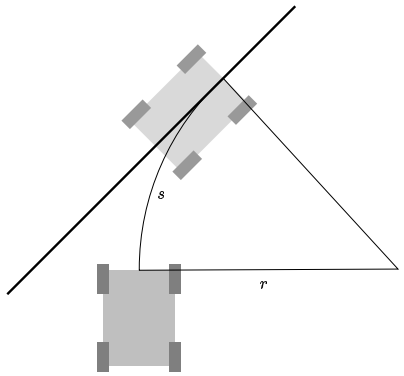
Bilen kan være i tre tilstande i forhold til stregen. På vej mod stregen, parallel med stregen og på vej væk fra stregen. Disse tilfælde er illustreret i figurerne 4.6–4.10. Vi antager her, at stregen ikke krummer, hvilket selvfølgelig kun gælder, hvis man betragter et passende kort stykke af den. Antagelsen passer godt med, at vores streggenkendelsesmetode giver en vektor, som repræsenterer den fundne streg.

En ting, vi er nødt til at tage særlig højde for, er bilens bredde. Hvis den er mere end nogle få cm. fra stregen, når den skal passere en port, vil den ramme porten og dermed være ude af konkurrencen. Figurerne til dette afsnit er for overskuelighedens skyld stærkt overtegnede. I praksis er det uacceptabelt, at bilen er flere vognbredder væk fra stregen. Stregen bør som minimum til alle tider være et sted under bilen.

Vi antager i de følgende afsnit, at robotten kan ændre hjulvinklen øjeblikkeligt. Dette er desværre ikke tilfældet med de bløde brede hjul, der sidder på robotten. Den skal køre adskillige cm. før hjulvinklen er i den ønskede position. Dette kan dog sandsynligvis udbedres ved at montere smalle hårde hjul på robotten.

### 4.4.1 Kørsel mod stregen

Tilfældet hvor bilen er på vej mod stregen kan håndteres med et enkelt 'kør mod linje'-primitiv, som efterlader bilen på stregen. I dette tilfælde er




Figur 4.6: Bilen er på vej mod stregen.

$s$ og $r$ givet ved (4.17) og (4.15). Dette er vist i figur 4.6.

En mere optimal strategi, med hensyn til hvor hurtigt stregen nås, er vist i figur 4.7. Denne kan være relevant at bruge i det tilfælde, at afstanden til stregen er stor. Her køres først direkte mod stregen, hvorefter der skiftes retning, så bilen rettes op på stregen. De på figuren viste $r_1$ og $r_2$ sættes til bilens minimale drejeradius $r_{\min}$ for den kortest mulige vej. Vi vil i de følgende afsnit udlede de tilhørende værdier for $s_1$, $s_2$ og $s_3$. Resultatet af anstrengelserne findes i afsnit 4.4.4.

### 4.4.2 Kørsel parallelt med stregen

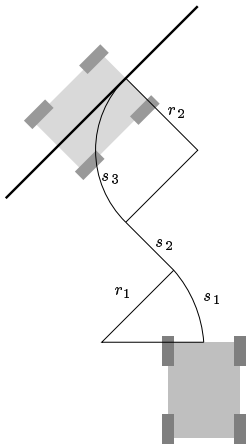
Befinder bilen sig parallelt med stregen i afstanden $d < 2r_{\min}$, kan den bringes tilbage på stregen ved brug af to køreprimitiver som vist i figur 4.8.

Her skal tages stilling til hvordan variablerne $r_1$, $r_2$, $s_1$ og $s_2$ vælges og beregnes.

Ud fra kriteriet om at vælge den kortest mulige vej til stregen sættes $r_1 = r_2 = r_{\min}$. Ud fra symbolerne på figuren findes:

$$f = r_1 - d \tag{4.18}$$

$$g = r_2 + f = r_2 + r_1 - d \tag{4.19}$$




Figur 4.7: Bilen er på vej mod stregen ad den kortest mulige bane.

$$\cos\phi = \frac{g}{r_1 + r_2} \tag{4.20}$$

Sammenholdes dette med (4.9) findes $s_1$

$$s_1 = r_1\phi \tag{4.21}$$

$$= r_1 \cos^{-1}\frac{g}{r_1 + r_2} \tag{4.22}$$

$$= r_1 \cos^{-1}\frac{r_2 + r_1 - d}{r_1 + r_2} \tag{4.23}$$

For $r_1 = r_2$ findes helt symmetrisk, at $s_2 = s_1$.

