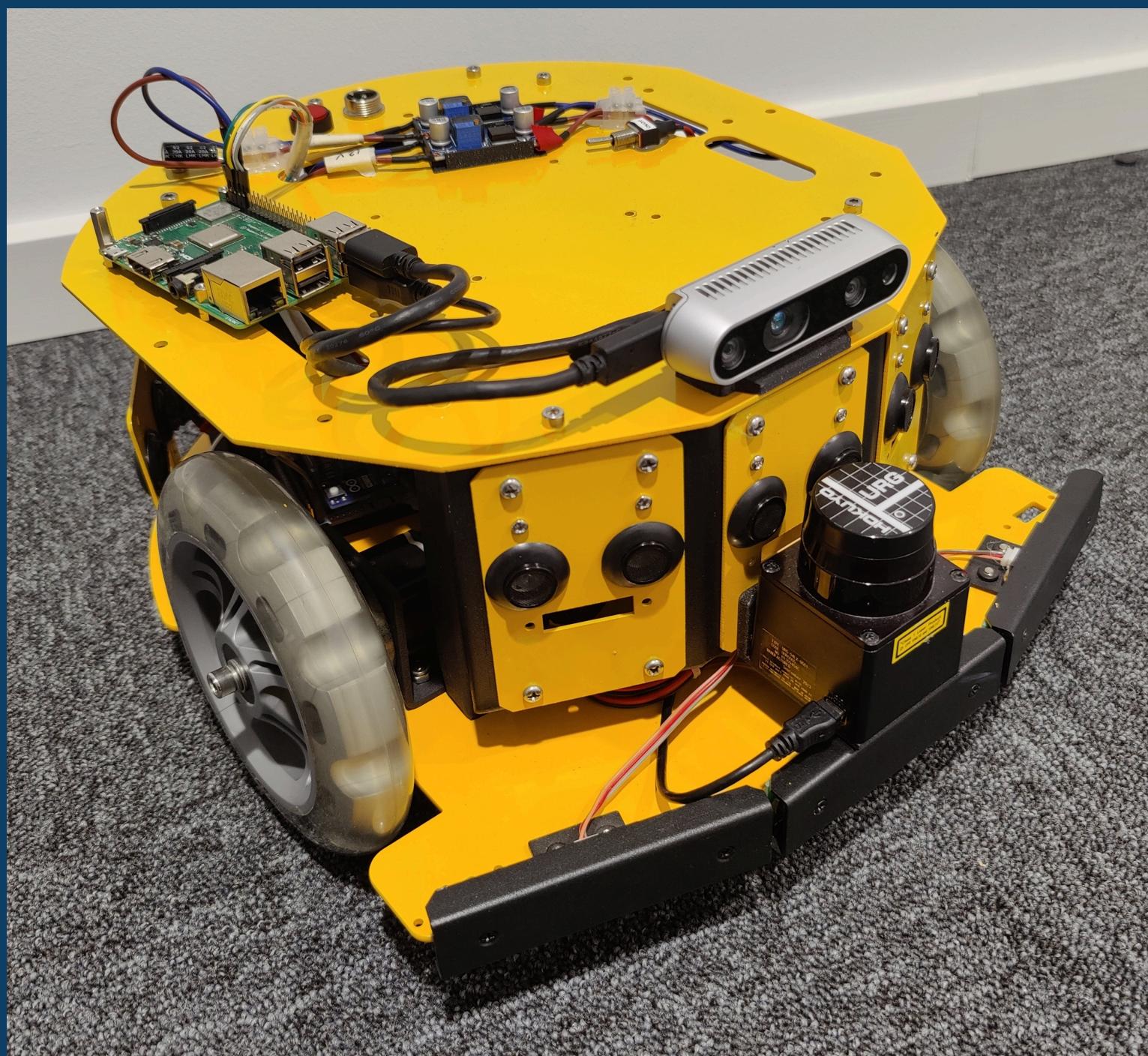


# **Autonom navigation og objektdetektering til en mobil robot**

## **Autonomous navigation and object detection for a mobile robot**

Afgangsprojekt 2021 - Efterår



## Godkendelse

Emil Siggi Asgeirsson - 468291

Kasper Gjødesen Mikkelsen - 138342858



Signature



Signature

03/01-2022

Date

03/01-2022

Date

## Forord

Denne afhandling er udarbejdet af Emil S. Asgeirsson & Kasper G. Mikkelsen, og lavet i samarbejde med Bionic System Solutions (BSS)[1]. Projektperioden er fra den 1. september 2021 til den 3. januar 2022.

Det antages, at læseren har en grundlæggende viden om neurale netværk, og kendskab til reguleringsteknik og sensorer. Derudover anvendes der engelsk decimal separator i stedet for dansk ved tal.

Total antal sider af 2400 tegn: 27 sider.

Antal tegn med mellemrum: 65458

Antal sider: 43

Projektet er udarbejdet i fællesskab, hvor vores individuelle ansvarsområder i projektet er følgende:

Emil      *Machine vision, neutralt netværk til objekt detektering & roadmap*

Kasper    *Reguleringsteknik, SLAM & lokalisering*

# Abstract

The project aims to develop a low-cost, low-power autonomous driving robot that can be implemented at facilities. Its purpose is to detect bluetooth sensors for communication purposes and be able to withdraw data from these sensors.

The robot shall be able to drive based on sensor data, mapping, avoiding obstacles, and object detection.

The focus was to detect objects using both machine vision for segmentation and neural network. Furthermore, sensors will be used for the regulation of the robot with PID regulators and a linear quadratic regulator, as well as mapping with SLAM to create routes and search for sensors. Multiple path planning algorithms assist in optimizing the driving routes and map coverage for searching.

Object detection using segmented images for the neural network, resulted in an accuracy of 98.36%. Combining a linear quadratic regulator with PID's resulted in robust control of the robot. SLAM successfully maps and corrects the robot's position. Obstacle avoidance using the camera does filter out most of the obstacles, but it requires the object to be entirely in the image for optimal results. Thereby adding lidar sensors can support the measurements from the camera. Due to limited computational power, the robot is not able to operate the combined areas.

# Indholdsfortegnelse

## Abstract

iii

---

## Introduktion

<b>Projektbeskrivelse</b>	<b>1</b>
<b>1 Introduktion</b>	<b>2</b>
1.1 Kravspecifikation . . . . .	3
1.2 Processer . . . . .	3
<b>2 Hardware</b>	<b>6</b>
2.1 Sensor præcision . . . . .	7

---

## Implementering

<b>3 Objektdetektering</b>	<b>8</b>
3.1 Data opsamling . . . . .	8
3.2 Billedsegmentering . . . . .	8
3.3 Transfer Learning . . . . .	10
3.4 Forberedelse af datasæt . . . . .	10
3.5 Model . . . . .	10
3.6 Resultater . . . . .	12
<b>4 Mekanisk model</b>	<b>14</b>
4.1 Konvertering af vektorer . . . . .	17
<b>5 Reguleringssteknik</b>	<b>19</b>
5.1 Motor regulatoren . . . . .	19
5.2 Position-/orienteringsregulator . . . . .	20
<b>6 Kortlægning</b>	<b>24</b>
6.1 Transformations matrix . . . . .	24
6.2 SLAM . . . . .	25
6.3 Partikelfilter . . . . .	31
<b>7 Rute planlægning</b>	<b>33</b>
7.1 Generaliseret Voronoi Diagram . . . . .	34
7.2 Cubic spline . . . . .	36
<b>8 Forhindringsdetektering</b>	<b>38</b>
8.1 Lidar . . . . .	38

8.2 Kamera . . . . .	39
----------------------	----

## Evaluering

---

<b>9 Konklusion</b>	<b>43</b>
---------------------	-----------

<b>Litteraturliste</b>	<b>44</b>
------------------------	-----------

## Bilag

---

<b>A Generaliseret Voronoi Diagram</b>	<b>I</b>
--	----------

<b>B Kode for segmentering</b>	<b>V</b>
--------------------------------	----------

<b>C Resultater fra Objektdekteringstests</b>	<b>XIV</b>
---	------------

# Projektbeskrivelse

I virksomheden, Bionic System Solution, opsamler man vibrationsdata ved brug af bluetooth low energy (BLE) sensorer, som er monteret synligt udenpå pumper, maskiner mv., generelt alt der har en vibration. BLE sensorene har en rækkevidde på 5 – 10 meter. En medarbejder går rundt til de enkelte BLE sensorer og indsamler data manuelt. Denne proces kan med fordel automatiseres med en mobil robot, som kører rundt og lokaliserer disse BLE sensorer. Det gøres ved brug af en vision sensor, på den mobile robot, til lokalisering. Herefter indsamles dataene fra BLE sensorerne, når de er indenfor rækkevidde.

I projektet bygges en mobil robot, som har til formål at kunne navigere rundt i et indendørs miljø. Systemet skal være i stand til at finde de synlige BLE sensorer, imens den undgår kollision.

Til at løse denne opgave, anvendes en embedded processor, som er monteret på bilen. Den vil blive hjernen for robotten, og håndtere kommunikationen til systemet. Den skal kontrollere bilens motorer, og håndtere reguleringsteknikken. Den vil modtage sensordata fra sensorer monteret på robotten, til at bestemme dens position og regulere det offset, der opstår under kørslen. Den embedded processor har også til formål at lokalisere BLE sensorer, ved brug af et vision input på den mobile robot. Machine vision anvendes, for at undgå at køre ind i forhindringer og detektere BLE sensorerne.

Vores problem formulering er følgende:

*Hvordan kan man udvikle en mobil robot, til autonomt at kunne navigere præcist i et indendørsmiljø, og lokalisere BLE sensorerne, ved hjælp af machine vision*

For at besvare denne problemformulering vil følgende delproblemer blive besvaret:

- Hvordan kan reguleringsteknik og sensorer anvendes til at få en mobil robot til at bevæge sig med præcision?
- Hvordan kan et kamera og flere sensorer anvendes til at kortlægge et rum?
- Hvordan kan machine vision detektere forhindringer under kørsel?
- Hvordan kan machine vision bruges til at lokalisere BLE sensorerne?

Projektet vil blive evalueret på baggrund af den mobile robots evne til at navigere og undgå kollision i et indendørs miljø og lokalisere BLE sensorer.

# 1 Introduktion

Dette projekt er den afsluttende opgave på Diplomingeniør i Robotteknologi uddannelsen. Denne opgave har til formål at dokumentere evnen til kritisk refleksion inden for et afgrænset, ingeniørfagligt emne, der er relevant for uddannelsen[2].

Industrien for robotter er i konstant udvikling. Dette betyder at anvendelsesområdet for robotter hele tiden bliver udvidet og dækker mere avancerede opgaver, samt afløser redundtant menneskelig arbejdskraft. Dette kan optimere kvaliteten af produkter, reducere omkostningerne og minimere risikoen ved farlige arbejdsmiljøer, o.lign. Dette gør sig også gældende i virksomheden **Bionic System Solutions**, hvor machine learning skal være med til at finde fejl i maskiner før eventuelle nedbrud. Derved vil de kunne få en reparatør til at vedligeholde systemet, når det bliver opfanget af machine learning. Dette vil kunne reducere arbejdet for reparatøren, som enten først bliver tilkaldt, når maskinen bryder ned, eller periodisk tjekker maskinen for unormale vibrationer. Til dette kan BLE sensorer, der gør brug af machine learning, validere maskinen på et bedre grundlag end reparatøren. Dette reducerer tiden imellem inspicering af maskinen, men der skal dog stadigvæk foregå manuel indsamling af data. Til det vil der i projektet blive udviklet en mobil robot, som autonomt kan indsamle dette data.

