Technical University of Denmark DTU

*Kasper Grue Understrup*

# Laserbaseret forhindringsundvigelse

Bachelor thesis, January 2011

DTU Electrical Engineering
Department of Electrical Engineering

**Laserbaseret Forhindringsundvigelse, Laser Based Obstacle Avoidance**

**Report written by:**
Kasper Grue Understrup

**Advisor(s):**
Jens Christian Andersen
Nils Axel Andersen

**DTU Elektro**
Technical University of Denmark
2800 Kgs. Lyngby
Denmark

studieadministration@elektro.dtu.dk

| | |
|---|---|
| Project period: | 2010.09.01 – 2011.01.31 |
| ECTS: | 20 |
| Education: | B. Science |
| Field: | Electro technology |
| Class: | Offentligt |
| Edition: | 1. edition |
| Remarks: | This report is submitted as partial fulfillment of the requirements for graduation in the above education at the Technical University of Denmark. |

Copyrights: © Kasper Grue Understrup, 2011

## Abstract

This report presents a reactive solution to obstacle avoidance based on a laser scanner. The report will first describe existing solutions to the problem and compare them to one another. Based on the knowledge of these algorithms, I will synthesize a new algorithm. This algorithm is capable of moving the robot past simple objects like a doorway, a box but also complex scenarios like many boxes, traps, tables and chairs. I describe how the algorithm works, and how the laser scanner on the robot is used. Next I will perform a number of tests with the robot in different scenarios, which is to be seen in the attached video material.

## Resumé

Denne rapport arbejder med en reaktiv løsning til ”obstacle avoidance” (forhindringsundvigelse) baseret på en laserskanner. Rapporten kigger først på eksisterende løsninger på problemet, samt på deres fordele og ulemper. Med baggrund i kendskab til disse algoritmer, syntetiseres en ny algoritme. Denne algoritme kan manøvrerer robotten udenom simple objekter som en døråbning eller en kasse, men også komplekse scenarier med mange kasser, fælder, borde og stole. Jeg beskriver hvordan algoritmen virker, og hvordan robottens laserskanner benyttes. Jeg foretager dernæst en række test af robotten i forskellige scenarier, som fremgår af det vedlagte videomateriale. Til sidst samles der op, for at konkluderer, om denne algoritme kan bruges i praksis.

# Forord

Før jeg begyndte på dette projekt havde jeg kun lidt erfaring med C programmering, og intet kendskab til C++. Pluginet er derfor skrevet i en C lignende C++ stil. Selvom jeg havde arbejdet med DTU's robotter før, var opbygningen af serverne, klienterne og pluginene ny for mig. Der er derfor gået en del tid med at blive fortrolig med systemet. Jeg føler selv, at dette projekt har lært mig utroligt meget, og jeg er tilfreds med outputtet af projektet.

Jeg har taget mig den frihed at dreje alle koordinatsystemer i rapporten 90 grader, da robotten altid starter med fronten i x-aksens retning, og robottens lokale koordinatsystem altid kigger ud af x-aksen. Jeg synes det visuelt giver bedre mening hvis fremad for robotten, er opad på papiret.

# Table of Contents

Abstract ....... 1
Resumé ....... 2
Forord ....... 3
Kap 1 – Introduktion ....... 5
1.1 Evt. Begrænsninger og forbehold ....... 6
1.2 Problemformulering ....... 6
1.3 Oversigt over rapporten ....... 6
Kap 2 – Analyse ....... 7
2.1 Kendte algoritmer til obstacle avoidance (State of the art) ....... 7
Visibility Graph ....... 7
BUG algoritme ....... 7
Potential Field ....... 7
Nearness Diagram (ND) ....... 8
Vektor Field Histogram (VFH) ....... 8
Dynamic Window Approach ....... 8
The Bubble Band Technique ....... 9
2.2 Beskrivelse af robotten ....... 9
2.3 Diskussion af algoritmerne og valg af algoritme ....... 10

Algoritmen ... 10
2.4 Pseudokode implementering af algoritmen ... 12
2.5 Diskussion af algoritmen ... 13
Kap 3 – Teori ... 15
3.1 Bearbejdning af laserscannerdata ... 15
Objektundersøgelse ... 16
Spring i laserskannet ... 17
Hulgenkendelse ... 19
3.2 De 3 koordinatsystemer ... 20
3.3 Kørselsstrategi ... 20
3.4 Algoritmen i detaljer ... 21
Kap 4 - Implementering og test ... 23
4.1 Testkørsel af algoritmen ... 23
4.2 Diskussion af algoritmen ... 24
4.3 Brugervejledning ... 25
4.4 TODO-liste ... 26
Konklusion ... 27
Perspektivering ... 27
Referenceliste ... 27
Appendiks A ... 29

# Kap 1 – Introduktion

Der er flere steder hvor mobile autonome robotter gør sit indtog, fx i husholdningen og på arbejdspladsen. Det stiller dog krav til hvordan robotten manøvrerer i dynamiske og komplekse miljøer. Hvad gør man fx når der står en kasse i vejen for den planlagte rute? Kører man den ene eller den anden vej rundt om? Eller ved man overhovedet at kassen er der og derfor kører ind i den. For at løse problemet bliver robotten nødt til at vide, hvad den står overfor, og hvor det er muligt for den at køre hen. Jeg vil referere til dette problem som mobilitetsproblemet. Robotten bliver derfor nødt til via dens sensorer at danne sig et billede af verden, for derefter at kunne tage en beslutning. Metoder til dette kaldes ”sensor based motion planning methods” eller ”reactive navigation methods” (reaktiv navigations metode). Reaktiv navigation er det eneste robuste alternativ i disse miljøer for at takle mobilitetsproblemet. Det at udfører en på forhånd fastsat bane, er ikke realistisk i dynamiske miljøer, hvor robotten er i fare for at støde ind i uforudsete objekter. Rene reaktive løsninger har dog det problem, at der ikke indgår global information i beslutningstagningsprocessen.

Man kunne ved en fejl forledes til at tro, at der findes algoritmer som optimalt løser problemet, da en algoritme som Visibility Graph, der er beskrevet senere, altid får robotten til at kører den korteste vej. Problemet er, at denne algoritme skal have hele kortet tilgængeligt på forhånd, og omgivelserne ikke må være dynamiske. Man kan derfor dele navigation op i to hovedgrupper: ”Motion planning” eller ”path planning”, som er baseret på kendte globale informationer, og reaktiv navigations metode, som er baseret på lokal information fra sensorerne. Der er også eksempler på hybride løsninger, hvor en reaktiv navigations metode er smeltet sammen med en stifinder (”path planner”), som i Global Nearness Diaggram eller the global Dynamic Window Approach. Disse algoritmer har den fordel, at robotten har nemmere ved at komme ud af fælder. Jeg vil dog ikke komme nærmere ind på dette område.

En algoritme siges at være komplet, hvis algoritmen altid får robotten til at kører fra A til B, givet at der er en vej fra A til B. Dette betyder ikke, at robotten altid kører den korteste vej til målet eller at den i praksis altid vil finde frem. Det betyder, at robotten i en ideel verden altid vil finde frem uanset hvilken forhindring robotten står overfor.

Det er svært at lave en algoritme som optimalt løser mobilitetsproblemet og samtidigt undgår at kunne blive fanget i fælder. Fælder er fx U-lignende objekter, som udgør et lokalt minimum for afstanden til målet. Hvis robotten er blevet fanget i et lokalt minimum, bliver den nødt til at kører i modsat retning af dens mål for i sidste end at nå frem til målet.

## 1.1 Evt. Begrænsninger og forbehold

Jeg beskriver ikke, hvordan algoritmen i praksis er implementeret, og der indgår ikke kodeeksempler andet end det som fremgår af bilagene og af Cd'en, hvor hele koden er vedlagt. Jeg beskriver heller ikke i detaljer hvordan robotten og softwaren er opbygget. Alt dette mener jeg er uinteressant for læseren, som muligvis arbejder på en anden platform. Jeg begrænser mig også til 2D, da dette gør problemet meget enklere, og fordi laserskanneren kun ser i 2D.

## 1.2 Problemformulering

Projektet skal kunne få en Small Mobile Robot (SMR) til selv at kunne køre fra punkt A til punkt B selvom der er et objekt i vejen. Projektet skal give brugeren af SMR'erne en mulighed for at benytte forhindringsundvigelse fra et SMRCL-script. Implementeringen af algoritmen skal være tilgængelig for brugeren, uden at brugeren skal kende til selve algoritmen. Løsningen skal simpel nok til, at problemet kan løses indenfor et bachelorprojekt, og således at implementeringen kan køres på SMR'erne. Projektet skal kunne indgå som en del af DTU's robotsoftware. I projektet vil der ikke indgå en global stifinder, men derimod en reaktiv navigationsmetode, der kan få SMR'en til at passerer simple forhindringer som en kasse eller en døråbning.