Er bilens afstand til stregen $d \geq 2r_{\min}$, skal der bruges et ekstra køreprimitiv som på figur 4.9. Vælges der, som vist på figuren, at køre direkte mod stregen bliver alle vinkler rette ($= \pi/2$) og

$$s_1 = \pi r_1/2 \tag{4.24}$$

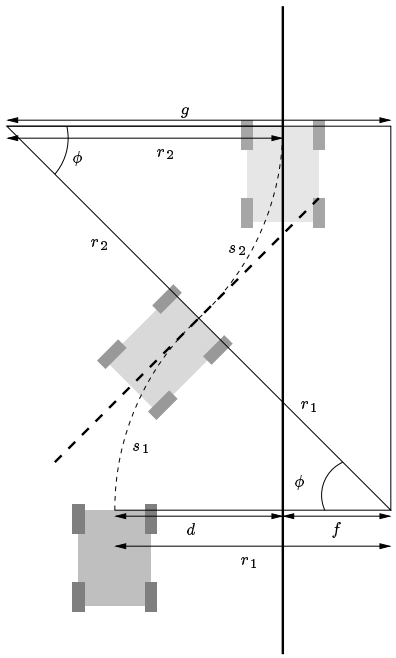
og

$$s_2 = \pi r_2/2 \tag{4.25}$$

findes ved hjælp af (4.9).

Af figuren ses tydeligt, at

$$s_4 = d - r_1 - r_2. \tag{4.26}$$



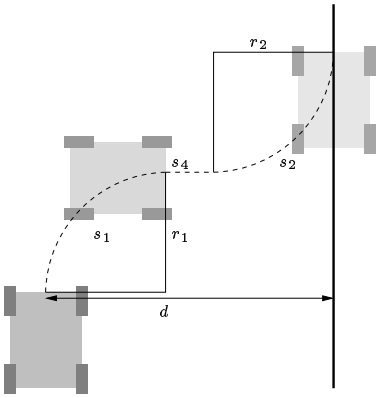

Figur 4.8: Bilen kører parallelt med stregen. Afstanden til stregen er $d < 2r_{\min}$

### 4.4.3 Kørsel væk fra stregen

Som det næsten fremgår af figur 4.10, er den vanskeligste manøvre at komme tilbage på stregen, når bilen er på vej væk fra denne. Bilen skal først rette op, så den er parallel med stregen. Derfra anvendes manøvren beskrevet i foregående afsnit til at køre ind på stregen. En forudsætning for at kunne anvende manøvren fra forrige afsnit er, at vi kender bilens afstand til stregen $d$, når den har rettet op. Vi skal altså i dette afsnit finde den drejningsradius $r_0$ og den afstand $s_0$, som bringer bilen parallel med stregen. Derudover skal vi finde bilens afstand $d$ til stregen, når den er rettet op.

Vi antager, at vi kender bilens afstand til stregen og dens orientering i forhold til stregen $\theta$. (Se figur 4.10 for definition af symboler.)

I overensstemmelse med det sædvanlige krav om optimering for korteste vej til stregen, vælges $r_0 =$




Figur 4.9: Bilen kører parallelt med stregen. Afstanden til stregen er $d \geq 2r_{\min}$

$r_{\min}$. Ud fra antagelsen om kendt $r_0$ findes $s_0$ ud fra (4.9):

$$s_0 = \theta r_0 \tag{4.27}$$

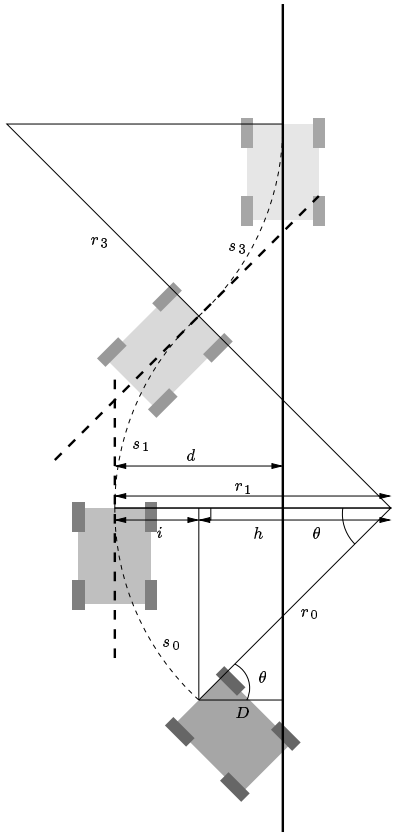
Ud fra symbolerne på figuren findes endvidere:

$$\begin{aligned} \cos\theta &= \frac{r_0}{h} && (4.28)\\ i &= r_0 - h && (4.29)\\ &= r_0 - \frac{r_0}{\cos\theta} && (4.30)\\ d &= i + D && (4.31)\\ &= r_0 - \frac{r_0}{\cos\theta} + D && (4.32) \end{aligned}$$