I dette projekt vil der blive lagt vægt på, at segmentere billeder og udvikle et neuralt netværk til objektdetektering af BLE sensorerne. Derudover vil en robust regulator til robotten blive implementeret, for at sikre en pålidelig kørsel til ønskede lokationer. Yderligere vil en let bearbejdning af lidar data, blive brugt til SLAM for at kortlægge og bestemme positionen for robotten. Sidst vil der blive kigget på forhindringsdetektering, for at undgå kollision ved kørsel på rute efter den er planlagt. Partikelfilter er kun til for at bestemme startpositionen i et kendt miljø.

BLE sensorene er blå, som ses på figur 3.1, og dermed kan en segmentering opsættes på baggrund af denne information. Med en segmentering af billederne, vil det neurale netværk have mindre information at bearbejde. En evaluering af det neurale netværk vil evalueres med og uden segmentering af billederne. Dette vil holdes op mod kravspecifikation 3 i sektion 1.1.

Reguleringsteknikken vil blive dokumenteret på baggrund af dens evne til, at få en mobil robot til at operere med præcision. Hertil vil en klassisk regulator, proportional-integral-derivative (PID), og en moderne regulator, linear-quadratic regulator (LQR), implementeres til at regulere robotten. Det vil blive evaluert på baggrund af præcisionen til at følge en reference, samt kravspecifikation 2 i sektion 1.1.

For at kortlægning og lokalisering kan bearbejdes på en **Raspberry Pi**, vil en light udgave af SLAM blive implementeret. Kravet for præcisionen er sænket lidt, grundet **Raspberry PI**'ens begrænset processor kraft. Resultatet for SLAM vil blive evaluert ud fra kravspecifikation 1 i sektion 1.1, samt dets evne til at skabe et kort ud fra behandlet lidar data. Efter

rummet er blevet kortlagt, så tilføjes et generaliseret voronoi diagram, som bidrager til at robotten bevæger sig i baner, som er længst væk fra kendte forhindringer.

I løbet af rapporten vil diskussioner og perspektiveringer blive gennemgået løbende i hvert emne.

## 1.1 Kravspecifikation

I forbindelse med vores projekt har vi følgende krav:

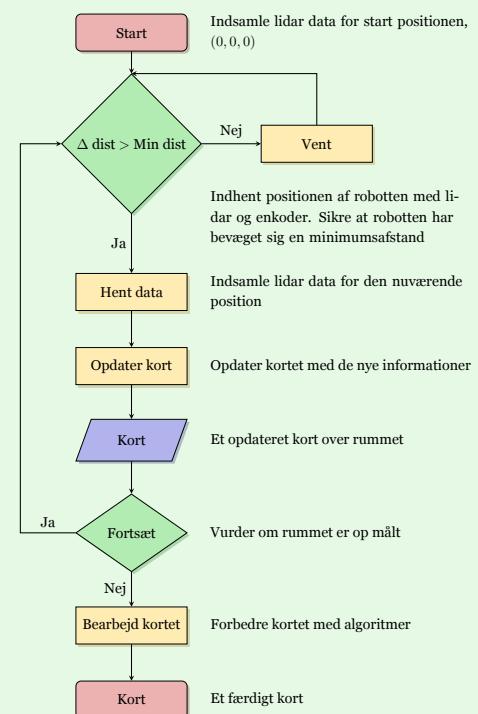
1. Kortlægge et rum på op til  $10 \text{ [m]} \times 10 \text{ [m]}$ .
2. Maksimale afvigelse af reguleringsteknikken på  $20 \text{ [cm]}$ .
3. Detektere BLE sensorer med en præcisionsscore over 80 %.
4. Detekteringen med kamera, skal kunne foregå i real tid.

## 1.2 Processer

For simplificering af projektet, er 3 processer illustreret med flowcharts i dette afsnit. Dette indebære at kunne kortlægge et vilkårligt rum, som efterfølgende kan blive brugt til rute planlægning. Derefter er der at objektdetektering i det kortlagte rum, og til sidst kørsel til objekterne. Dette kan ses på figur 1.1.

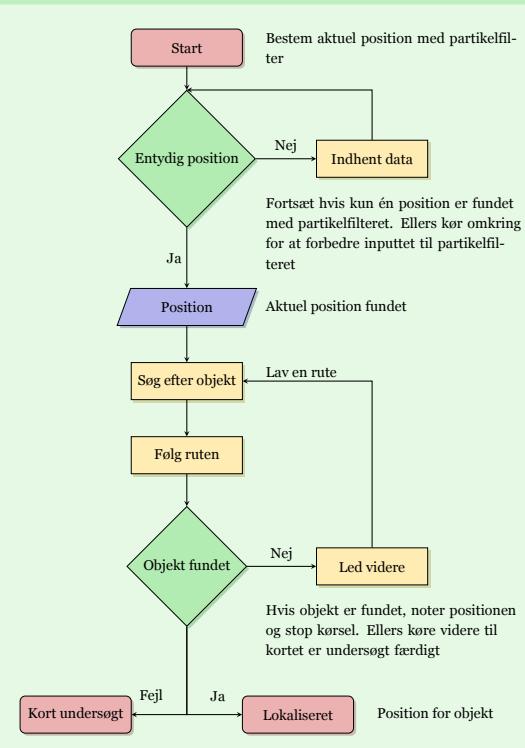
- Kortlægning, figur 1.1a.
- Objektdetektering, figur 1.1b.
- Ruteplanlægning, figur 1.1c.

## 1.1 Kortlægning



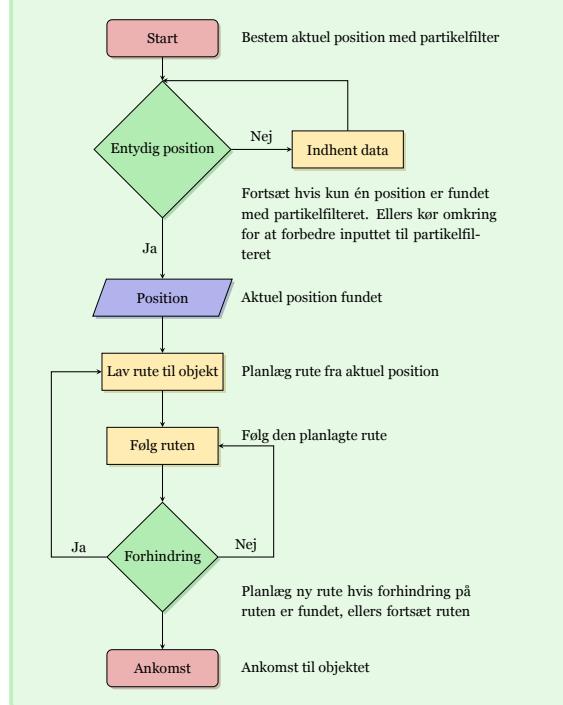
(a)

## 1.2 Objektdetektering



(b)

### 1.3 Ruteplanlægning



(c)

Figure 1.1: Flowcharts for processerne:

Flowchart figur 1.1a. Til kortlægning af et rum bruges lidar sensoren til afindsamle afstandsmålinger i rummet, samt enkoder til at estimere position. Dette korrigeres af lidarens input, når kortet er ved at blive skabt.

Flowchart figur 1.1b. Til objektdetektering anvendes et RGB-kamera, som giver et billede input, der segmenters og sendes til et neutralt netværk.

Flowchart figur 1.1c. Til ruteplanlægning til objekt, anvendes et kamera og lidar sensor til at detektere forhindringer på dens rute.

## 2 Hardware

Hardwaren, der bruges i dette projekt, er en tohjulet differential drevet robot[3], som kan ses på figur 2.1. Den har en motor med gearboks og enkoder på hver af de fremdrivende hjul, samt bumper sensorer i front til nødstop. Hjulene styres af en Arduino, som ved brug af enkoder data regulere motorerne med PID regulatorer. Oven på dette hardware lag er en Raspberry Pi, der styrer Arduino'en, og bearbejder data fra et 3D kamera samt en lidar sensor. De interne sensorer i robotten indebærer enkodere, mens de eksterne er kamera, lidar og bumpers.

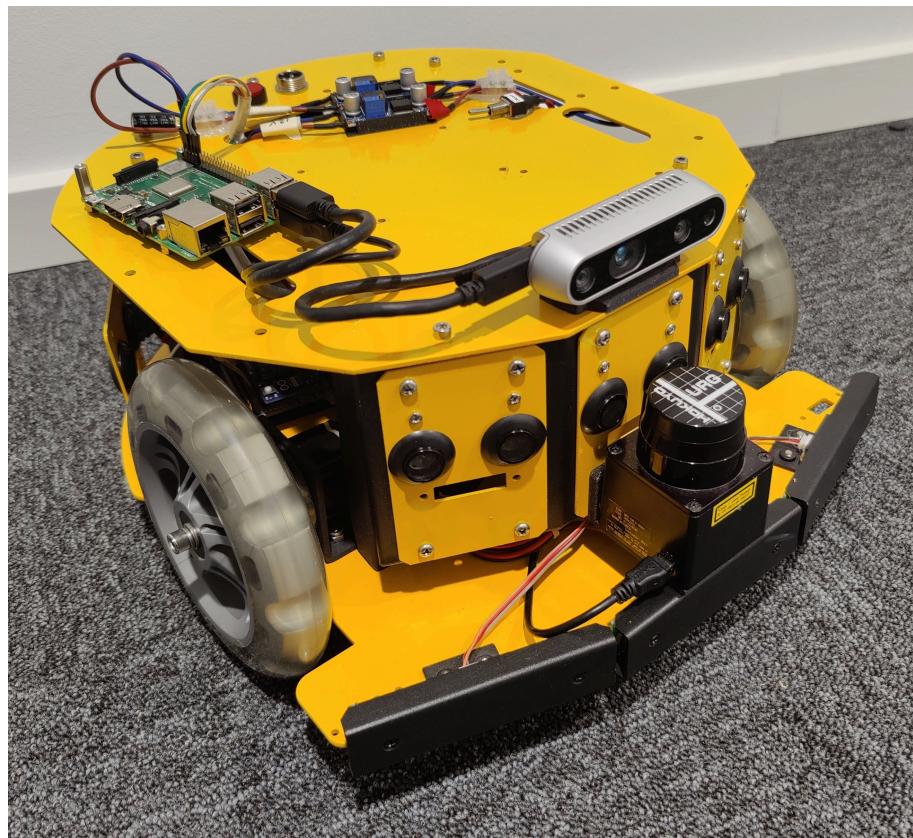


Figure 2.1: Den anvendte mobile robot

De interne sensorer dækker over bumper sensorer og enkoderne. I fronten af den mobile robot sidder tre taktile sensorer, hver med en rækkevidde på 3 mm. Disse sensorer blokere motorene ved aktivering, og sikrer at robotten stopper i tilfælde af en kollision. Derudover ses to enkodere, som mäter antallet af omdrejninger for hvert hjul. Dette data bruges til at finde robotens hastighed, samt estimere dens position. Enkoderne er optisk inkrementelle roterende med én skive. Dette medfører en usikkerhed i form af sensor aliasing, grundet den manglende evne til at detektere bevægelsesretningen på motoren.