## 1.3 Oversigt over rapporten

Efter denne introduktion til emnet, vil jeg i kapitel to gennemgå algoritmerne som udgør "state of the art" indenfor forhindringsundvigelse. Derefter giver jeg en kort introduktion til robotten der arbejdes med. Dernæst vil jeg forklarer om min egen algoritme til forhindringsundvigelse, og belyse fordele og ulemper for denne algoritme. Kapitel tre indeholder Teorien som indgår i algoritmen. Først er der en del om hvordan laserskannerdata benyttes, og hvordan jeg har valgt at arbejde med de forskellige koordinatsystemer. Kapitel fire viser testresultater for algoritmen, og hvilke forhindringer algoritmen kunne klare.

# Kap 2 – Analyse

## 2.1 Kendte algoritmer til obstacle avoidance (State of the art)

Forhindringsundvigelse betyder, at man ændrer på robottens bane, for at få robotten til at køre uden om forhindringer. Der findes adskillige algoritmer til at håndterer forhindringsundvigelse. Disse har hver især deres fordele og ulemper, og der derfor ikke er et entydigt svar på, hvilken man bør vælge. Selvom problemet er det samme for alle algoritmerne, er det vidt forskelligt hvordan problemet bliver grebet an af algoritmen, og hvordan verden opfattes. Algoritmerne har til opgave at foretage beslutninger på baggrund af indsamlede data, og fungerer derfor som den kognitive del af softwaren. Her er mit bud på ”state of the art”:

### Visibility Graph

Metoden benytter sig af graf teori og er komplet. Visibility Graph[1] kan bruges som stifinder og benytter sig af et på forhånd kendt kort. Algoritmen går ud på, at indføre strategiske punkter på kortet som knuder i en graf og synlige forbindelser mellem knuder som kanter. De strategiske punkter er: startpunkt, mål og hjørner på polygoner som repræsenterer objekter, disse polygoner har fået tillagt en hvis sikkerhedsmargin. Kanterne imellem knuderne er synlige forbindelser imellem punkterne, hvor robotten kan kører imellem. Når grafen er dannet kan Dijkstra's algoritme[2] bruges til at finde den korteste vej fra A til B. Der er to ulemper ved denne algoritme. Den ene er, at robotten kommer til at kører tæt op ad objekterne, hvilket kan resulterer i kollision. Det andet problem er, at hvis der kommer mange objekter i spil, kan algoritmen blive tung at køre, med kørselstid på $O(n^2)$.

### BUG algoritme

BUG[3] algoritmen er den simpleste algoritme til forhindringsundvigelse. Den er komplet, simpel at implementere og håbløst ineffektiv. BUG algoritmen er en reaktiv navigations metode. Ideen er, at man kører direkte mod målet indtil man møder et objekt. Derefter følger man objektet indtil man finder det punkt, som ligger tættest på målet. Når punktet er fundet køres der hen til dette punkt hvorefter der fra dette punkt køres direkte mod målet. Enten når man målet eller også gentages algoritmen. Problemet med denne algoritme er, at robotten højst sandsynligt kommer til at kører meget længere end det er nødvendigt for at nå frem til målet. Hvis første forhindring er en væg, vil robotten følge væggen rundt i hele bygningen igennem alle rum før den finder det nærmeste punkt. Hvis problemet er en klassisk labyrint med én indgang og én udgang, er denne algoritme klart at foretrække, da den optimalt løser problemet. En klassisk labyrint kan altid løses ved at følge en væg. Der er flere udgaver af denne algoritme, hvor Tangent Bug er den mest effektive.

### Potential Field

Potential Field[4] algoritmen er en reaktiv navigations metode. Algoritmen udregner et potentialfelt over hele kortet som skal dirigerer robotten hen imod målet. Dvs. der til et hvert punkt knyttes en gradient. Robotten

opfattes som et punkt, og forhindringer opfattes som toppe, som frastøder robotten. Man kan forestille sig udgangspositionen som toppen af en bakke og målet som bunden af bakken, og robotten så er en kugle som triller fra toppen mod bunden. Da man ikke nødvendigvis har hele kortet på forhånd opdateres det potentielle felt som robotten kører. Algoritmen er ikke komplet, da kuglen kan blive fanget i lokale minima, som skabes af fx U-formede forhindringer.

### Nearness Diagram (ND)

Nearness Diagram[5] er en reaktiv algoritme og fungerer efter del og hersk princippet. Ideen er, at alle tænkelige omgivelser robotten kan befinde sig i, kan inddeles i 5 scenarier, som igen er inddelt i to hovedgrupper. De to hovedgrupper er ”high safety”, som betyder, at der ikke er et objekt tæt på robotten og ”low safety”, som betyder, at der er et objekt tæt på robotten. Hvis robotten fx befinder sig i ”low safety”, er der to scenarier, et hvor objektet er på den ene side, og et hvor objektet er på begge sider. I ”high safty” er der 3 scenarier, et hvor robotten kan køre direkte til målet, og to hvor robotten ikke kan køre direkte til målet. Algoritmen er primært god til snævre steder, og knap så effektiv på åbne områder som gange, da algoritmen har en tendens til at få robotten til at slingre og lave vrikkende bevægelser. Algoritmen er primært beregnet til runde robotter.

### Vektor Field Histogram (VFH)

Vektor Field Histogram[6] er en reaktiv algoritme, hvor der laves et polært histogram med vinklen ud ad x-aksen og sandsynligheden for at det er en forhindring ud ad y-aksen. Derefter udregnes en prisfunktion hvor målets retning, hjulenes orientering og robottens forrige retning indgår. Prisfunktionen giver fx en stor pris hvis man kører væk fra målet og en mindre pris hvis man kører hen imod målet. Ud fra prisfunktionen besluttes hvilken vej robotten skal køre, idet der altid vælges den vej, der har lavest pris.

### Dynamic Window Approach

I Dynamic Window Approach[7], foregår styringen vha. hastigheder. Algoritmen sammenstykker robottens rute ved sekvenser af banesegmenter. Der dannes først et hastighedsrum, hvor hvert banesegment repræsenteres ved en hastighedsvektor, som robotten vil kunne opnå i den givne bane. Der tages her højde for, at robotten også skal kunne bremse før den rammer objekter. Hastigheden, robotten kan opnå på et banesegment, afhænger derfor af robottens nuværende hastighed og på afstanden til et eventuelt objekt, som befinder sig på banesegmentet. Alle banesegmenter, som robotten kan nå at stoppe før den rammer, behandles af algoritmen. Da algoritmen prøver at få robotten til at kører så hurtigt som muligt, er det nødvendigt at kører algoritmen relativt ofte. Algoritme henvender sig til robotter som bevæger sig hurtigt igennem dynamiske miljøer som fx en arbejdsplads.

### The Bubble Band Technique

The Bubble Band Technique[8] kan være reaktiv, hvis algoritmen udføres mens robotten køre, men det er ikke en decideret forhindringsundvigelsesalgoritme. Algoritmen skal benyttes i forlængelse af en stifinder for at give robotten bløde bevægelser. The Bubble Band Technique udspringer af Elastic Band algoritmen, som arbejder med tiltrækkende og frastødende elementer. Ideen er, at danne et kort af bobler på robottens rute. En bobbel dannes ved at ekspandere en cirkel indtil den rammer et objekt. Robotten skal derefter kører igennem midtpunktet af hver bobbel. The Bubble Band Technique er ikke i sig selv komplet.

## 2.2 Beskrivelse af robotten

Robotten, der arbejdes på, ejes af DTU og disponerer over en laserscanner, differentiel motorstyring, et trådløst netværkskort og enkodere på hjulene med mere.

Laserskanneren, som er af typen Hokuyo URG-04LX, kan levere data om objekter der befinder sig mellem 2cm og 4 meter væk fra laserskanneren i vinkler på mellem -120 og 120 grader. Opløsning er på 0,351562 grader/punkt. Da laserscanneren er placeret imellem forhjulene på robotten, er synsfeltet i praksis indskrænket til -90 til 90 grader.

Robottens dimensioner er 30cm i bredden og 30cm i længden. Robotten drejer om midten af bagakslen. Robotten holder sig indenfor en afstand af 33,5cm fra dette punkt. Jeg vil referere til disse 33,5cm som robottens radius.