Vælges $r = r_0 = r_1 = r_2$, hvilket er oplagt, da de alle optimalt er lig $r_{\min}$, kan strækningerne $s_0$ og $s_1$ sammensættes til et styreprimitiv. Bilens drejeradius skal da sættes til $r = r_{\min}$, og afstanden der skal køres findes ved addition af (4.27) og (4.23) for $d < 2r_{\min}$ og ved (4.24) for $d \geq 2r_{\min}$.

For $d < 2r_{\min}$ fås

$$\begin{aligned} s_{0+1} &= s_0 + s_1 && (4.33)\\ &= \theta r + r\cos^{-1}\frac{2r-d}{2r} && (4.34) \end{aligned}$$



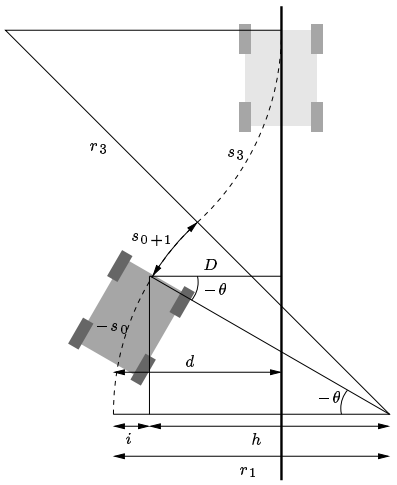

Figur 4.10: Bilen er på vej væk fra stregen og skal styres ind på stregen.

heri indsættes $d$ fra (4.32)

$$\begin{aligned} s_{0+1} &= \theta r + r\cos^{-1}\frac{2r-d}{2r} && (4.35)\\ &= \theta r + r\cos^{-1}\frac{2r-(r-\frac{r}{\cos\theta}+D)}{2r} && (4.36)\\ &= r(\theta + \cos^{-1}\left(\frac{1}{2} + \frac{1}{2\cos\theta} + \frac{D}{2r}\right) && (4.37) \end{aligned}$$

For $d \geq 2r_{\min}$ fås

$$\begin{aligned} s_{0+1} &= s_0 + s_1 && (4.38)\\ &= \theta r + r\pi/2 && (4.39) \end{aligned}$$




Figur 4.11: Bilen er på vej mod stregen og skal styres ind på stregen.

$$= r(\theta + \pi/2) \tag{4.40}$$

### 4.4.4 De universelle styreregler

I dette afsnit betragter vi (4.37) og (4.40), som ved passende valgt koordinatsystem, viser sig at kunne bruges i alle situationer, hvor bilen befinder sig i afstanden $D$ fra stregen og har en vinkel $\theta$ i forhold til denne.

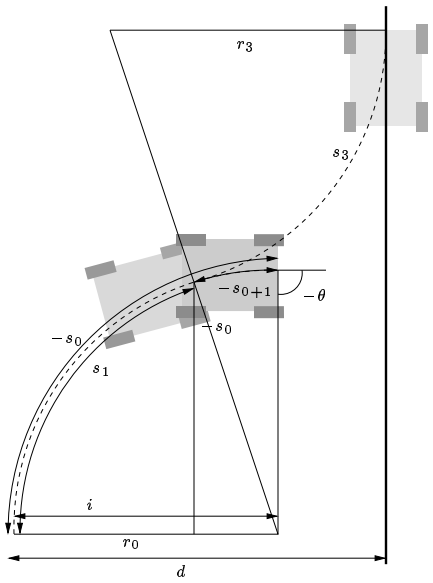
Reglerne gælder naturligvis i tilfældet, hvor bilen er på vej væk fra stregen ($\theta > 0$), da det er udgangspunktet for dem. I tilfældet hvor bilen er parallel med stregen ($\theta = 0$) reduceres (4.37) til

$$s_{0+1} = r(\theta + \cos^{-1}\left(\frac{1}{2} + \frac{1}{2\cos\theta} + \frac{D}{2r}\right) \tag{4.41}$$

$$= r(0 + \cos^{-1}\left(\frac{1}{2} + \frac{1}{2\cos 0} + \frac{D}{2r}\right) \tag{4.42}$$

$$= r\cos^{-1}\left(\frac{1}{2} + \frac{1}{2} + \frac{D}{2r}\right) \tag{4.43}$$

Af (4.32) ses umiddelbart, at $D = d$ for $\theta = 0$ hvor-




Figur 4.12: Bilen er på vej mod stregen, men er for tæt på til at den kan rette op på stregen. Den bakker i stedet afstanden $-s_{0+1}$.