## 2.1 Sensor præcision

Til at indsamle data fra omgivelserne bruges de eksterne sensorer. Den ene er en lidar af model URG-04LX fra Hokuyo Automatic Co.[4]. Denne lidar anvendes til at kortlægge samt bestemme af robottens position. Den kan måle op til 4.095 [m] med  $\pm 1\%$  nøjagtighed[4], og sensorvidden er på  $240^\circ$ , med en pitch vinkel på  $0.36^\circ$ . Den er designet til indendørs brug, hvor lux niveauet ikke er for højt.

Den sidste eksterne sensorer er 3D kameraet, der anvendes, er et Intel Realsense model D435[5]. Det anvendes til objektundgåelse og objektdetektering under kørsel. Kameraet mäter afstande fra 0.2 [m] til +10 [m], og har en sensorvidde på  $90^\circ$ [5].

Ud fra fabrikantens foreskrifter er det angivet, at dybdesensoren på Realsense har en præcision på  $< 2\%$  ved 2 [m][6]. Med disse som udgangspunkt, er der foretaget en præcisions test, for at konkludere om kameraet overholder de mål, som fabrikanten har opgivet. Der er foretaget to tests, og til begge tests er der sat et objekt op foran kameraet, hvor kameraet laver en afstandsmåling. Objektet er blevet sat ud fra et målebånd, som starter ved kameraet forkant, og dette målebånd anvendes som "ground truth" for målingerne.

Der er foretaget målinger ved 0.5 [m], 1.5 [m] og 2.0 [m]. Til hver afstand er der taget fire billeder med to objekter, hhv. en grå boks, og en papkasse. En afvigelse udregnes for hvert objekt, på baggrund af de fire resultater og opgives i procent.

Afstand	Metal boks	Papkasse	Gennemsnit
0.5 [m]	0.700%	0.100%	0.400%
1.0 [m]	2.375%	2.675%	2.525%
1.5 [m]	3.000%	4.016%	3.508%
2.0 [m]	4.900%	5.200%	5.050%

Ovenstående viser at kameraet på 2 [m] har en fejlmåling på 5.050 %. Det er ringere end hvad fabrikanten foreskriver. Dog skal det bemærkes, at der ikke er foretaget en kalibrering af kameraet. Kaliberingen er blevet fravalgt, da en afvigelse på  $\approx 5\%$  er tolerabel for projektet. Afstandsmålingen skal løse to opgaver; objektdetektering og objektundgåelse. Det er nødvendigt for robotten med højere præcision ved kortere afstande, for at undgå objekter. Til objektdetektering er præcisionskravet heller ikke højt, da den kun skal fortælle om vi er tæt nok på, til at forbinde til BLE sensorerne.

# 3 Objektdetektering

Til objektorientering benyttes først segmentering af billederne, hvor den blå farves isoleres i billedet. Dernæst bliver datasættet markeret med bokse omkring BLE sensorernes placering, som forberedelse til træningen af et konvolutionelt neuralt netværk. Den anvendte models fordele forklares i afsnit 3.5, og den testes på datasættet før og efter segmenteringen.

## 3.1 Data opsamling

Data opsamlingen er sket ved at køre robotten kører rundt i hele bygningen. Her tager den billeder med eget kamera fra eget perspektiv. Dette sikrer billedernes kvalitet, lysindfald og vinkler til omgivelserne. Det har giver os et datasæt på 241 billeder. Alle billeder kan ses i den vedlagte zip fil i mappen '/data/AlfaLaval'.

## 3.2 Billedsegmentering

Billedsegmentering sker på baggrund af farverne. BLE sensoren er blå og afskillem sig fra miljøet, som den sidder i. Se figur 3.1. Det gør det muligt at benytte en farvesegmentering til at filtrere BLE sensoren fra de omkringliggende ting.

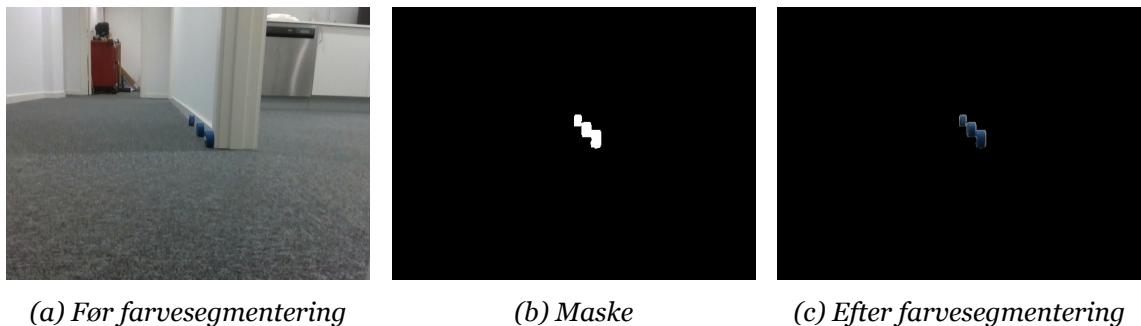


Figure 3.1: BLE Sensor

Source: [https://www.alfalaval.com/globalassets/images/products/fluid-handling/automation/sensing-and-control/cm/cm\\_top\\_640x360.jpg](https://www.alfalaval.com/globalassets/images/products/fluid-handling/automation/sensing-and-control/cm/cm_top_640x360.jpg)

På figur 3.2 ses et billede, som er blevet farvesegmenteret. Som det fremgår af billedet, har segmenteringen filteret alle andre nuancer effektivt fra, og kun de tre sensorer er tilbage på figur 3.2c. Farvesegmenteringen er blevet udviklet ved at benytte funktioner fra OpenCV i Python. Inputbilledet bliver først konverteret fra et RGB (Rød, Grøn og Blå) billede til HSV (Hue, Saturation, and Value), der er en anden type repræsentationsmodel. Her er farverne beskrevet på en anden måde. Eftersom den digitale version af et objekt, afhænger af mængden af lys som det rammes af, kan det være svært at diskriminere objekter fra hinanden ved brug af RGB. Her er det ofte mere brugbar information, som man får af HSV[7].

Til HSV billedet laves der en maske, som kun tillader bestemte HSV-værdier at komme igennem. Det resulterer i en maske, som kun indeholder pixels i det angivne HSV-farvespektrum. Nogle af disse pixelgrupper er små og ubetydelige, så for at filtrere yderligere i masken, har OpenCV en metode, der kan finde konturer i et billede. Den returnerer et array af arrays, hvor hvert array består af en sammensætning af pixels, der beskriver et område på billedet. På baggrund af en threshold fjernes konturer under en hvis størrelse for at fjerne støj på billedet. Thresholden er sat på baggrund af en minimumsstørrelse, hvorfra det er relevant at detektere en BLE sensor.



*Figure 3.2: Farvesegmentering*

Ved kontrol af billederne undersøges de for fejldetektering, i form af en confusion matrix, som ses i tabel 3.1. Der ikke er klassificeret nogen falsk negative BLE sensorer, hvilket giver segmenteringen en 100 % detekteringsrate. Udo over BLE sensorerne finder segmenteringen andre objekter, der er i samme farvespektrum. Sand negativ tages ikke med i optællingen, da det blot er alle områder, hvor algoritmen ikke har sat en maske, da der ikke er noget objekt.

Faktisk \Forudsagt	Positiv	Negativ
Positiv	291	0
Negativ	123	-

*Table 3.1: Confusion matrix*

$$\text{Præcisionsscore} = 100 - \left( \frac{123}{291} \cdot 100 \right) = 57.73\% \quad (3.1)$$

Segmenteringen har en rigtig evne til at finde alle BLE sensorerne, men den leverer en del falsk positiver. Dette giver en præcisions score på under 60 %, som er for lavt til krav 3 i sektion 1.1. Det forsøges at forbedre præcisionsscoren ved brug af transfer learning, der udover at segmentere farverne, kan lære at genkende mønstre ud fra den langt reduceret datamængde, der bliver sendt i modellen. Koden for segmentering kan ses i den vedlagte zip fil i mappen 'segmentering'.

### 3.3 Transfer Learning

For at mindske arbejdet i at lave et datasæt til et neuralt netværk, benyttes transfer learning frem for at lave et konvolutionelt neuralt netværk. Fordelen ved transfer learning er, at man benytter et eksisterende konvolutionelt neuralt netværk, hvor størstedelen af netværket 'fryses', så de features, som den er trænet i bibeholdes. Nogle af de øvre lag, som ikke er 'frosset' vil på baggrund af nyt data lære de nye features af kende for specifikke objekter. Til transfer learning er Edge Impulse[8] blevet benyttet til at konstruere netværket, som er en udviklingsplatform til embedded machine learning.

### 3.4 Forberedelse af datasæt

Manuelt skal der laves etiketter til hvert billede, for at beskrive billedets ground truth. Ground truth er et billede, hvor (i dette tilfælde) der er en firkant omkring det objekt, som man ønsker at finde. Når netværket trænes bruger den ground truth til at verificere de resultater, som netværket får løbende. Data, der beskriver ground truth, står i tekstdokumenter og beskriver, hvor boksen ligger.

Inden billederne bliver kørt igennem transfer learning, så ændres størrelsen på billedet til  $320 \times 320$ , hvor den længste akse bliver tilpasset for resten bliver fyldt med sorte pixels. På den måde sikres det at hele billedet er med, og der ikke er tab fra informationer, som ligger i periferien af billedet.

### 3.5 Model

Til det neurale netværk benyttes MobileNet V2, som er en udvikling af MobileNet v1. I et standard neurale netværk benyttes standard convolutions, som fungerer ved at man benytter en kernel over et billede, hvor disse værdier ganges på. Eksempelvis et RGB billede på  $8 \times 8 \times 3$ , hvor tretallet beskriver dybden af billedet, benyttes en kernel på  $3 \times 3 \times 3$ . MobileNet baserer sig på depthwise separable convolutions, som splitter en standard convolution til to lag: depthwise convolutions og pointwise convolutions. Depthwise convolution filterer kun billedet, og bagefter skaber pointwise convolution nye features. Ved Depthwise convolutions opdeles alle dybdelag til enkelte lag og en  $3 \times 3 \times 1$  kernel bruges på hvert lag frem for en enkelt iteration over billedet af  $3 \times 3 \times 3$ . Disse tre nye billeder stables oven på hinanden og kaldes for en depthwise convolution. Det næste lag lægger de nye billede sammen og en  $1 \times 1 \times 3$  pointwise convolution udføres over dem. At benytte depthwise separable convolutions med  $3 \times 3$  kernels reducerer beregningsomkostingerne med otte til ni gange i løbet af hele netværket, mod kun en meget lille reduktion i præcisionen[9].