Uden at gå i detaljer omkring styringen af robotten, kan det nævnes, at robotten styres af kommandoer fra et SMRCL-script. Scriptet kan kalde forskellige hjælpefunktioner fra forskellige servere som robotten kan køre. Kommunikationen mellem laserscannerserveren og scriptet kan kun foregå igennem 10 tal af typen double. Implementeringen af algoritmen foregår derfor i to dele. Delen hvor algoritmen er implementeret: Laserserveren, og delen som styrer robotten vha. Kommandoer: Scriptet. Implementeringen af algoritmen styrer derfor kun indirekte robotten. Der kan godt være en forsinkelse fra at skriptet kalder en funktion fra laserserveren til at svaret bliver modtaget via doubleværdier. For at tjekke om data er opdaterede fra laserskannerserveren, bruger jeg en af de ti doubleværdier til at tjekke, om der er modtaget nye data. Tre andre værdier bruger jeg til styringen af robotten. Jeg bruger en værdi til at styre, hvor meget robotten skal starte med at dreje, en værdi til hvor langt den efterfølgende skal køre lige ud og en værdi til efterfølgende at få robotten til at dreje hen imod målet. Jeg bruger også en værdi til at tjekke, om robotten er nået frem til målet, og derved om algoritmen skal kaldes igen. De resterende værdier er uinteressante for læseren, da jeg personligt bruger dem til at ”debugge” programmet. For at benytte pluginet og derved min implementering af forhindringsundvigelse, skal brugeren derfor benytte 5 doubleværdier i skriptet, som modtages af laserskannerserveren.

## 2.3 Diskussion af algoritmerne og valg af algoritme

Forhindringen som robotten skal passere er ikke veldefineret, men i og med at forhindringen kan være en døråbning, er BUG algoritmen udelukket. Hvis forhindringen kun var kasser eller en labyrint ville jeg have valgt denne algoritme. Potetial Field algoritmen er en spændende algoritme, som jeg gerne ville have brugt, men som ikke er velegnet på DTU's robotsoftware. Dette skyldes at algoritmen fungerer ved kontinuert at følge gradienterne i punkterne som robotten besøger. DTU's robotter fungerer ved at de modtager enkelte kør eller drej kommandoer, og derfor er denne algoritme ikke optimal. Det samme gør sig gældende for Dynamic Window Approach, som helst skal køres hvert 0,25 sekund, ved hastigheder på 95cm/s. Bubble Band Tecknicken kan kun fungere sammen med en stifinder, som jeg ikke har til rådighed, så denne algoritme har ikke nogen interesse. ND og VFH algoritmerne er begge interessante algoritmer, som med garanti ville kunne løse problemet med en kasse eller en døråbning. Disse algoritmer er gennemprøvede og virker, men gør efter min mening problemet sværere end det er. Fx Vektor Field Histogram, som først skal udregne et Polar histogram, hvorefter der skal udregnes en prisfunktion, for at finde ud af, at få robotten til at køre venstre eller højre om kassen. Dette er efter min overbevisning spild af ressourcer og kan gøres enklere. Nearness Diagram vil nok i virkeligheden være det bedste bud på en algoritme, som kan klare udfordringen med en kasse eller døråbning, og samtidigt effektivt blive implementeret på DTU's robotsoftware. Jeg har dog ikke valgt at implementere denne algoritme, da denne algoritme er gennemprøvet og gennemtestet. Jeg har derfor besluttet mig til vha. ideer fra de andre algoritmer, at danne min egen algoritme til forhindringsundvigelse.

### Algoritmen

Gennem diskussion med mine vejledere er jeg kommet frem til følgende strategi for algoritmen. Algoritmen skal kigge efter spring i laserscannet, hvis der ikke kan kører direkte til målet. Et spring er defineret som to på hinanden efterfølgende punkter i laserscannet, som har en afstand mellem sig på mere end robottens radius.

Til ethvert spring defineres et tilhørende hul. Hullet hørende til springet A,B er liniestykket G,H med følgende egenskaber:

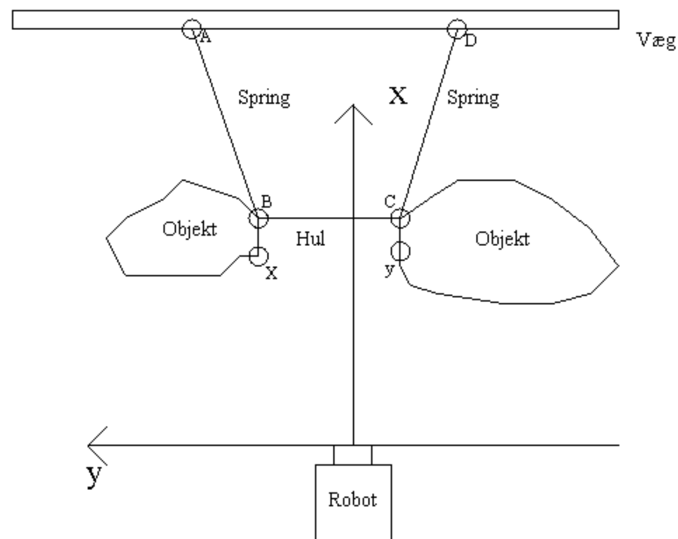
- G og H er punkter fra laserskannet.

- Det ene af punkterne G og H ligger i mængden {A,B}.

- Det andet af punkterne G og H ligger på den modsatte side af springet.

- Længden fra G til H er mindst mulig under iagtagelse af ovenstående begrænsninger.

Som specialtilfælde kan hullet hørende til et spring A,B være liniestykket A,B selv.



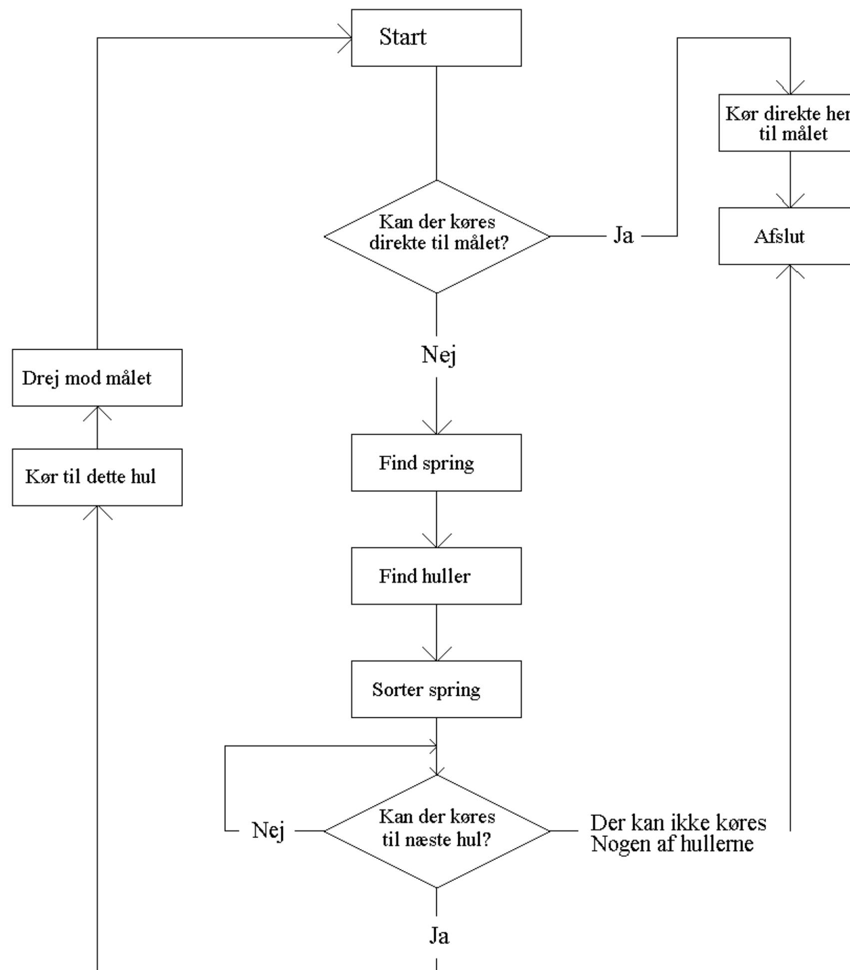

Figur 1. Figuren viser hvordan der findes spring og huller i laserskannet. På figuren ses to spring og to huller.

På figuren ovenfor ses to spring (A,B) og (C,D), som dannes af mellemrummet mellem objekterne og væggen. Disse spring dannes fordi to punkter, som ligger lige efter hinanden i laserskannet, har en afstand mellem sig, som er større end robottens radius. Ud fra disse spring dannes to huller: (B,C) og (B,C), som i dette tilfælde er sammenfaldende. Hvert hul er her det kortest mulige linjestykke, hvor hullets punkter ligger på hver side af springet, og hvor en af punkterne fra springet indgår. Linjestykket (X,Y) ville aldrig kunne være et hul, fordi ingen af punkterne indgår i et spring. Linjestykket (B,Y) er hul i stedet for (B,C) hvis:

$$|(B,Y)|<|(B,C)|$$

Hvis dette er tilfældet, og C i forhold til B er det korteste linjestykke til springet (C,D) , er de to huller hørende til springende (A,B) og (C,D) henholdsvis: (B,Y) og (B,C).

Alle hullerne sorteres nu efter hvilket der ligger tættest på målet, og de huller som er for små til robotten slettes. Der køres til det hul som ligger tættest på målet, som det er muligt at køre hen til. Derefter drejes således at robotten peger hen imod målet.