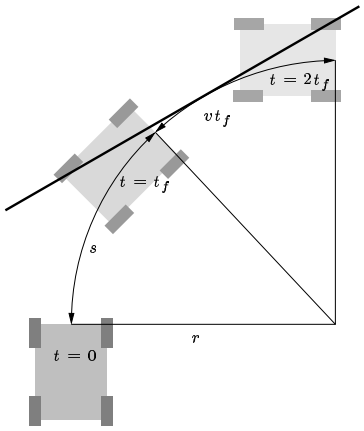
for vi får

$$s_{0+1} = r\cos^{-1}\left(\frac{2r+d}{2r}\right) \tag{4.44}$$

som er ækvivalent med (4.23).

Vælges koordinatsystemet så $\theta$ er negativ når bilen er på vej mod stregen, som vist på figur 4.11 findes, at $s_0$ som er defineret i (4.27) bliver negativ, mens $d$ (4.32) og dermed $s_1$ (4.27) ikke er afhængige af fortegnet af $\theta$. Resultatet er, at bilen, som vist på figur 4.11, følger den optimale vej mod stregen, som blev introduceret i afsnit 4.4.1.

Kommer bilen så tæt på stregen, at det ikke er muligt for den at dreje skarpt nok til at komme op på den, opstår situationen vist i figur 4.12. Her bliver $|s_0| > |s_1|$, og da bilen i denne type situationer er på vej mod stregen, er $s_0 < 0$. Resultatet er, at $s_{0+1}$ bliver negativ, og bilen bakker til den er langt nok fra stregen til, at den kan køre op på den med det sidste køreprimitiv.




Figur 4.13: Bilen er på vej mod stregen. Til tiden $t = 0$ sætter den drejeradius til $r$. Da der bliver taget billede til tiden $t = t_f$, er bilen igen på vej væk fra stregen.

## 4.5 Styresekvenser

En forudsætning for, at teorien beskrevet i de forgående afsnit kan fungere er, at bilen skifter hjulposition på de rigtige steder — eller, hvis bilen er i konstant bevægelse, på de rigtige tidspunkter.

Vi kan vælge kun at ændre een gang på hjulenes retning for hvert billede, der er blevet taget og behandlet. Da er det vigtigt, at den strækning, bilen kører per billede $vt_f$, er væsentligt mindre end de afstande $s_x$, vi har fundet i de forgående afsnit. Her er $v$ den aktuelle hastighed, og $t_f$ er tiden, der går, mellem der bliver taget et billede. Er $vt_f$ for stor i forhold til $s_x$, vil situationen illustreret i figur 4.13 opstå: Til tiden $t = 0$ sætter bilen drejningsradius til $r$. Da det næste billede bliver taget til tiden $t = t_f$, er bilen stadig på vej mod stregen og fortsætter med drejningsradius $r$. Først da det efterfølgende billede er blevet behandlet til tiden $t = 2t_f$, ved bilen, at den skal skifte kurs, men da er den langt fra stregen.

For at undgå dette skal hastigheden sættes ned, så $vt_f$ bliver mindre, eller også skal $r$ sættes op, så $s_x$ bliver større. Det er altså nødvendigt at lave en afvejning mellem fart, og hvor tæt bilen kører på stregen.