Aktiveringsfunktionen anvendes for at reducere kompleksiteten imellem lagene og anvendes derfor efter hver convolution. MobileNets benytter ReLU[10], hvor output ved negative tal bliver sat til nul og positive tal er en lineært stigende funktion. Helt præcis anvendes ReLU6, som i den positive del flader ud ved 6. Det reducerer yderligere kompleksiteten af netværket.

Hvis depthwise og pointwise lag tælles med, så har MobileNet 28 lag, og efter hvert depthwise eller pointwise, så er der batch normalisering og ReLU. En ekstra parameter, som MobileNet har tilføjet er en width multiplier, som gør at man uniformt kan reducere et lag yderligere, hvilket reducerer beregningsomkostningerne, stadig med en rimelig præcision, reaktionstid og størrelsens trade off.

MobileNetV2 benytter inverted residual blocks og lineære bottlenecks. Inverted residual blocks er en afgrening af residual blocks. Residual blocks anvendes, fordi meget dybe netværk er svære at træne pga. forsvindende eller eksploderende gradienter. De tager inputtet og fører det fremad, hvor det adderes efter en standard eller depthwise separable convolution lige før den tilhørende aktiveringsfunktion. Se billede 3.3a. Det afhjælper den forsvindende / eksploderende gradient, så netværket ikke stagnerer, men fortsætter med at øge præcisionen, indtil at den flader ud ved en højere præcision[11]. Forskellen imellem residual blocks og inverted residual blocks ligger i, hvilken data, som bliver født fremad i netværket. Residual block tager fra en bred blok, hvor dataen bliver mindske til en smal blok og udvidet igen til en bred blok, mens inverted tager data fra en smal blok, udvider den til en bred blok og mindske den igen til en smal blok, så den passer igen med næste convolution, som ses på figur 3.3.

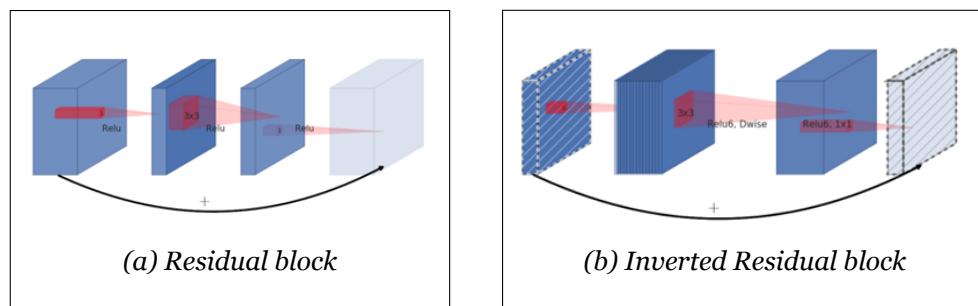


Figure 3.3: Residual blocks

Source: <https://arxiv.org/pdf/1801.04381.pdf>

Linear bottleneck betyder at det sidste lag af en residual blok fjernes aktiveringsfunktionen, som giver et lineært output i stedet for ReLU's ulineære output. Det giver et mindre tab af data på bekostning af ydeevne, og derfor kaldes det for bottlenecks. Sammen med inverted residual blokke, så gør det databehandlingen mere hukommelseseffektiv[12]. Se figur 3.4, hvor billede 3.4a viser MobileNet V1's lagstruktur og 3.4b viser MobileNet V2's lagstruktur. Forskellen ses på lag, hvor der kun er et enkelt stride, så anvendes inverted residual blocks og lineære bottlenecks, mens hvis der en stride på mere end en, så anvendes kun den lineære bottleneck.

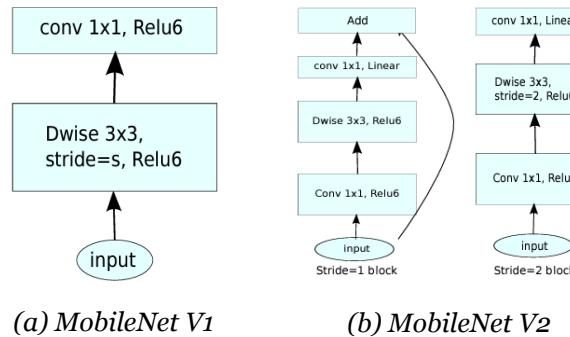


Figure 3.4: MobileNet layer structure

Source: <https://arxiv.org/pdf/1801.04381.pdf>

### 3.6 Resultater

I dette afsnit sammenlignes resultaterne af transfer learning for datasættet med og uden segmentering. På figur 3.5 ses det bedste resultat af transfer learningen. Resten af tests og resultater kan ses i bilag C. Til denne test er der benyttet 20 epoker og en læringskurve på 0.015. Output efter hver epoke med loss score og validation loss score er illustreret sammen. Loss scoren fortæller, hvor sandsynligt det er, at modellen forudsiger det rigtige resultat. Rød er træningsloss med segmentering, blå er valideringsloss med segmentering, orange er træningsloss uden segmentering og grøn er valideringsloss uden segmentering. Ved datasættet uden segmentering ses det, at valideringsloss stagnerer cirka imellem 0.4 og 0.6 i loss score, mens træningsloss falder ned til omkring 0.2. Det betyder, at datasættet her er overfittet i træningsdataen. I den segmenterede del følger de to kurver hinanden. Validation loss når ikke helt ned under 0.2, som trænings loss, men den er ikke langt fra, hvilket betyder at modellen bliver trænet godt uden overfitting.

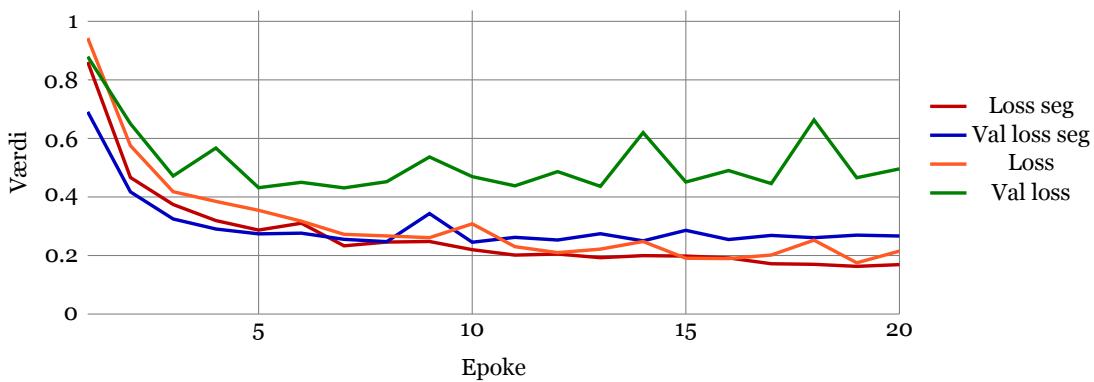


Figure 3.5: Edgeimpulse

Herunder ses præcisionsscorerne fra de to netværk. De to scorer er resultater af Cocos metric score og viser henholdsvis resultater fra valideringssættet og testsættet. Resultaterne uden segmentering viser at datasættet er blevet meget overfittet, da præcisionen er meget lav for testsættet. Omvendt er præcisionen for testsættet højere end valider-

ingssættet, og derfor er modellen rigtigt god til at detektere objekterne. Thresholden er sat på 0.25. Det er en lav threshold, men modellen finder et meget lavt antal af falsk positive, hvilket giver den en høj præcisionsscore.

	Præcisionsscore - validering	Præcisionsscore - test
Segmenteret	74.3	70.49
Usegmenteret	95.8	98.36

*Table 3.2: Resultater for Transfer Learning*

Det neurale netværk har sine fordele og ulemper. Når det bliver benyttet udelukkende på datasættet, så opnår det ikke en særlig høj detekteringsgrad. Det samme sker for segmenteringen af datasættet. Forskellen imellem de to er, at segmenteringen sørger for at finde alle BLE sensorer, mens det neurale netværk overser nogen af dem. Når de segmenterede billeder benyttes af det neurale netværk, så opnår det en høj præcision, hvor den finder 98.36 [%] af BLE sensorerne, hvilket er over det krav, som er sat i kravspecifikation 3 i sektion 1.1. Det neurale netværk forbedrer resultaterne fra segmenteringen ved at sortere den sidste støj ud af billedet, som segmenteringen ikke fanger.

## 4 Mekanisk model

Til at beskrive robotten i forhold til reguleringsteknikken vil forholdet mellem enkoder og rotation blive udledet, dele af robottens state space model, og til sidst en konvertering af kræfter fra lineær og rotationelle til robottens to parallelle hjul.

Den anvendte mobile robot i dette projekt er en differential drevet mobil robot. En visualisering af dennes opbygning kan ses på figur 4.1. Den styres af en Arduino som genererer et pulsbreddemodulation (PWM) signal til hver motor, hvilket skaber kræfterne  $\vec{v}_l$  og  $\vec{v}_r$ . Den har der uover et fritløbende hjul i bag til at holde robotten stabil, en gearkasse med 1 : 64 forhold på hvert hjul[3], samt en 12 counts per revolution[3] (CPR) enkoder i enden af hver gearkasse. Udledning af enkoderens præcision kan ses i udledning 4.1.

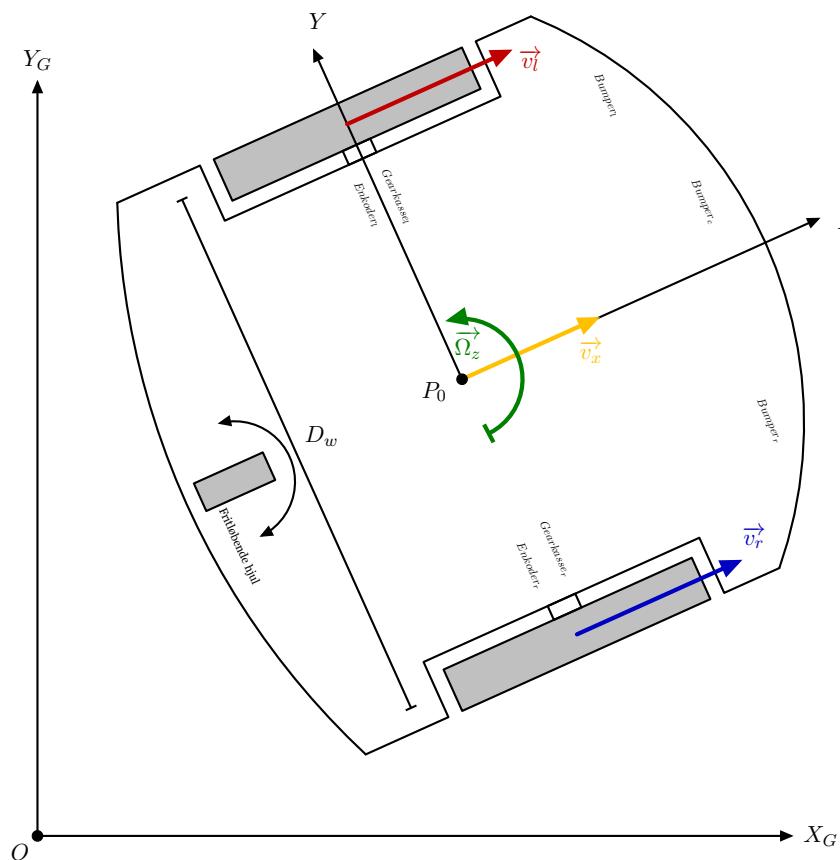


Figure 4.1: Kinematisk model af differential drevet robot

#### 4.1 Udledning af enkoderens præcision

$$\text{Gearboks ratio} = 1 : 64$$

$$\text{Enkoder spor} = 12 \text{ CPR}$$

$$\text{Hjul diameter} = 0.145 \text{ [m]}$$

Enkoderen bruges både på rising og falling edges, og der kan hentes 24 counts ud pr rotation. Dette giver følgende counts for hvert hjuls rotation med gearkassen:

$$\begin{aligned} 1 \text{ hjul rotation} &= 64 \cdot 24 \text{ CPR} \\ &= 1536 \text{ CPR} \end{aligned}$$