Figur 2 – Her ses et metodediagram over algoritmen.

På figur 2 ses første forsøg på algoritmen. Ud fra robottens position afgøres det, om robotten kan køre direkte til målet. Hvis den kan dette, køres der til målet, og hvis ikke, findes der et delmål (ny position) at køre til. Delmålet findes ved først at finde spring og dernæst huller i laserskannet. Disse huller sorteres efter hvilket der ligger tættest på målet. Hvis der kan køres til dette hul, køres der til dette delmål, og ellers tages det næste hul i rækken. En gentagelse af algoritmen giver derfor en ny position, enten ved at der køres hen til målet, eller ved at der køres til et delmål. Hvis robotten er blevet fanget i en fælde, dvs. At der ikke kan køres til nogen af de eventuelle huller, stopper algoritmen, uden at positionen ændrer sig.

## 2.4 Pseudokode implementering af algoritmen

While robotten ikke har nået målet og ikke er afsluttet
Drej robotten så den peger mod målet
Undersøg om robotten kan kører direkte til målet
If robotten kan køre direkte til målet

```
                        Kør til målet
                        Afslut
            End if
            Hvis robotten ikke kan køre direkte til målet
            Finde huller som robotten kan kører igennem
            Sorter disse huller efter hvilket der ligger tættest på målet
            for alle huller startende med det nærmest beliggende til målet
                        Undersøg om det er muligt at køre over til hullet
                        Hvis det er muligt at kører over til hullet
                                    Kør til hullet
                        End hvis
            End for
            Hvis det ikke er muligt at kører over til nogen huller
                        afslut
            End hvis
End while
```

## 2.5 Diskussion af algoritmen

Fordelen ved algoritmen er, at den er forholdsvis nem at implementere, og at den burde kunne få robotten til at finde vej rundt om simple objekter. Ulempen ved algoritmen er, at den ikke er komplet. Hvis robotten er kørt ind i en åbning hvor den ikke kan se et hul, stopper algoritmen. Algoritmen fungerer som en forsimplet udgave af Vektor Field Histogram og Nearness Diaggram. VFH fordi algoritmen hele tiden forsøger at tage den vej som ligger tættest muligt på målet. Dette gøres ikke ved en prisfunktion, men ved at sorterer i mulige veje robotten kan køre, og tage den som ligger nærmest målet. ND fordi der i stedet for 5 scenarier er 2 scenarier uden hovedgrupper. Scenariet hvor der kan køres direkte til målet, findes også i ND og de to andre scenarier i ”high safety”, er i min algoritme blevet slået sammen til et scenarie. ”Low safety” er ikke implementeret i min algoritme, da der tjekkes for, om robotten kører ind i objekterne. Min algoritme har den store fordel i forhold til ND, at robotten ikke laver vrikkende bevægelser, og den fungerer på åbne områder.

Der er brug for flere funktionaliteter for at få algoritmen til at virke. Først og fremmest er det nødvendigt at kunne finde huller og åbninger via en laserskanner, dernæst er det nødvendigt at kunne tjekke, om det er muligt at køre hen til det givne punkt og sidst men ikke mindst en funktion der sorterer hullerne og finder den optimale rute for robotten. Disse funktioner mener jeg er muligt at konstruerer indenfor den tidsramme som der er afsat til bachelorprojektet, således at der haves en algoritme, som i mange tilfælde vil løse mobilitetsproblemet.

Algoritmen vil i princippet kunne undgå U-formede fælder, da et U ikke, forudsat at robotten kommer direkte hen imod U’et, vil give anledning til et hul eller spring. Dog vil robotten være fanget i U’et, hvis den kommer ind

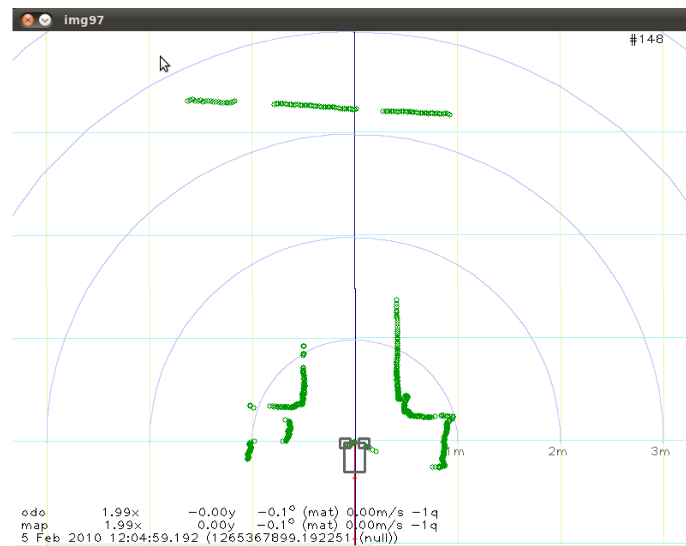
i det, da robotten ikke kan køre væk fra målet men kun hen imod. I denne situation vil Vektor Field Histogrammets kompleksitet kunne gøre en forskel. Problemet ved robottens opsætning er, at laserskanneren ikke kan se 360 grader rundt om robotten. Derfor er det svært at danne et fuldstændigt billede af omgivelserne. DTU's robotsoftware indeholder til gengæld en virtuel 360 graders laserskanner, som fungerer ved at huske punkter som robotten har kørt forbi. Jeg har ikke brugt denne funktionalitet, men den vil i givet fald kunne tilføje flere muligheder for algoritmen som ikke har noget hukommelse eller stifinder.

# Kap 3 – Teori

## 3.1 Bearbejdning af laserscannerdata

Laserskannerdata består af mellemrumseparerede double værdier. Først kommer et tidsstempel, dernæst et skan nummer, vinklen mellem hver af målingerne, den første vinkel og dernæst alle målingerne. Målingerne er repræsenteret på polær form med vinkel og afstand. Som standart er laserskanneren stillet til at levere 682 punkter startende fra 119.531 grader til -120.114 grader med -0,351562 graders interval. Vinkelen fremgår kun indirekte af laserskannet, og man bliver derfor nødt til at udregne vinklen til hvert punkt. Punkt nummer 30 svarer derfor til en vinkel på v = 119.531 – 30 * 0,351562 = 108,984 grader. Antallet af punkter nedsættes fx hvis vinklen reduceres til 180 grader.

Strukturen for laserskannerdata er allerede givet i DTU's robotsoftware. Strukturen giver mulighed for at bruge et hav af forskellige forprogrammerede funktioner og variable. Jeg bruger flere af funktionerne fra strukturen blandt andet getAngleRad, som returnerer vinklen i et givent punkt i skannet i radianer, getRangeMeter, som returnerer afstanden til et givent punkt i skannet og getRangeCnt, som returnerer antallet af punkter i skannet. Nedenfor ses et laserskan taget af en SMR.



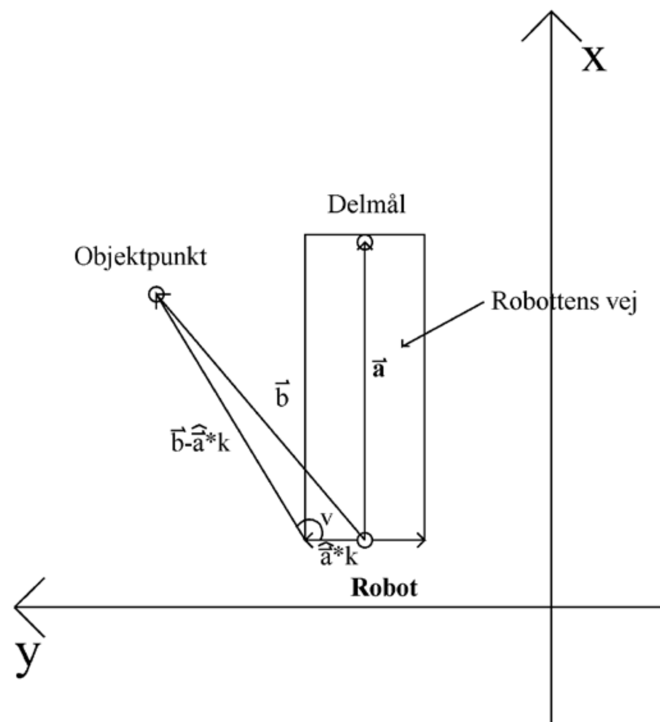

Figur 3 – Laserskan af robotten på vej igennem en døråbning.