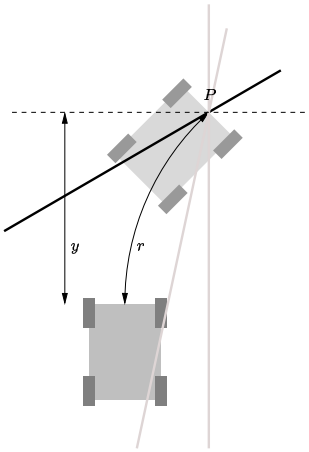
En bedre løsning er at udnytte, at bilen har et indbygget takometer. For hvert billede lægges en køreplan på formen $(r_1, s_1), (r_2, s_2), \ldots, (r_n, s_n)$. Ansvaret for at overholde køreplanen lægges over på en selvstændig proces, som kun skal aflæse takometeret og derved kan reagere meget hurtigt. Ved at bruge styresekvenserne fra afsnit 4.4.4 sikres der, at bilen altid er på vej mod stregen eller er på stregen. Kameraet skal i teorien kun bruges til at justere for stregens forløb. Derved fjernes kravet om, at $s_x \ggg vt_f$, og begrænsningerne på bilens topfart er kun begrænset af, hvor langt bilen kan se frem på banen. Det skal bemærkes, at styresekvenserne i afsnit 4.4.4 er konsistente i den forstand, at bilen ved en ret linje i teorien vil følge den samme bane uafhængigt af, hvor ofte der bliver taget billeder. Eller lidt mere formelt så vil der blive beregnet den samme banekurve for alle punkter på banekurven.

## 4.6 Den implementerede styrestrategi

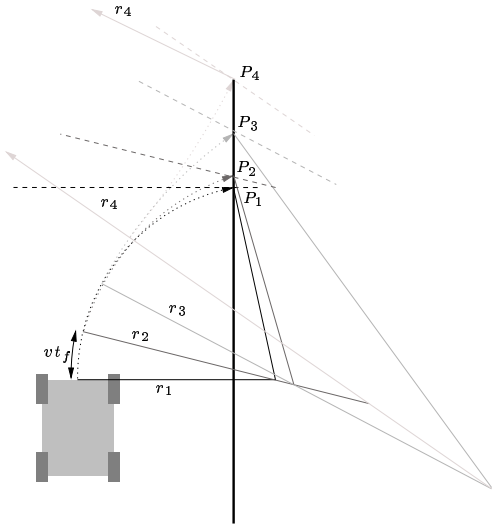
Metoden beskrevet i de forgående afsnit forudsætter en række ting: at bilen kan skifte hjulretning og kørselsretning på et punkt, og at vi kan kontrollere bilens drejningsradius eksakt. Det er bestemt ikke tilfældet for vores bil. Metoden kræver med andre ord justering, før den kan forventes at fungere tilfredsstillende. Da hardwaren først fungerede få dage før RoboCup blev afholdt, måtte vi opgive at få den avancerede strategi til at fungere og valgte i stedet at bruge en mere simpel strategi, som er lettere at implementere og ikke mindst afprøve.

I afstanden $y$ foran bilen lægges en kunstig horisont (se figur 4.14). Liniens skæring med horisonten $P$ vælges som mål for bilen, og hjulene sættes som forskrevet i (4.8).

Denne strategi er i modsætning til den avancerede ikke konsekvent: banekurven vil ændre sig efterhånden som bilen kører. I figur 4.15 har vi




Figur 4.14: I den simple styrestrategi kører bilen mod skæringen mellem stregen og en virtuel horisont (stiplet linje). Denne metode kan bruges uanset om stregen er foran bagved eller parallelt med bilen.




Figur 4.15: En sekvens af 4 korrektioner i henhold til den simple styrestrategi. Banen bilen forsøger at følge er markeret med punkteret streg, de virtuelle horisonter er stiplede. Det bemærkes at styreradierne bliver større jo tættere bilen kommer på stregen. Det fjerde trin har radien $r_4$, skiftet fortegn og er tæt på uendelig, hvilket vil sige at bilen kører tilnærmelsesvis lige ud.

tegnet en sekvens af 4 korrektioner. Da bilen styrer efter at nå stregen efter kørt afstand $\sim y$, vil den, hvis $y > vt_f$, aldrig nå stregen, det sikrer at bilen ikke oscillerer omkring stregen. Selvom den i teorien aldrig vil nå stregen vil den dog altid nærme sig denne. En forudsætning for at strategien virker er at $y \gg vt_f$ hvilket giver anledning til en begrænsning af hastigheden som beskrevet i afsnit 4.5.