Omkredsen af hjulet er:

$$\text{Omkreds} \simeq 0.456 \text{ [m]}$$

Et tick svarer derved til:

$$1536 \text{ ticks} \simeq 0.456 \text{ [m]} \quad (4.1)$$

$$1 \text{ tick} \simeq \frac{0.456 \text{ [m]}}{1536} \quad (4.2)$$

$$1 \text{ tick} \simeq 0.000296875 \text{ [m]} \quad (4.3)$$

$$1 \text{ tick} \simeq 0.297 \text{ [mm]} \quad (4.4)$$

I forbindelse med måling af enkodernes præcision er der målt et mekanisk slack på  $\pm 1.7^\circ$ . Dermed kan hjulet rotere ganske kort, før enkoderen følger med, og dette skaber en usikkerhed for præcision af enkoderne. Dette slack vises ved opstart, hvor der dermed kan være et offset fra start position, og under rotationer hvor motoren slår til og fra. Dette kan i edge cases påvirke målingen med:

$$\frac{1.7^\circ}{360^\circ} \cdot 0.145 \text{ [m]} \cdot \pi = 0.00215 \text{ [m]} = 2.15 \text{ [mm]} \quad (4.5)$$

Ved antagelse af blot det ene hjul er i en edge case, vil det påvirke orientering som følgende:

$$\cos(\theta_z) = \frac{\text{hosliggende}}{\text{hypotenusen}} \quad (4.6)$$

De to kateter er hhv. bredden på robotten og afstanden det ene hjul har forskudt sig fremad. Robottens bredde er på 0.29 [m]:

$$\cos(\theta_z) = \frac{0.29 \text{ [m]}}{\sqrt{0.00215^2 \text{ [m]} + 0.29^2 \text{ [m]}}} \quad (4.7)$$

$$\theta_z = \arccos\left(\frac{0.29 \text{ [m]}}{\sqrt{0.00215^2 \text{ [m]} + 0.29^2 \text{ [m]}}}\right) \quad (4.8)$$

$$\theta_z = 0.42477^\circ \quad (4.9)$$

Til at opsætte LQR bruges en statespace model over systemet. Udledning af denne state space model kan ses i definition 4.2[13].

## 4.2 Udledning af state space

I forbindelse med state space modellen er der kun behov for at udlede  $x_t$ ,  $u_{t-1}$ ,  $A_{t-1}$ , og  $B_{t-1}$ . Dette skyldes at LQR ikke har behov for mere information til at udføre reguleringen.

Fra figur 4.1, kan ligningerne for  $x_t$ ,  $y_t$  og  $\theta_t$  beskrives, hvor at  $dt$  er et tidstep.

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + \cos(\theta_{t-1}) \cdot dt \cdot v_{t-1} \\ y_{t-1} + \sin(\theta_{t-1}) \cdot dt \cdot v_{t-1} \\ \theta_{t-1} + dt \cdot \omega_{t-1} \end{bmatrix} \quad (4.10)$$

Her er  $x_t$  systemets states, som indebærer  $x$ ,  $y$  og  $\theta$ :

$$x_t = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} \quad (4.11)$$

Det samme udledes for  $x_{t-1}$ , som er udledt af  $x_t$  til tiden  $t - 1$ :

$$x_{t-1} = \begin{bmatrix} x_{1t-1} \\ x_{2t-1} \\ x_{3t-1} \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} \quad (4.12)$$

Formlen for  $x_t$  i state space som diskretiseret, er som følgende:

$$x_t = A_{t-1} \cdot x_{t-1} + B_{t-1} \cdot u_{t-1} \quad (4.13)$$

Robotten har to hjul, som kan rotere og translatere i forhold til en akse. Dermed kan følgende udledes for input matrixen  $u_{t-1}$ :

$$u_{t-1} = \begin{bmatrix} u_{1t-1} \\ u_{2t-1} \end{bmatrix} = \begin{bmatrix} v_{t-1} \\ \omega_{t-1} \end{bmatrix} \quad (4.14)$$

System matrixen  $A_{t-1}$  kan udledes ved brug af  $x_t$  og  $x_{t-1}$ :

$$A_{t-1} = \begin{bmatrix} \frac{\partial x_1}{\partial x_{1t-1}} & \frac{\partial x_1}{\partial x_{2t-1}} & \frac{\partial x_1}{\partial x_{3t-1}} \\ \frac{\partial x_2}{\partial x_{1t-1}} & \frac{\partial x_2}{\partial x_{2t-1}} & \frac{\partial x_2}{\partial x_{3t-1}} \\ \frac{\partial x_3}{\partial x_{1t-1}} & \frac{\partial x_3}{\partial x_{2t-1}} & \frac{\partial x_3}{\partial x_{3t-1}} \end{bmatrix} = \begin{bmatrix} \frac{\partial x_t}{\partial x_{t-1}} & \frac{\partial x_t}{\partial y_{t-1}} & \frac{\partial x_t}{\partial \theta_{t-1}} \\ \frac{\partial y_t}{\partial x_{t-1}} & \frac{\partial y_t}{\partial y_{t-1}} & \frac{\partial y_t}{\partial \theta_{t-1}} \\ \frac{\partial \theta_t}{\partial x_{t-1}} & \frac{\partial \theta_t}{\partial y_{t-1}} & \frac{\partial \theta_t}{\partial \theta_{t-1}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.15)$$

Udledning af input matrixen  $B_{t-1}$ , udledes efter samme metode som udledningen af  $A_{t-1}$ . Her anvendes  $x_t$  og  $u_{t-1}$ , og giver følgende:

$$B_{t-1} = \begin{bmatrix} \frac{\partial x_1}{\partial u_{1t-1}} & \frac{\partial x_1}{\partial u_{2t-1}} \\ \frac{\partial x_2}{\partial u_{1t-1}} & \frac{\partial x_2}{\partial u_{2t-1}} \\ \frac{\partial x_3}{\partial u_{1t-1}} & \frac{\partial x_3}{\partial u_{2t-1}} \end{bmatrix} = \begin{bmatrix} \frac{\partial x_t}{\partial v_{t-1}} & \frac{\partial x_t}{\partial \omega_{t-1}} \\ \frac{\partial y_t}{\partial v_{t-1}} & \frac{\partial y_t}{\partial \omega_{t-1}} \\ \frac{\partial \theta_t}{\partial v_{t-1}} & \frac{\partial \theta_t}{\partial \omega_{t-1}} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{t-1}) \cdot dt & 0 \\ \sin(\theta_{t-1}) \cdot dt & 0 \\ 0 & dt \end{bmatrix} \quad (4.16)$$

Disse udledninger kan sammes og indsættes i ligning 4.13. Det giver følgende state space for robotten:

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \cos(\theta_{t-1}) \cdot dt & 0 \\ \sin(\theta_{t-1}) \cdot dt & 0 \\ 0 & dt \end{bmatrix} \cdot \begin{bmatrix} v_{t-1} \\ \omega_{t-1} \end{bmatrix} \quad (4.17)$$

Udledning af matricerne  $y$ ,  $C$  og  $D$  ses udeladt, eftersom de ikke skal bruges til regulering med LQR.

## 4.1 Konvertering af vektorer

Fra LQR, sektion 5.2 gives en lineær hastighedsvektor,  $\vec{v}_x$ , og en rotationel hastighedsvektor,  $\vec{\Omega}_z$ , for at opnå den ønskede position og orientering af robotten. Disse vektorer skal konverteres til en lineær vektor for hvert hjul, for at kunne kontrollere robotens bevægelser ved brug af PID regulatorer, sektion 5.1. Konvertering af disse vektorer kan ses i udledning 4.3.

### 4.3 Udledning af $\vec{v}_l$ og $\vec{v}_r$ ud fra $\vec{v}_x$ og $\vec{\Omega}_z$

$\vec{v}_l$  og  $\vec{v}_r$  er hastigheden for hvert hjul.  $\vec{v}_x$  er den lineære hastighedsvektor og  $\vec{\Omega}_z$  er den rotationelle hastighedsvektor fra LQR, sektion 5.2.

Den rotationelle hastighed kan udledes fra figur 4.1, hvor  $D_w$  er bredden fra hjul til hjul og  $r$  er radius på hjulet. Dette giver følgende ligning:

$$\vec{\Omega}_z = \frac{-\vec{v}_l + \vec{v}_r}{D_w \cdot r} \quad (4.18)$$

$$\vec{\Omega}_z \cdot D_w \cdot r = -\vec{v}_l + \vec{v}_r \quad (4.19)$$

Hvor at  $\vec{v}_l$  og  $\vec{v}_r$  er defineret som følgende.  $\vec{\omega}_y$  betegner rotationshastigheden af hjulene.

$$\vec{v}_l = \vec{v}_x - \frac{\vec{\omega}_y \cdot r}{2} \quad \vec{v}_r = \vec{v}_x + \frac{\vec{\omega}_y \cdot r}{2} \quad (4.20)$$

$\vec{v}_l$  og  $\vec{v}_r$  kan substitueres med ligning 4.20, og giver følgende udtryk, som kan reduceres.

$$\vec{\Omega}_z \cdot D_w \cdot r = -\left(\vec{v}_x - \frac{\vec{\omega}_y \cdot r}{2}\right) + \left(\vec{v}_x + \frac{\vec{\omega}_y \cdot r}{2}\right) \quad (4.21)$$

$$\vec{\Omega}_z \cdot D_w \cdot r = -\vec{v}_x + \frac{\vec{\omega}_y \cdot r}{2} + \vec{v}_x + \frac{\vec{\omega}_y \cdot r}{2} \quad (4.22)$$

$$\vec{\Omega}_z \cdot D_w \cdot r = \frac{\vec{\omega}_y \cdot r}{2} + \frac{\vec{\omega}_y \cdot r}{2} \quad (4.23)$$

$$\vec{\Omega}_z \cdot D_w \cdot r = 2 \cdot \frac{\vec{\omega}_y \cdot r}{2} \quad (4.24)$$

$$\vec{\Omega}_z \cdot D_w \cdot r = \vec{\omega}_y \cdot r \quad (4.25)$$

$$\vec{\Omega}_z \cdot D_w = \vec{\omega}_y \quad (4.26)$$

Fra ligning 4.20 og ligning 4.26 kan følgende udledes:

$$\vec{v}_l = \vec{v}_x - \frac{\vec{\Omega}_z \cdot D_w}{2} \quad \vec{v}_r = \vec{v}_x + \frac{\vec{\Omega}_z \cdot D_w}{2} \quad (4.27)$$

$\vec{v}_l$  og  $\vec{v}_r$  kan simplificeres til:

$$\vec{v}_l = \vec{v}_x - \vec{\Omega}_z \cdot \frac{D_w}{2} \quad \vec{v}_r = \vec{v}_x + \vec{\Omega}_z \cdot \frac{D_w}{2} \quad (4.28)$$

# 5 Reguleringssteknik

Reguleringssteknikken for den mobile robot består af en position-/orienteringsregulator, og motor regulator. Position-/orienteringsregulator bruger en linear-quadratic regulator (LQR), som leverer en lineær-/rotationel hastighed. Dette konverteres til  $\vec{v}_l$  og  $\vec{v}_r$  for robotens to fremdrivende hjul, som reguleres med to PID regulatorer.