Som det ses på figuren, er hvert punkt repræsenteret som en grøn ring. Objekter fremstår derfor som en kontur af grønne ringe rundt om objektets ene side (den side robotten kan se). Det fremgår også, at der i laserskannet er nogle fejl. I væggen lige foran robotten har laserskannet to "huller" (manglende måledata), som er væg i virkeligheden. Dette kan fx skyldes at væggen er sort på disse punkter, og derfor ikke reflekterer laserlyset lige så effektivt, som hvis væggen havde været hvid på disse punkter. Samtidigt observeres det, at robotten også kan se sig selv, da der er grønne ringe lige til højre for robotten selv. Når man arbejder med

laserskannerdata, er det derfor vigtigt at tage højde for fejl og mangler i selve skannet og gøre algoritmen robust over for fejlmåling. I algoritmen indgår der ikke hukommelse, men hvis man fx arbejder med ”Simultaneous Localization and Mapping” (SLAM), vil fejl på laserskanneren og odometien i sidste ende medfører, at det globale kort over omgivelserne bliver fyldt op med fejldata, så robotten ikke kan bevæge sig uden at støde ind i fiktive objekter.

## Objektundersøgelse

For at kunne afgøre om robotten kan kører hen til et givent delmål, er det nødvendigt at bruge informationerne fra laserskanneren. Hvis der befinder sig et objekt imellem robotten og målet, vil robotten kører ind i det givne objekt, og missionen vil fejle. Der er flere måder at løse problemet på, og jeg har efterfølgende erfaret, at der i DTU’s software allerede ligger en løsning, der dog er forskellig fra min egen. Her er min løsning: Ideen er, at bruge skalarprodukter til at undersøge om et punkt ligger indenfor det rektangel som robotten overstryger når den kører i lige linje fra start til slut.




**Figur 4 – Figuren viser hvordan der tjekkes for objekter mellem robotten og robottens mål. A er vektoren fra robottens position til robottens delmål. A-hat gange k, er vektoren fra robotten, som går vinkelret på a med den halve robotbredde som længde. Vektor b er vektoren fra robotten til punktet som undersøges. Figuren viser situationen i et globalt koordinatsystem.**

Hvis vinkel v er mindre end 90 grader er der mulighed for at objektet ligger i vejen for robotten. Dette undersøges ved at benytte skalarproduktet. Der gælder følgende tre udsagn om skalarproduktet s og vinkel v:

$$s = 0 \Leftrightarrow v = 90^\circ$$
$$s < 0 \Leftrightarrow v > 90^\circ$$
$$s > 0 \Leftrightarrow v < 90^\circ$$

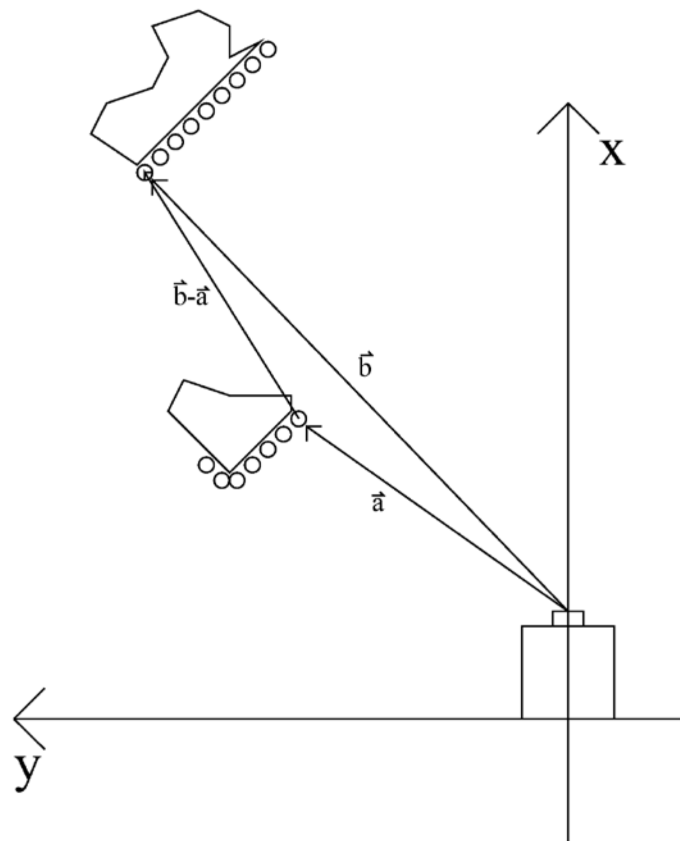
Hvis begge sider af robotten tages i betragtning: a-hat gange k og minus a-hat gange k, kan det kan det bestemmes, om objektet ligger på robottens vej. Nu defineres:

$$s1 = (\hat{\vec{a}} * k) \bullet (\vec{b} - \hat{\vec{a}} * k)$$
$$s2 = (-\hat{\vec{a}} * k) \bullet (\vec{b} + \hat{\vec{a}} * k)$$

Hvis s1 og s2 begge er mindre end nul, ligger objektet på robottens vej. Der er dog stadig den mulighed, at objektet ligger bag ved målet. Derfor er der et tredje nødvendigt tjek der skal foretages, nemlig om $|\vec{a}| + robotbredde > |\vec{b}|$. Hvis $(s1 < 0 \wedge s2 < 0) \wedge |\vec{a}| + robotbredde > |\vec{b}|$ ligger punktet i vejen for robotten. Hvis bare et af kriterierne ikke gælder, ligger punktet ikke i vejen. Disse tre tjek skal foretages for samtlige punkter i laserskannet. Bemærk, at punktet ikke kan ligge bag robotten, da der kun benyttes et 180 graders laserskan.

### Spring i laserskannet

For at kunne finde et spring i laserskannet, er det nødvendigt at have kendskab det foregående reelle punkt. Ved reelle punkt mener jeg et punkt som ikke skyldes fejldata. Alle fejldata og out of range data bliver af DTU's robotsoftware placeret indenfor 2 cm, men erfaringer med data viser, at indenfor 6,5 cm er der fejldata. Derfor ignorerer jeg alle målinger indenfor 7 cm for at have en buffer.




Figur 5 – Illustration af et laserskan, hvor der findes et spring. Illustrationen er vist i robottens lokale koordinatsystem.

Som det fremgår af illustrationen, er der et spring i laserskannet (a,b). For dette spring gælder der, at robotten potentielt set kan komme igennem. Springene findes ved at undersøge om afstanden mellem et givet punkt og det forrige er større end robotten radius. Hvis springet opfylder det, gemmes springet i et array til senere brug. Tjekket foretages på følgende måde:

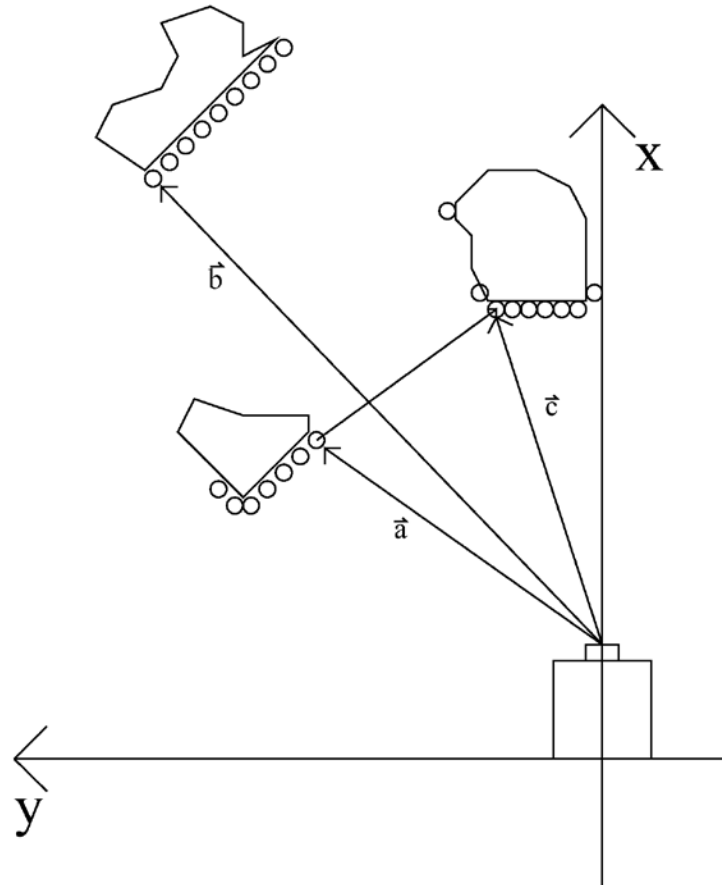
$$abs(|\vec{b}| - |\vec{a}|) > Radius$$

Her er vektor a vektoren til det foregående punkt og vektor b vektoren til det efterfølgende punkt.

Det burde ikke give problemer for algoritmen, at ”max range” opfattes som længder på under 2 cm, da disse længder opfattes som fejldata. Hvis der mellem 2 objekter er ”max range”, vil disse data blive sorteret fra, og afstanden mellem de to objekter vil blive udregnet som vist ovenfor.

## Hulgenkendelse

På figur 6 ses hvordan hulgenkendelsen fungerer. Springet i laserskannet er her fra a til b. Dette spring er gemt i spring-arrayet. Når hulgenkendelsen skal foretages, søges der igennem laserskannet for punkter som opfylder kravene for et hul. I dette tilfælde er punkt c det punkt, som vil give det mindste hul for springet a-b. Derfor indsættes hullet a-c i hul-arrayet.