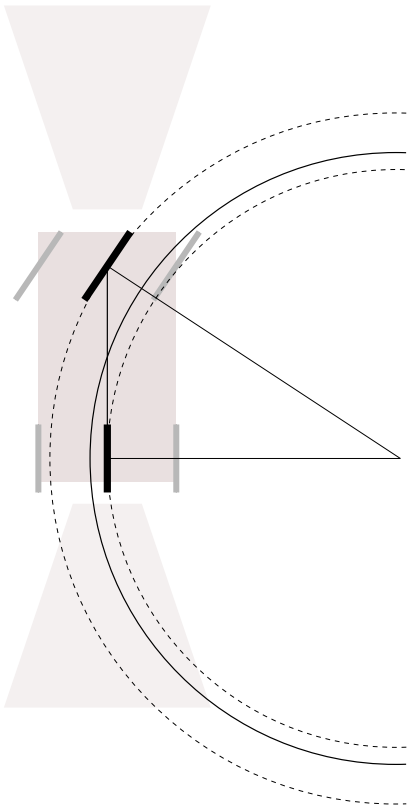
## 4.7 Valg af kørselsretning

Da bilen ved fast hjulposition følger en cirkelbue, er det oplagt, at bilens manøvredygtighed principielt er uafhængig af kørselsretningen. Derimod er udsynet foran bilen stærkt afhængigt af kørselsretningen. Dette har ført til overvejelser om, hvorvidt det er bedst at køre forlæns (lade de styrende hjul sidde forrest) eller baglæns (lade de styrende hjul sidde bagerst) gennem banen.

Det bemærkes, at de styrbare hjul altid vil følge den største cirkel. Det medfører, at den ende af bilen, hvor de styrbare hjul sidder, vil pege svagt ud af cirklen, som bilen følger, og at bagenden vil pege svagt ind i cirklen. Dette er illustreret i figur 4.16. Af figuren ses, at bilen, når den følger en streg gennem en kurve, vil kunne se mest af stregen ved baglænskørsel.

At vi under RoboCup har valgt at lade de styrende hjul sidde forrest i bilens kørselsretning skyldes udelukkende, at bilen tilsyneladende er optimeret til at køre forlæns. Med læs på bilen havde de styrende hjul en tendens til at dreje sig mod hinanden. Denne effekt blev forværret ved baglænskørsel og forøgede samtidig bilens rullemodstand så kraftigt, at motoren havde svært ved at trække bilen. Derudover havde de styrende hjul en tendens til at 'rette op' ved baglænskørsel, hvilket betyder, at hjulene ikke

kan drejes maksimalt, og styreegenskaberne forringes.



Figur 4.16: Kameraets synsfelt forlæns hhv. baglæns er markeret med en grå trapez. Det fremgår, at bilens udsyn over den bane, den følger, er bedst ved baglænskørsel.

# Kapitel 5

# Faldgrupper

## 5.1 Thorkild Thyrring

Racerkøreren Thorkild Thyrring har umiddelbart intet med autonome robotter at gøre, men han er afgjort forbundet med fart. Grunden til, at han her alligevel bliver blandet ind i projektet, er, at vi på et tidspunkt forsøgte os med en strategi, der var god på papiret, men i praksis hurtig blev døbt "Thorkild Thyrring styring" og skrottet på grund af uforudsete problemer.

Ideen var, at hvis robotten stødte på et sving, der var meget skarpt, ville tapestregen hurtigt forsvinde ud af synsfeltet, dels fordi vi har valgt at styre med forhjulene og at synsretningen derfor, som tidligere nævnt, ville pege lidt ud af den cirkel, vi følger, og dels fordi svingets radius kunne være mindre end robottens svingradius. Derfor skulle robotten udføre en modificeret trepunktsvending. Dvs. at den skulle stoppe, dreje hjulene modsat svingets retning, bakke og derefter dreje i svingets retning igen og køre frem, når tapen igen kom ind i synsfeltet. For at udføre manøvren med rimelig hastighed, var det nødvendigt at stoppe og slå forholdvist hårdt bak. Det havde dog den uheldige effekt, at det tilsyneladende gav en elektromagnetisk puls, der "dræbte" takometeret i et par sekunder. Det skal lige siges, at vi ikke med sikkerhed ved, at det var dette, der var skyld i problemet, men efter megen eksperimenteren fik vi udelukket alt andet, vi kunne finde på. Resultatet af det døde takometer var, at robotten troede den stod stille og derfor blev ved med at skrue op for hastigheden, som selvfølglig hurtigt blev meget høj og truede robottens (og folk og deres robotter i nærhedens) sikkerhed.

Den høje hastighed betød dermed, at robotten pludselig forlod banen med relativ høj hastighed, og herfra stammer navnet "Thorkild Thyrring styring". Vi mener dog stadig, at bakkestrategien kunne være fordelagtig, hvis man får løst problemet med takometerets spontane dødsfald.