Koden for reguleringssteknikken er vedlagt i bilag zip filen i mappen '/reguleringssteknik'.

## 5.1 Motor regulatoren

Robottens to hjul, som set på figur 4.1, styres af en Arduino. Den håndterer konverteringen fra en lineær-/rotationel hastighed til hastighedsvektorer for hvert hjul med formlen for  $\vec{v}_l$  og  $\vec{v}_r$ , fra udledning 4.3. Disse hastighedsvektorer bruges som input til PID regulatorerne for at opnå en ønskede ydeevne.

De anvendte PID regulator er parallelt koblet, og dette tillader at regulere uafhængigt af den proportionelle konstant,  $K_p$ . Formlen og de anvendte parametre for PID regulatorerne kan ses i definition 5.1.

Til implementering af denne regulator, på den anvendte Arduino, er et bibliotek blevet anvendt[14]. Dette bibliotek er blevet let optimeret til robottens anvendelsesområde. Arduino'en regulerer spændingen til hjulene ved et PWM (pulsbredde modulation) signal på 31250 Hz. Resultaterne af motor regulatoren kan ses på figur 5.1. Årsagen til at PWM signalet bruger en høj frekvens, er for at udjævne spændingen til motorene, og sikre en stabil spænding under load.

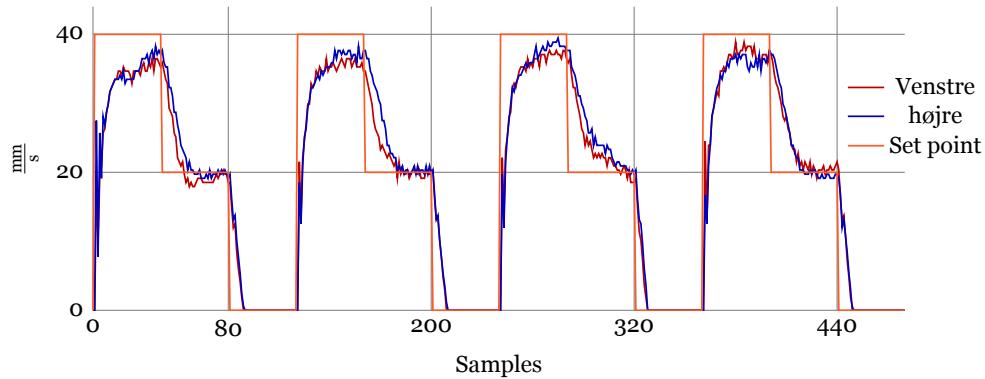


Figure 5.1: Målinger på PID

## 5.1 PID regulatoren

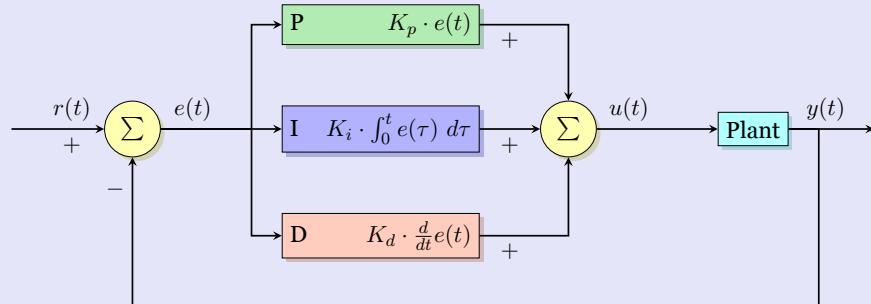


Figure 5.2: Opbygning af den anvendte parallel PID.

Den anvendte formel for PID regulatoren:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau) d\tau + K_d \cdot \frac{d}{dt} e(t) \quad (5.1)$$

Hvor at  $e$ :

$$e = \text{hastighed} - \text{input} \quad (5.2)$$

For PID regulatoren er følgende parametre anvendt:

Venstre	Højre
$K_p = 0.5$	$K_p = 0.55$
$K_i = 1.5$	$K_i = 1.695$
$K_d = 0.01$	$K_d = 0.011$

## 5.2 Position-/orienteringsregulator

Positions-/orienteringsregulering vil blive håndteret af en optimal regulator anvendes i form af en lineær quadratic regulator (LQR). Den skal kunne regulere et dynamisk system med minimale omkostninger. Hertil bruges lineære differentialligninger, som beskriver systemdynamikken, der er udledt i definition 4.2, samt omkostningerne for reguleringen, som er en kvadratisk funktion[13]. Definition af disse omkostninger bruges to kost matricer, som vægter input omkostningen,  $R$ , og tolerancen for afvigelse,  $Q$ . Dette bruges til LQR for at finde den optimale regulering i forhold til kost matricerne.

LQR har flere forskellige fordele. LQR udregner en statisk gain matrix,  $K$ . Dermed er ordnen også den samme som for systemet[15]. Derudover er LQR robust og nem at regulere på baggrund af de to kost matricer. LQR har også fordeloen at det ikke er tilstandene,  $x$ , som skal minimeres, men i stedet outputvariablerne,  $y$ . Dette er muligt grundet vægtningen af  $Q$ , og  $R$ .

Implementeringen af LQR er lavet i python til Raspberry Pi'en. Til at implementere denne regulator er der taget udgangspunkt i en online artikel om en simulering af LQR i python[13]. Dette har givet et fornuftigt udgangspunkt af implementeringen. Koden er efterfølgende blevet ændret fra at være en simulering til, at håndtere den fysiske robot. Derudover er meget omskrevet for at passe til robotten. Ved simuleringen blev en afstandsgrense fra robotten til den ønskede position anvendt, som blev brugt til at vurdere om kørslen til punktet var fuldført. Efter at have fuldført kørslen til et punkt, vil det næste punkt på ruten blive det nye mål, og således ville den fortsætte til robotten havde kørt til det endelige mål. Dette viste at være et større problem for den fysiske robot, som kunne risikere slet ikke at komme indenfor grænsen til punktet på grund af ekstern støj. Derfor blev implementeringen ændret til at den enten skulle være inden for en grænse, som her kunne være lavere end tidligere, eller at robotten er tættere på det nye punkt end det forrige. Dette gjorde at robotten kunne køre mere jævnt, som følge af at lidt afvigelse var tilladt. Yderligere blev kræfterne fra LQR algoritmen faktoriseret til at robotten skulle bevæge sig med en hastighed på op til 3 km/t. Problemet med den tidligere implementering var at der ved store afstande blev sat en højere hastighed for at komme tæt på målet, men i takt med at robotten nærmede sig målet blev hastigheden drastisk nedsat. Dette medførte derfor også meget acceleration og deacceleration på ligestrækningerne. Ved at faktorisere robotten til at hjulene kunne køre med en top fart på 3 km/t, medførte dette en jævn kørsel på lige strækninger. Yderligere ved at trække den rotationelle kraft fra den lineære kraft betød dette, at robotten også kunne håndtere sving mere jævnt. Dette blev ikke kun implementeret for at robotten køre mere jævnt, men i ligeså høj grad fordi robotten havde svært ved at accelerere, uden at trække lidt skævt.

Til at udregne LQR,  $u_t^{LQR}$ , bruges matricerne for *performance*,  $Q$ , og *effort*,  $R$ . Disse to matricer er positiv semidefinitive matricer.

$$Q = \begin{bmatrix} q_1 & & 0 \\ & \ddots & \\ 0 & & q_n \end{bmatrix} \quad (5.3)$$

$$R = \begin{bmatrix} p_1 & & 0 \\ & \ddots & \\ 0 & & p_n \end{bmatrix} \quad (5.4)$$

$$\mathcal{J} = \int_0^{\infty} (x^T Q x + u^T R u) dt \quad (5.5)$$

Til at styre robotten med et bestemt interval skal en diskretiseret jacobian anvendes, som udledes i ligning 5.5. Udledning af LQR med diskretiseret algebraisk Riccati-ligning kan ses i definition 5.2.

## 5.2 Udledning af LQR med discretiseret algebraisk Riccati-ligning

Discrete-time algebraic Riccati equation[16], hvor  $P \in \mathbb{R}^{n \times n}$ .

$$P_{t-1} = Q + A^T P_t A - A^T P_t B (B^T P_t B + R)^{-1} B^T P_t A \quad (5.6)$$

Den optimale gain matrice,  $K$ , udregnes ved brug af  $P_{t-1}$ .

$$K = (R + B^T P_{t-1} B)^{-1} B^T P_t A \quad (5.7)$$

For  $t = 0, \dots, N-1$ , hvor  $N$  er antallet af rækker i  $u, x \in \mathbb{R}^n$  &  $u \in \mathbb{R}^p$ , fås det LQR optimerede input:

$$u_t^{LQR} = K_t x_t \quad (5.8)$$

For simuleringen er følgende matricer for *performance*,  $Q$ , og *effort*,  $R$ , anvendt:

$$Q = \begin{bmatrix} 0.3 & 0.0 & 0.0 \\ 0.0 & 0.3 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad (5.9)$$

$$R = \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & 0.2 \end{bmatrix} \quad (5.10)$$

Eftersom at simuleringen kun afspejler et ideelt system, kræver det lidt optimering af parametrene for at få den optimale regulator på det fysiske system. Det skyldes også at det fysiske system bruger PID regulatorer til at opnå de ønskede hastigheder for hvert hjul, hvor imod simuleringen af LQR regulatoren antager at robotten har bevæget sig den ønskede afstand, og har derfor ikke nogen forsinkelse fra PID'erne eller støj.