**Figur 6 – Illustration af et laserskan. Der detekteres et hul (A,C) i laserskannet ud fra springet (A,B).**

Matematisk kan dette skrives som:

$$abs(|\vec{b}|-|\vec{a}|) > abs(|\vec{c}|-|\vec{a}|)$$

Hvis længden af linjestykket mellem c og a er mindre end linjestykket mellem b og a. Det er nu nødvendigt igen at tjekke om robotten kan passerer igennem hullet, da hullet kan være smallere end robotten. På denne måde findes med stor sandsynlighed den snævreste passage, som er i den pågældende retning. Hullerne i er modsætning til springene sværere at passerer igennem, da afstanden mellem punkterne er mindre. Ved at kende det snævreste sted, er det nemmere at finde den bedst mulige vej robotten kan køre.

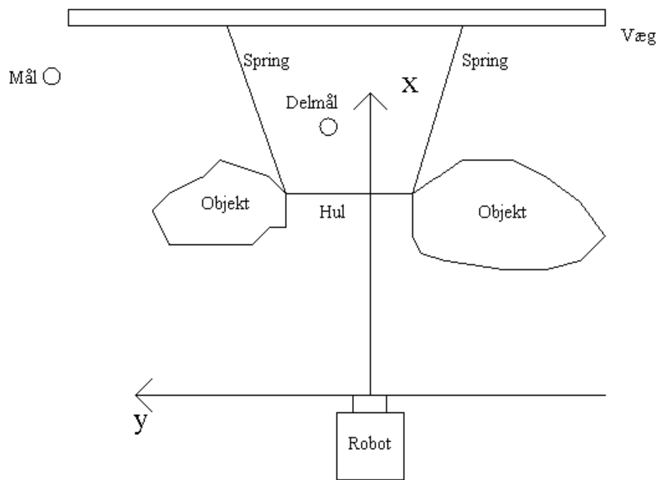
## 3.2 De 3 koordinatsystemer

I projektet arbejdes der i tre koordinatsystemer: Det globale koordinatsystem, som placerer robotten i origo med retning ud ad x-aksen når scriptet starter og ikke følger robotten. Det lokale koordinatsystem for robotten, som altid har origo imellem bagakslen. Endeligt er der laserskannerens koordinatsystem som er i virkeligheden er det samme som det lokale koordinatsystem bare forskudt 30 cm i det lokale koordinatsystems retning. Grunden til at jeg tager det sidste koordinatsystem med er, at det er muligt at forveksle (0,0) i laserskannet med (0,0) i det lokale koordinatsystem. Til omregning mellem det lokale og det globale bruger jeg en allerede eksisterende funktion i DTU's robotsoftware kaldet getToMapPose.

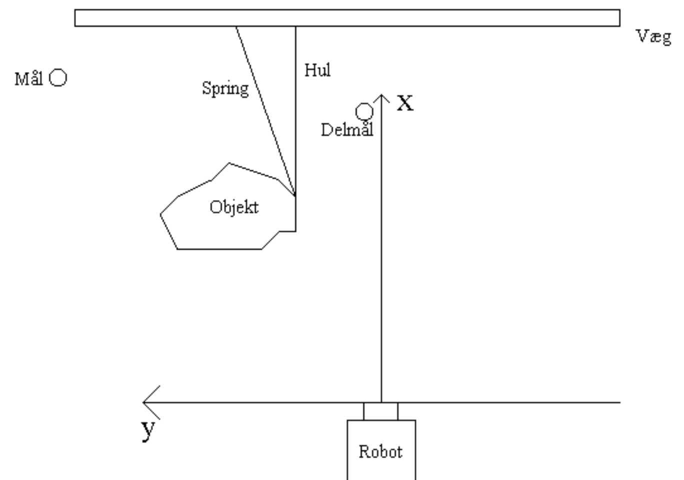
Jeg har i mit projekt valgt at arbejde i det lokale koordinatsystemer når jeg arbejder med laserskannet, og når jeg skulle planlægge robottens rute, så omregne alle huller og spring til globale koordinater. Robottens position er nemlig i globale koordinator. Ved at kende såvel delmålet som robotten i globale koordinator, kan kørslen fra robottens nuværende position til delmålet nemt findes. Det er meget vigtigt at holde de to koordinatsystemer adskilt, da der ellers kan opstå fejl, der er svære at finde. For eksempel starter robottens lokale og globale koordinatsystem med at være identiske. Ved at blande de to koordinatsystemer, har man en robot, som klarer den første, og muligvis den anden kommando uden bemærkelsesværdige fejl.

## 3.3 Kørselsstrategi

Indtil videre har strategien for algoritmen været, at hvis ikke kunne køre direkte til målet, skulle robotten køre hen til et hul. Dette er dog ikke helt optimalt, da hullet kan udgøre det mindste mellemrum mellem to objekter. Da robotten skal dreje hen imod målet efter hver gang algoritmen bliver kaldt, betyder det, at robotten skal dreje på det mindst mulige areal. Dette er ikke optimalt, og kan føre til, at robotten kolliderer med objektet. Derfor har jeg lavet en anden kørselsstrategi for robotten. Enten skal robotten kører direkte igennem hullet og forsætte en robotradius på den anden side af hullet i samme retning. Eller også skal robotten køre hen ud for midtpunktet af hullet en robotbredte fra hullet. Hvis forhindringen fx er en døråbning, hvor robotten står skævt for, vil algoritmen derfor få robotten til først at køre hen foran døråbningen, og derefter igennem. De to scenarier ses på figur 7a og 7b.




Figur 7a. Illustrastion af robotten der kører igennem et hul.



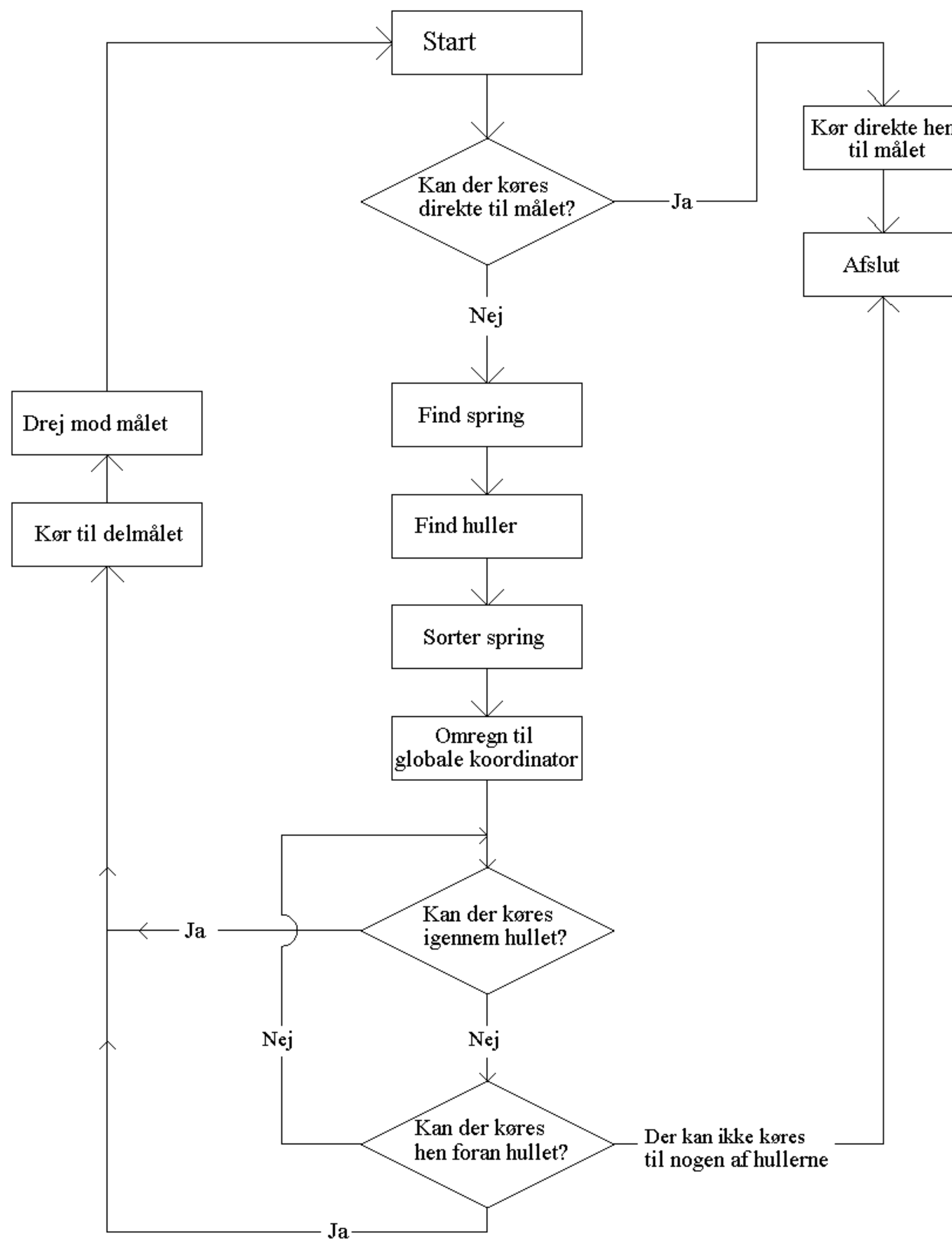

Figur 7b. Illustrastion af robotten der kører hen foran et hul.