## 5.2 Underdimensioneret relæ

Robottens evne til at køre henholdsvis frem og tilbage afhænger af et relæ på MacGyverkortet. Dette relæ viste sig, i løbet af natten op til løbet og på selve løbsmorgenen, at være en anelse underdimensioneret (beregnet til 5 A). Pga. den hårde belastning opstod der en belægning i relæet der gjorde at relæet ikke kunne klikke over, og robotten kunne kun køre baglæns. Det kunne dog klares ved at åbne relæet og file på det med et målebånd. Ved videre arbejde med robotten skal man være opmærksom på et det kan (vil) blive et problem igen.

## 5.3 Takometer

Takometeret er som sagt lavet af de små hulskiver der i en pc-mus måler, hvor meget musen bevæger sig i ticks. Det store problem har været at fast-

sætte disse hulskiver samt foto- og lysdiode der måler hvor hurtigt hulskiverne bevæger sig. Den endelige løsning bruger lim til at sætte et takometer på bagakslen ved hvert hjul, men alligevel kunne vores målinger tyde på at det ene takometer er faldet af eller sidder meget dårligt fast. Et andet problem kan være at takometeret i musen ikke er af en kvalitet der kan klare alt for store hastigheder. Det burde dog være istand til at klare den moderate hastighed vi kørte med under finalen.

# Kapitel 6

# Programbeskrivelse

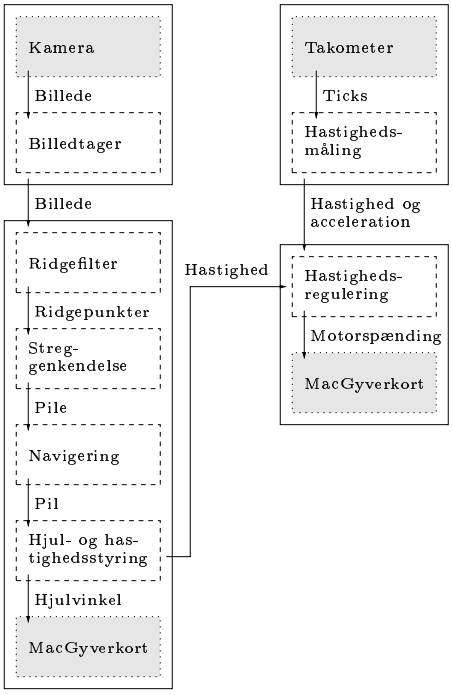
En grundig tilbundsgående beskrivelse af, hvordan det faktiske program er implementeret, inklusive datastrukturer ville blive voldsomt stor og desuden være af begrænset interesse.

## 6.1 Dataflow

Vi valgt at give et overblik over programmets logiske komponenter, samt dataflowet i programmet. Se figur 6.1.

Programmet er opdelt i 4 tråde, som kører parallelt. Disse kommunikerer sammen vha. delt hukommelse. De steder hvor det er relevant benytter vi semaforer til at sikre at data er konsistent. F.eks. sikrer to semaforer at et billede først bliver vidrebehandlet når det er helt indlæst og at der højst er et ubehandlet billede i bufferen.

Programmet er skrevet i C og en udskrift af programmet kan ses i bilag A.




Figur 6.1: Dataflow i programmet. Fuldt optrukne kasser repræsenterer tråde (threads), stiplede kasser repræsenterer logiske programkomponenter og kasser med grå baggrund repræsenterer hardware. Pilene angiver retningen af dataflowet.

## 6.2 Anvendt programmel

Til implementeringen af programmet har vi, hvor det kunne lade sig gøre, anvendt allerede eksisterende komponenter.

Intel Integrated Performance Primitives

Som hjælp til implementeringen af billedbehandlingen har vi anvendt Intels Integrated Performance Primitives (IPP) [4]. IPP kan bruges gratis, såfremt det ikke bruges til kommercielle formål. IPP er en samling af lavniveaufunktioner til billed- og signalbehandling optimeret specielt til Intels processorer (IA32 og Strong ARM).

GTK+

GTK+ [3] er en bibliotekssamling til implementering af grafiske brugerflader, som vi anvender til vores GUI-program, se afsnit 3.4. GTK+ indeholder desuden en samling datastruktur-primitiver, hvoraf vi anvender en dobbelthægtet liste. GTK+ er frit software.

Linux

Som operativsystem på den bærbare computer anvender vi Linux [7], som er et frit Unix og POSIX kompatibelt operativsystem udviklet af Linus Torvalds og mange andre.

Sammenhængende komponenter

Til at finde sammenhængende komponenter bruger vi en funktion implementeret af Rasmus Friis Kjeldsen til et tidligere kursus [9].