Derfor er matricen fra  $Q$  ændret til:

$$Q = \begin{bmatrix} 0.5 & 0.0 & 0.0 \\ 0.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad (5.11)$$

Test banen for LQR regulatoren er lavet til at teste robottens evne til at køre i skarpe sving, buer,  $90^\circ$  sving og lige strækninger, samt at krydse sin egen bane. Resultaterne af LQR regulatorens evne til at følge en reference rute kan ses på figur 5.3, hvor figur 5.3a er en simulation af systemet, og figur 5.3b er testet på robotten med PID regulatorene til at regulere hjulene.

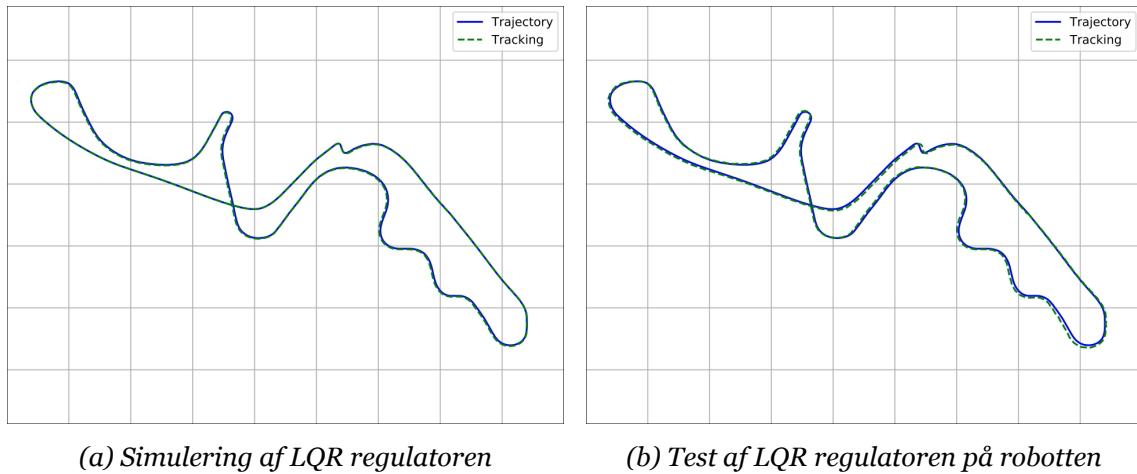


Figure 5.3: Test af LQR regulatoren. Grid størrelsen er 0.5 [m]

Det kan ses på figur 5.3 at simuleringen følger ruten med en højere præcision end ved testen på robotten. PID regulatorerne introducerer støj, hvis de ikke kører helt perfekt ift. LQR, og derfor afviger den fra simuleringen. Resultaterne viser at robotten er i stand til at følge referenceruten, med en tilstrækkelig nøjagtighed for projektet.

Med PID regulatoren kunne det ses at den kan følge en reference på figur 5.1, og dette er fintunet til at samarbejde med LQR. Hvis PID regulatoren justeres mere aggressivt, resulterede dette i komplikationer for kørslen. Dette skyldtes at der er en forskel i belastningen på de 2 hjul, og at der derfor er forskellige karakteristikker for hver motor under acceleration og deceleration. Dette er årsagen til at overshooting er blevet fravalgt ved tuningen.

For LQR fremgår det tydeligt på figur 5.3a, at regulatoren er robust. Det vurderes at et mindre trade off for præcisionen er mulig, for at tillade en mere flydende kørsel. Dermed skal robotten blot være indenfor  $\pm 5$  [cm] af ruten, før næste punkt på ruten vælges.

Sammensætning af disse regulatorer kan ses på figur 5.3b. Sammensætningen medfører at præcisionen nedsættes fra  $\pm 5$  [cm] til  $\pm 10$  [cm] offset til reference ruten. Dette offset er stadig inden for kravspecifikation 2 i sektion 1.1, som er på 20 [cm]. Dette krav er sat efter at robotten skal kunne navigere sikkert gennem døråbninger og snævre passager.

# 6 Kortlægning

Kortlægning har to formål. At skabe et kort over området, som robotten skal operere inden for og at kunne bestemme sin egen position i kortet, ud fra sensor data. Hertil er en SLAM algoritme blevet implementeret, for at kunne håndtere begge formål.

Koden for kortlægning er vedlagt i bilag zip filen i mappen '/kortlægning'.

Under udførelsen af kortlægningen antages det at forhindringerne midlertidigt er fjernet. Dette gøres for at simplificere kortlægningen, og undlade at håndtere at nogle lidar målinger kan tilhøre forhindringer.

## 6.1 Transformations matrix

Fra tidligere sektion 2, figur 2.1, kan det ses at lidaren er monteret foran robotten for bedre at kunne detektere forhindringer. Dette betyder at målingerne, som opfanges af lidaran, skal transformeres ind til centrum af robotten ved at gøre beskrive målingerne i forhold til robotten.

Lidaren er placeret 15 [cm] fra centrum af robotten, og derved skal alle målingerne forskydes med 15 [cm], for at kunne beskrives i forhold til centrum. Denne translations matrix, som multipliceres på alle målingerne, kan ses i ligning 6.1.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0.15 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (6.1)$$

Efter korrigering af disse målinger, vil de justeres i forhold til robottens position,  $p$ . Første led er at korrigere punkterne efter robottens position,  $(p_x, p_y)$ , som gøres med en translations matrix. Efterfølgende tilføjes orientering af robotten,  $p_\theta$ , ved brug af en rotations matrix, som multipliceres på alle målingerne som ses i ligning 6.2.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos p_\theta & -\sin p_\theta & 0 \\ \sin p_\theta & \cos p_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (6.2)$$

Disse translations matricer og rotation matrix, kan samles til en transformations matrix, som er udledt i ligning 6.3.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0.15 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos p_\theta & -\sin p_\theta & 0 \\ \sin p_\theta & \cos p_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (6.3)$$

Reducering af translations matricerne:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & p_x + 0.15 \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos p_\theta & -\sin p_\theta & 0 \\ \sin p_\theta & \cos p_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (6.4)$$

Sammenlægning med rotations matrixen:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos p_\theta & -\sin p_\theta & p_x + 0.15 \\ \sin p_\theta & \cos p_\theta & p_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (6.5)$$

## 6.2 SLAM

Den anvendte algoritme, som er blevet implementeret, er lidar SLAM[17], som baserer sig på afstandsmålinger indsamlet fra lidar sensoren. Derefter anvendes *iterative closest point* (ICP), som giver en relation mellem det eksisterende data og de nye målinger. Dette er CPU tungt grundet ICP algoritmen anvender singular value decomposition (SVD). For at reducere dette load, som pålægges den anvendte Raspberry Pi, er enkoder data tilføjet. Dette giver et udgangspunkt for forskydningen af de nye data målinger, således at ICP algoritmen kan spare adskillige af iterationer.

Til at forbinde det indsamlede data fra lidaren anvendes ICP, som giver en transformations matrix,  $H$ . Denne matrix kan multipliceres på målingerne for at opnå den bedste sammenlignighed mellem to datasæt.

Der er forskellige ICP algoritmer alt efter det tilgængelige data. Den anvendte i dette projekt er en *Point to point* ICP algoritme med ukendt data sammenhæng. Denne version finder de nærmeste punkter mellem to datasæt, hvor det forsøges at minimere afstanden mellem datasættene, ved at estimere den optimale relative rotation og translation. Denne proces udføres så længe at fejlen mellem datasættene reduceres, og resultere i en optimal relativ rotation og translation mellem de to datasæt. Dette kan ses som psedukode i algoritme 1.

Som tidligere nævnt eksisterer der forskellige ICP algoritmer, hvor de mest populære er *point to point* og *point to plane*[18]. Forskellen mellem disse er hastigheden, præcisionen, robustheden. *Point to point* algoritmen er hurtig, fordi den sammenligner punkter, og kræver dermed ikke nogen bearbejdning. Derimod skal *point to plane*, første konvertere det eksisterende datasæt om til linjer, som efterfølgende sammenlignes med de nye punkter. Dette er hovedsageligt årsagen til valget af *point to point* algoritmen. Sammenligning direkte mellem punkter i to datasæt har også den ulempe at i tilfælde af støj kan det medføre at denne algoritme ikke konvergere, samt at det er svært at fra vælge afgivende punkter. Det gør den mindre stabil end *point to plane* algoritmen, som sammenligner punkter til linjer i stedet for punkterne som danner disse linjer[19].

---

**Algorithm 1** Point to point ICP algoritme

---

**Input:**  $\text{list}(\text{prev\_data})$ ,  $\text{list}(\text{new\_data})$

**Output:**  $H_{3,3}$  ▷ Homogeneous matrix

**Ensure:**  $\text{prev\_data} \neq \text{null}$

**Ensure:**  $\text{new\_data} \neq \text{null}$

```

1: procedure ICP
2:    $\text{error} = \infty$ 
3:    $H = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ 
4:    $\text{weight} = \text{weight\_function}$ 
5:   while ( $\text{error decreased}$ ) and ( $\text{error} < \text{threshold}$ ) do
6:      $\text{offsets}, \text{error} = \text{nearest\_neighbor}(\text{prev\_data}, \text{new\_data})$ 
7:      $\text{offsets} = \text{offsets} \cdot \text{weight}$  ▷ Weight the list of distances
8:      $W = \sum_{(i, j) \in \mathbb{R}} (q'_i - \mu_Q) \cdot (p'_j - \mu_P)^T$  ▷ Compute the cross-covariance matrix
9:      $U, D, V^T = \text{SVD}(W)$  ▷ Use the SVD to decompose
10:     $R = U \cdot V^T$  ▷ Calculate the rotation matrix
11:     $T = \mu_Q - R \cdot \mu_P$  ▷ Calculate the translation vector
12:     $H = \text{update}(H, R, T)$ 
13:     $\text{new\_data} = H \cdot \text{new\_data}$  ▷ Update the new_data
14:   end while
15:   return  $H$  ▷ Return the homogeneous matrix
16: end procedure

```

---

Ved den anvendte ICP algoritme er det besluttet at bruge et sæt punkter, for at reducere kompleksiteten. Derudover tillader det også for feature baseret data, hvor punkter som repræsenterer hjørner og lige strækninger kan udtages. Dette gør det muligt at lave algoritmen på baggrund af et simplificeret datasæt, som indeholder informationer om vigtige features, og fjerne afvigende punkter som vil introducere støj. Således kan en let bearbejdning af datasættet øge præcisionen for resultatet samtidigt med at opnå en hurtigere eksekvering på Raspberry Pi'en. Ud over muligheden for feature baseret data kan datasættene som anvendes også vægtes. Det kan justere hastigheden for konvergeringen, som enten kan justeres mod forbedret performance eller stabilitet. Betydningen for valget af features og vægtningen har indflydelse på både eksekveringstiden og præcisionen. Påvirkningen af vægtning og features kan ses på figur 6.1.