Figur 7a viser hvordan robotten kører igennem et hul. Kørslen er fra robottens position gennem midtpunktet af hullet og forsætter i denne retning med en længde på robottens radius. Kørslen ender derfor *ikke* vinkelret ud for hullets midtpunkt. Dette er gennemprøvet, og gør algoritmen væsentlig dårligere. Figur 7b viser hvordan robotten kører hen foran hullet. Dette gøres fordi det ikke er muligt at køre igennem hullet. Delmålet ligger vinkelret på hullet med en afstand på robottens radius.

Jeg er af den overbevisning, at jo færre beslutninger der skal til for at få robotten til at kører fra A til B, jo bedre. Hvis robotten først skal kører hen foran hullet og bagefter igennem, er antallet af gentagelser af algoritmen to i stedet for et, hvilket giver større risiko for fejl og større risiko for at missionen fejler. Derfor tjekkes der altid først for, om robotten kan køre igennem hullet.

## 3.4 Algoritmen i detaljer

Det er nu muligt at kigge lidt nærmere på algoritmen. Algoritmen modtager et globalt mål. Algoritmen undersøger nu, om det på baggrund af lokale data fra laserskanneren, er muligt, at køre direkte til målet. Hvis dette er muligt, kører robotten til målet, ellers søges der i laserskannet for det bedste delmål for robotten. Der findes nu spring og huller i robottens lokale koordinatsystem, og hullerne gemmes i et hul-array. Disse huller sortes nu efter deres position i forhold til det globale mål, og alle huller, som ikke er brede nok for robotten slettes. Der omregnes nu til globale koordinater. Hullerne tjekkes i rækkefølge. Hvis robotten kan køre igennem et af hullerne, gøres dette. Ellers tjekkes der for, om robotten kan køre hen foran hullet. Hvis robotten kan køre hen foran hullet, gøres dette. Ellers går algoritmen videre til det næste hul i hul-arrayet. Hvis alle huller er løbet igennem, og der ikke er fundet noget muligt delmål, afsluttes algoritmen.




Figur 8 – Detaljeret metodediagram af algortimen.

Her ses et metodediagram over algoritmen. I praksis modtager pluginnet det globale mål hver gang algoritmen kaldes. Hver gang algoritmen når hen til enten ”Start” igen eller ”Afslut”, returneres der besked tilbage til skriptet om hvad robotten skal gøre. Hvis der er nået til afslut fra den ene eller den anden side, kaldes pluginnet ikke igen, mens det kaldes igen, hvis algoritmen lander på ”Start”.

# Kap 4 - Implementering og test

## 4.1 Testkørsel af algoritmen

Jeg har foretaget en række test af algoritmen, hvor der indgår kasser, spraydåser og andre objekter. Testene, jeg beskriver nedenfor, er der videodokumentation af på den medfølgende cd.

Jeg afprøvede robottens evne til at kører igennem døråbninger, og det viste sig, at den ikke havde nogen problemer med det. Heller ikke selvom robotten stod meget skævt for døren således at den ikke kunne køre direkte igennem den. Jeg har vedlagt to videoer på cd'en, som viser robotten der kører igennem en døråbning. På begge videoer er robotten blevet bedt om at kører 4 meter frem. På videoen "Døråbning 2" er robottens batteriniveau lavt, og den har derfor svært ved at kører hen over dørkarmen, trods dette når den frem til målet. Robotten står i dette scenarie meget skævt på døråbningen. Udover at opfylde sidste punkt i problemformuleringen, viser denne test, at algoritmen kan få robotten til at bevæge sig fra rum til rum. En anden ting som denne video fortalte var, at algoritmen ikke har noget problem med åbne områder. Som beskrevet tidligere, har algoritmen nogle lighedspunkter med Nearness Diagram. Nearness Diagram har den svaghed, at algoritmen får robotten til at slingre på åbne områder, dette gør min algoritmen ikke.

Dernæst afprøvede jeg algoritmens evne til at få robotten til at køre rundt om en kasse som det ses i videoen "Niveau 1". Dette gjorde robotten også uden problemer. Med disse to succesfulde test, har jeg mødt kravene fra problemformuleringen, men det betyder ikke, at algoritmen ikke kan klarer mere komplekse forhindringer. I videoerne "Niveau 2" og "Niveau 3" ses flere og flere kasser blive puttet ind, som øger kompleksiteten af forhindringen. På trods af dette finder robotten stadig vej til målet, som ligger 2 meter fremme. I "Niveau 3" indgår der 4 skraldespande, spraydåser, 5 papkasser kasser, en håndvask og en trækasse. På trods af dette, klarer robotten forhindringen. Dette fortæller mig, at algoritmen og implementeringen af algoritmen virker. Hvis ikke det virkede, ville robotten ikke klare den stigende kompleksitet.

I videoen "Zikzak" har jeg opstillet 4 papkasser og 2 træplader for at få robotten til at kører slalom mellem dem. Robotten klarede forhindringen hvis den var opstillet med passende afstand. Hvis forhindringerne kom for tæt på hinanden, havde robotten en tendens til at køre ind i forhindringerne, hvilket undrer mig. Det burde ikke være muligt for robotten at støde ind i ting, og derfor er der muligvis stadig en fejl i implementeringen af algoritmen, som kun nogle gange gør sig gældende.

I videoen "Kompleks forhindring" prøvede jeg at lave en U-formet fælde, det som ses ude til venstre i billedet, og med en kun meget snæver vej imellem to bordben. Algoritmen klarede forhindringen efter jeg havde gjort bordbenene synlige for robotten med spraydåser. At robotten klarede denne fik mig til at lave nogle test med U-formede fælder. Kun i tilfælde hvor robotten står inde i fælden med fronten mod bunden af U'et, dvs. At robotten ikke kan se nogen huller, fejler algoritmen.

I få af mine testkørsler, kørte robotten ind i kanterne på forhindringerne. Dette burde ikke kunne ske, og det tyder derfor på, at der stadig er en fejl i implementeringen. Fejlen skyldes ikke algoritmen, hvilket ville være

langt være. Jeg tror fejlen ligger i funktionen der tjekker, om robotten kan kører direkte til et delmål, men da robotten i de fleste tilfælde har klaret forhindringerne, har jeg ikke prioriteret fejlfindingen over testene.

I en af testene jeg foretog, havde jeg sat målet som robotten skulle hen til inde i objekt. Dette førte til, at robotten på skift kørte hen til de huller som lå tættest på målet. Algoritmen stoppede aldrig i dette tilfælde, da der altid var et hul at køre hen til. Dette problem er der ikke taget højde for i algoritmen. En løsning kunne være, at tjekke for om robotten havde været på det givne punkt tidligere. I stedet for denne lappeløsning, kunne man tage skridtet fuldt ud, og indføre en stifinder.

## 4.2 Diskussion af algoritmen

De første test af den færdige algoritme afslørede, at der var nogle grundlæggende fejl i implementeringen. Derfor blev jeg nødt til at gennemteste hele softwaren for fejl. Efterfølgende har algoritmen fungeret overraskende godt og langt bedre end forventet. Testene ovenfor er foretaget efter denne gennemtestning. Algoritmen klarer sig godt i komplekse scenarier med mange kasser, borde og døråbninger. De problemer robotten har, er primært hardware orienterede, da robotten generelt ikke kan se sorte objekter og objekter som ikke er i højde med laserskanneren. Dette har jeg taget højde for i mine testscenarier ved at spille ting foran bordben, således at robotten kan se dem og ved kun at have objekter med, som er i laserskannerens højde.

Algoritmen har en kvadratisk kørselstid på O(n*m), hvor n beskriver antallet af spring, der kan være i et laserskan, og m beskriver antallet af mulige huller, der kan være til hvert spring. N og m er sat til maks at kunne indeholde 100 og 7000, og derfor kører algoritmen meget hurtigt. Pluginnet når altid at sende svar tilbage i det tidsrum der er afsat af systemet til at svarer scriptet. Dette er ikke essentielt for, at robotten fungerer, men det gør, at der ikke er nogen ventetider mens robotten kører. Det er en god ting, at algoritmen er hurtig. For det første, er det irriterende at vente på et langsomt program, og for det andet, er det nødvendigt hvis algoritmen skal tilpasses til dynamiske miljøer. I øjeblikket er der en maks. grænse på kørselskommandoer på fire meter. Maks. grænsen er sat fordi robotten ikke kan se længere end fire meter. Hvis robotten skulle tilpasses dynamiske miljøer, kunne robotten sættes til kun at køre korte stræk ad gang og med kurvede sving i stedet for skarpe sving, hvor robotten skal stoppe hver gang. Dette ville medføre to ting: At algoritmen ville kunne tilpasses dynamiske miljøer, og at robotten ville kunne detekterer objekter bedre, da robotten primært har problemer med sorte objekter. Sorte objekter er primært et problem, hvis de enten er langt fra robotten, eller de har en spids vinkel i forhold til laserskanneren.