Vektorprimitiv

Som vektorprimitiv har vi videreudviklet på et eksisterende vektorprimitiv, udviklet til et tidligere kursus af Martin Leopold, Lars Bo Thulin og Rasmus Friis Kjeldsen [8].

CVS

Under programudviklingen brugte vi versionskontrolsystemet Concurrent Versions System [1] til at håndtere kildefilerne. CVS er frit software.

# Kapitel 7

# Beskrivelse af vedlagte CD

På den vedlagte CD (Bilag B), har vi lagt følgende:

- *dtufinale1_org.mpg* og *dtufinale1_filter.mpg*:
  Film sammensat af hhv. billederne optaget under første finalekørsel i RoboCup 2002, og de filtrerede billeder incl. visualiseringen af navigationsfunktionerne beslutninger. Visualiseringen er beskrevet i afsnit 3.3.6.

  *dtufinale2_org.mpg* og *dtufinale2_filter.mpg*:
  Som ovenstående, men fra 2. gennemløb.

- Mapperne *dtufinale1/* og *dtufinale2/*:
  Disse indeholder de billeder, som ovenstående film er sammensat af.

- Mappen *program/*:
  Indeholder kildekoden til programmet.

# Kapitel 8

# Konklusion

Den billedbehandlingsmæssige side af projektet må siges at være vellykket. Robotten er i stand til at detektere tapestreger, med en robusthed der muliggøre tilfredsstillende navigation på RoboCup-banen. Dette blev ved projektets start betragtet som den største udfordring pga. de vanskelige lysforhold på banen.

Grundet det nævnte tidspres har vi dog kun kunne nå at implementere den simple styrestrategi. Hvilket måske er den vigtigste grund til at robotten ikke kunne gennemføre banen.

Dertil kommer, at vi ikke har kunne nå at få monteret afstandssensorene. Vi var derfor nødt til at opgive at lade robotten følge den nævnte væg og dermed også opgive en banegennemkørsel med fuldt pointtal. Grundet den simple styrestrategi var vi desuden nødsaget til at vælge den simpleste rute - at holde til højre.

Under første finaleløb kom vi gennem 3 porte inden vores, lidt for store, robot snittede den fjerde port. Under andet og sidste finaleløb gik det lidt bedre, og vi kom gennem 4 porte inden robotten ramte den 5. port.

Resultatet af RoboCup 2002 var, at vi kom ind på en 12. plads ud af i alt 16 deltagere. Med den tid vi havde til rådighed mener vi, at resultatet var rimeligt. Samtidig er vi dog ikke i tvivl om, at man ved at implementere den mere avancerede styrestrategi, og ved brug af afstandssensorer, kunne få en langt bedre placering.

Det vil også være en god ide at udstyre webkameraet med en optik med mindre brændvidde, eller bruge et andet webkamera. Det begrænsede "tunnelsyn" var en anden vigtig hindring i at gennemføre banen. Når robotten fulgte en skarp kurve forsvandt stregen ud af synsfeltet og den kørte i blinde til den var ude af kurven. Dette ses tydelige på de vedlagte film i bilag B.

# Litteratur


[1] cvshome.org. *Concurrent Versions System.* http://cvshome.org.

[2] Danmarks Tekniske Universitet. *DTU RoboCup: robotternes tumleplads.* http://www.robocup.dtu.dk/.

[3] gtk.org. *GTK+ - The GIMP Toolkit.* http://gtk.org.

[4] Intel. *Intel Integrated Performance Primitives.* http://www.intel.com/software/products/ipp/index.htm.

[5] Johnson Electric S.A. *Motor Reference Guide, HC683G.* http://www.johnsonmotor.com/downloads/-pdf/remote_control_car.pdf.

[6] Tony Lindeberg. Edge detection end ridge detection with automatic scale selection. *Int. J. of Computer Vision*, 30, 1998.

[7] Linus Torvalds. *Linux.* http://kernel.org.

[8] Martin Leopold, Lars Bo Thulin og Rasmus Friis Kjeldsen. *Vektorprimitiv.* http://www.diku.dk/-research-groups/image/lab/robots/eyebot/programmer/leopoldRazzleLarsbo/listVector/.

[9] Rasmus Friis Kjeldsen. *Sammenhængende komponenter.* http://www.diku.dk/research-groups/image/-lab/robots/eyebot/programmer/leopoldRazzleLarsbo/hsiCcl/.

[10] Søren I. Olsen. *Forelæsningsnoter til introduktion til digital billedbehandling.*