En af de ting, som algoritmen ikke kan klare, er hvis vejen hen til målet ligger i en anden retning end målet. Dette skyldes, at robotten altid vil kører til det hul som ligger nærmest målet, og den ikke kan se baglæns.

Robotten vender som bekendt altid hen imod målet efter den er kørt til et delmål. Dette er ikke et nemt problem at løse, og løsningen er heller ikke entydig. En simpel løsning, som ikke ville virke i alle tilfælde, er at indsætte en 360 graders laserskanner. Dette ville gøre algoritmen i stand til at få robotten til at køre i modsat retning af målet. Problemet er, at så snart robotten kan se et hul, som ligger tættere på målet end den vej, som det kræves for robotten at køre, vil robotten køre ind i det samme hul igen. Den eneste solide løsning til dette problem er en form for stifinder. Denne stifinder behøver ikke en stor forkromet løsning som SLAM for at kunne fungerer. Jeg har overvejet på at implementerer et array af knuder i en graf, som minder lidt om Visibility Graph bare med huller. Hver knude skal indeholde informationer om hvorvidt robotten har været der og punktets koordinater. Alle huller til et givent punkt skal indføres som knuder med kanter imellem hullerne og robottens position. På denne måde kan der "backtraces" tilbage til et punkt, hvis robotten er kørt ind i en blindgyde. Hullerne kan derefter undersøges et ad gangen. Der opbygges på denne måde et kort af robottens omgivelse som enkelte punkter med meget information i stedet for meget data med lidt information. Denne implementering af en stifinder, ville ikke være krævende for robotten at køre, og ville efter min bedste overbevisning gøre algoritmen komplet.

Jeg lader altid robotten rette op efter hver kørsel, så robottens vinkel i det globale koordinatsystem er nul. Dette har jeg valgt at gøre af hensyn til en eventuel bruger af systemet, så det er muligt at forsætte en mission efter kørslen af forhindringsundvigelsesalgoritmen.

Jeg mener, at algoritmen bidrager positivt til DTU's robotsoftware. Det at have en forhindringsundvigelsesalgoritme til rådighed, som kan passerer udefinerede objekter, giver en anden frihed til brugeren af SMR'erne. Hvis man kender det punkt man vil hen til på forhånd, men ikke kender vejen derhen, har man nu et værktøj, som muligvis kan få robotten der hen. Dette er specielt en fordel i dynamiske miljøer. Under en test kom der en person ind foran robotten, jeg fik bedt personen om at stå stille, hvilket personen gjorde, og robotten kørte udenom personen og hen til målet. Dette var ikke planlagt, men det beviste, at algoritmen kan klare sig i dynamiske miljøer. Hvis personen ikke havde stået stille, havde robotten måske kørt ind i personen. Da pluginnet kun køres imellem hvert sæt af kørselskommandoer, er algoritmen på nuværende tidspunkt kun til nogen grad beregnet til dynamiske miljøer. Objekter må gerne bevæge sig under kørslen af robotten, men dette skal ske i ryk, således at objekterne står stille mens robotten kører.

## 4.3 Brugervejledning

For at benytte avoidpluginnet til forhindringsundvigelse, skal DTU's robotsoftware først hentes og installeres fra DTU Elektros infosite [9]. Der er en guide til dette på siden. Dernæst skal laserskannerserveren startes på en af robotterne via en ssh forbindelse, og avoidpluginnet skal køres på denne server. Dernæst laves en anden ssh forbindelse til robotten, hvor avoidskriptet kan køres. Avoidskriptet kan modificeres som man ønsker. Maalx er x-koordinaten kan ønsker robotten hen til, og maaly er y-koordinaten. Hvis man ønsker at indsætte

kommandoer, som robotten skal udføre før algoritmen startes, indsættes dette et sted i skriptet senest før ”turn tal” linjen, og hvis man ønsker at udføre kommandoer efter, indsættes dette efter ”if($l4==0) "loop"” linjen. Både avoidskriptet og avoidpluginnet er vedlagt på cd’en.

## 4.4 TODO-liste

Lav en funktion der tjekker robottens opgivelser for, om det er muligt at dreje hen imod målet. Dette har ikke indtil videre voldt nogen problemer, men ville i sjældne tilfælde kunne gøre det.

Find fejlen til i pluginnet, som gør, at algoritmen nogle gange kører ind i kanten af objekter.

Lave en stifinder som kan gøre algoritmen komplet. Dette kunne fx være som beskrevet i afsnittet: Diskussion af algoritmen.

## Konklusion

Jeg har analyseret eksisterende algoritmer til forhindringsundvigelse, og på baggrund af dette syntetiseret en reaktiv algoritme. Algoritmen er blevet implementeret på DTU's robotter, således at den er enkel at bruge. Brugeren af systemet behøver ikke at kende til implementeringen af algoritmen, men kan bruge det vedlagte SMRCL-script til forhindringsundvigelse. Algoritmen får i mange tilfælde robotten korrekt frem til målet ved at køre udenom forhindringer. Dette betyder at både algoritmen og implementeringen af algoritmen virker.

Jeg fortæller detaljeret i teoriafsnittet hvordan laserskannerdata benyttes til at analyserer robottens omgivelser. Dette er gjort uden kodeeksempler således, at det er muligt at genskabe projektet på en anden platform. Jeg beskriver også teorien bag kørselsstrategien for algoritmen.

Algoritmen kører meget hurtigt, og der er ikke nogen betydelig kørselstid at tage højde for. Dette betyder, at det vil være muligt at tilpasse implementeringen af algoritmen til dynamiske miljøer.

Algoritmen kan både klare simple forhindringer med en kasse eller en døråbning, men også komplekse forhindringer med kasser, spraydåser, skraldespande osv.. Algoritmen kan derfor mere end det jeg havde målsat i problemformuleringen. Disse eksempler kan ses på det vedlagte videomateriale.

## Perspektivering

Jeg mener bestemt, at denne algoritme ville kunne bruges i virkeligheden, hvis der arbejdes videre på den. Det er en simpel algoritme, som er nem at implementerer, og som virker. Den er ikke følsom overfor U-formede objekter, og den kører hurtigt. Selvom algoritmen ligner Nearness Diagram, har algoritmen den fordel, at robotten ikke laver vrikkende bevægelser når den kører, og den godt kan lide åbne områder.

## Referenceliste

[1] Latombe, J.-C., Robot Motion Planning. Norwood, MA, Kluwer Academic Publishers, 1991.

[2] E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Matematik, 1 (1959), 269-271.

[3] Lumelsky, V., Stepanov, A., "Path-Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape".

[4] Andrews, J. R. and Hogan, N., "Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator." Control of Manufacturing Processes and Robotic Systems, Eds. Hardt, D. E. and Book, W., ASME, Boston, 1983, pp. 243-251.

[5] Minguez, J., Montano, L., “Nearness Diagram Navigation (ND): A New Real Time Collision Avoidance Approach,” in Proceedings of the IEEE/RSJ international Conference on intelligent Robots and Systems, Takamatsu, Japan, October 2000.

[6] Borenstein, J., Koren, Y., “The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots.” IEEE Journal of Robotics and Automation, 7, 278-288, 1991.

[7] Fox, D., “KLD-Sampling: Adaptive Particle Filters and Mobile Robot Localization.” Advances in Neural Information Processing Systems 14. MIT Press, 2001.

[8] Khatib, O., Quinlan, S., “”Elastic Bands: Connecting, Path Planning and control,” in Proceedings of IEEE International Conference on Roborics and Automation, Atlanta, GA, May 1993.

[9] http://timmy.elektro.dtu.dk/rse/wiki/index.php/Main_Page

# Appendiks A

```
%SMRCL script som bruger avoid plugin til at finde vej

maalx=2.0
maaly=0.0

tal=atan2(maaly,maalx)/3.14169*180
turn tal

label "loop"

stringcat "avoid x=" maalx " y=" maaly

laser "$string"
label "vent"
wait 0.1
if(tal == $l3) "vent"
tal=$l3

eval $l0; $l1; $l2; $l3; $l4; $l5; $l6; $l7; $l8

ignoreobstacles
turn $l0
ignoreobstacles
fwd $l1
ignoreobstacles
turn $l2

if($l4==0) "loop"
```