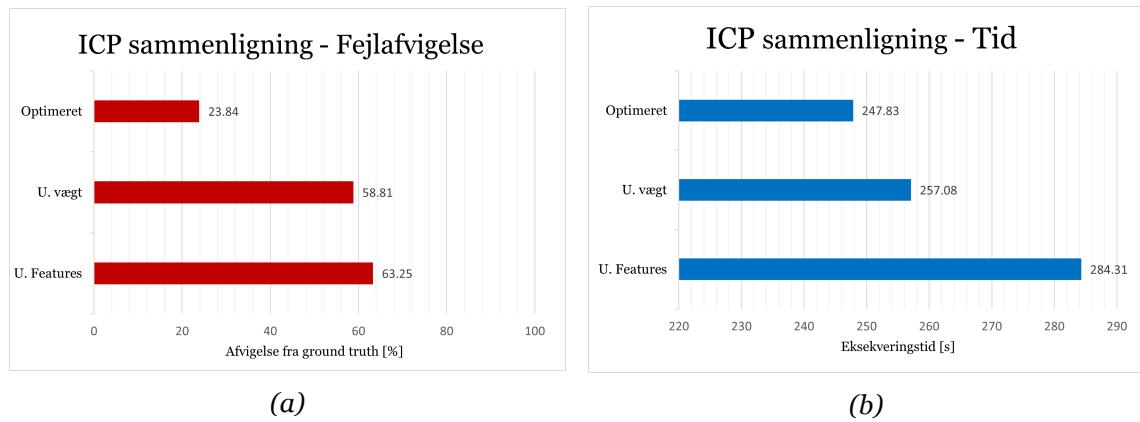


Figure 6.1: Sammenligning for ICP algoritmen ved brug af 1834 datasæt:

På figur 6.1a ses en procent afvigelse fra ground truth billede. Denne målinger viser hvor stor procent del af resultaterne, som ligger inden for en tolerance af 1 [cm]. Her kan det ses vægtnings og valg af features har en stor betydning for præcisionen.

På figur 6.1b ses det tydeligt at features og vægtnings forbedrer eksekveringstiden. ICP algoritmen kan med de rette input konvergere hurtigere og reducere kompleksiteten, som resultere i de hurtigere tider.

ICP kan, ud over at finde sammenhængen mellem to data sæt, også bruges til kortlægning og lokalisering. Ved at multiplicere den homogene matricen fra ICP algoritmen på de nye målinger, kan målingerne korrigeres til at hænge sammen med de forrige. Derved kan dette skabe et kort og korrigere robottens position. Udfordringen ved dette er, hvis der sker en fejl i forbindelsen mellem datasæt, som vil få robotten til at tro den er et andet sted end hvor den er. For at reducere den risiko antages det at enkoderne giver en præcis forskel mellem datasættene. Dermed indikerer enkoderne, hvor meget offset der er være mellem de to datasæt.

Til at vælge punkterne til ICP algoritmen er der taget udgangspunkt i en artikel baseret om 'line segment extraction algorithm', som anvender data fra en laser[20]. Deres løsning er udvidelse af split and merge algoritmen, som er blevet effektiviseret til sorteret laser data. Den performance forbedring der opnås, og at de kan finde vinkler, har været brugt som inspiration, hvor dele af deres algoritme er blevet implementeret til at vælge punkter som ligger i hjørner og på linjer. Disse udvalgte punkter betegnes som feature punkterne og er udvalgt ud fra det nye dataset, se på figur 6.2.

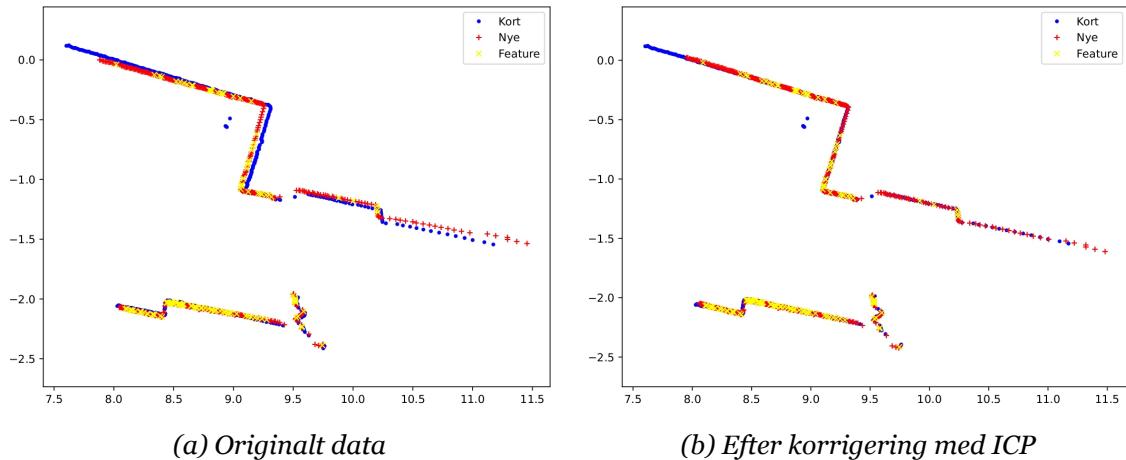


Figure 6.2: Effekt af ICP

Et eksempel på denne algoritmes funktion kan ses på figur 6.2, hvor der på figur 6.2a ses før ICP algoritmen og på figur 6.2b ses efter ICP algoritmen. De blå punkter er det eksisterende kort, og de røde og gule er de nye målinger. De gule er feature punkterne som er anvendt til ICP algoritmen, for at reducere eksekveringstiden på den anvendte Raspberry Pi.

Det er tidligere nævnt at den anvendte lidar sensor er designet til indendørs brug, hvor lysniveaet ikke er for højt, og i stand til at måle op til 4.095 [m]. Dette er en vigtig faktor, fordi lidar sensoren har problemer med præcisionsafmålinger, hvis lyset er for skarpt. Derudover har den problemer med at bestemme afstande til overflader, der er reflekterende, som ved glas og blanke metal overflader. Det er også blevet observeret, at der er en stigende usikkerhed for målingerne i relation til afstanden. Dette gør at afstande fra 3.5 – 4.0 [m] udelades grundet præcisionen, hvor målingerne bliver for korte i forhold til de rigtige afstande. Ved at inkludere data over 3.5 [m] forårsages et forvrænget resultat, som også kan ses på figur 6.2a, hvor de blå punkter på kortet har en svag indadgående kurve, som vil forårsage problemer for SLAM algoritmen. Det skyldes disse punkter vil trække resultatet lidt skævt, og som løbende vil akkumulere med den anvendte algoritme.

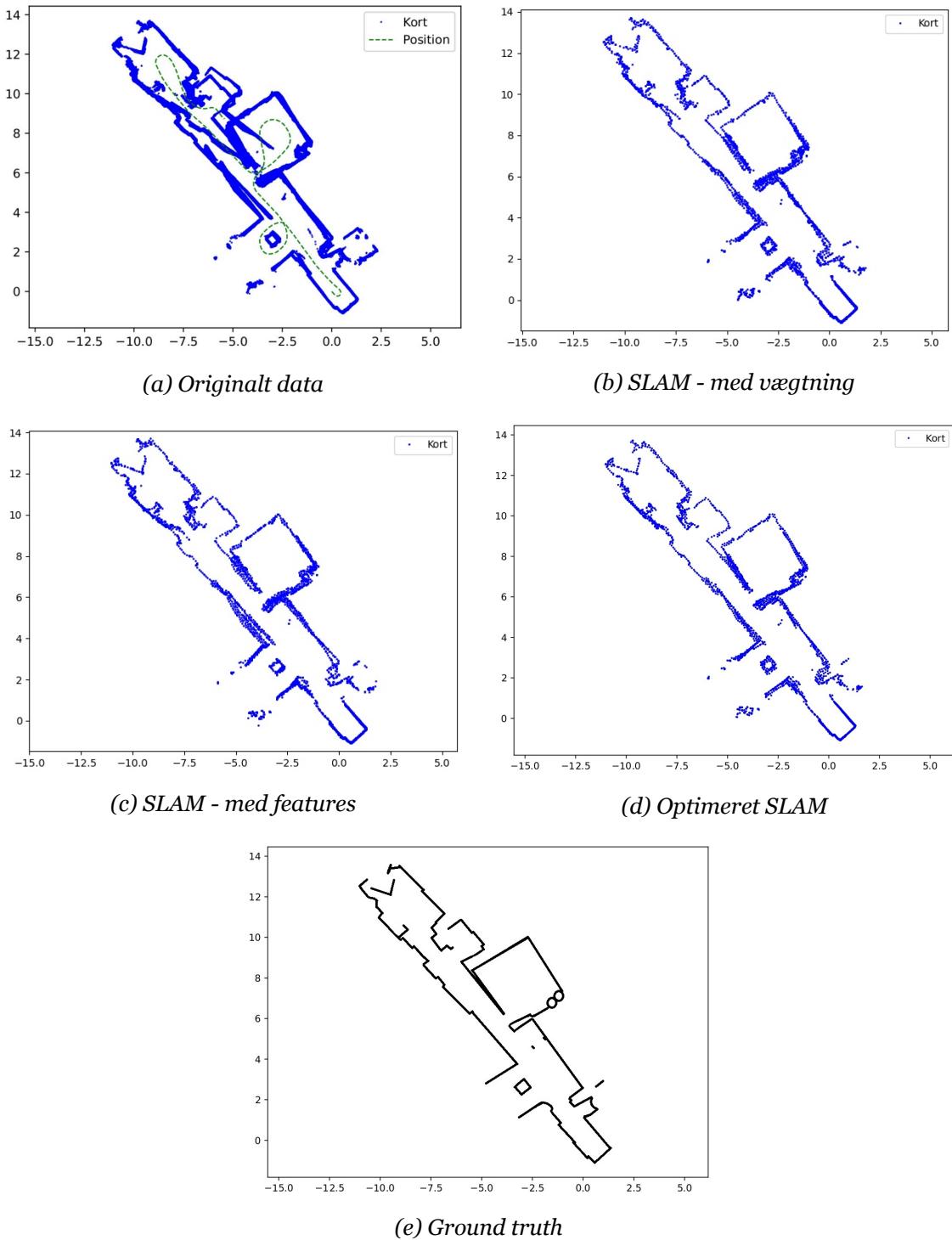


Figure 6.3: Effekten af SLAM, set i forhold til kvaliteten på billede

På figur 6.3e ses ground truth, som er blevet skabt ud fra målinger med lidar sensoren. Disse målinger er efterfølgende blevet kontrolleret, målt op med målebånd, og sammensat manuelt, for at sikre at alt passer overens med opmålingerne. Grunden til at dette er udarbejdet således, er fordi der ikke er findes en plantegning over rummet. Yderligere er det blevet vurderet, at dette gav den bedste præcision med det tilgængelige udstyr.

Efter at have bekræftet at ICP algoritmen virker efter hensigt, er en komplet kortlægning blevet eksekveret, som kan ses på figur 6.3. På figur 6.3a kan det oprindelige kort ses, som er baseret på enkoderne. Dette viser at der er en glidende offset, som gør sig særligt bemærket på orienteringen, hvilket forårsager usikkerheder omkring det faktiske kort, se figur 6.3e. Dette kan overvejende skyldes det tidligere omtalte mekaniske slack i hjulene, som gør sig særligt gældende ved opstart og rotation. Ses der derimod på figur 6.3d, så er det tydeligt at kortet er blevet korrigert til ikke at overlappe den køрte rute, samt en reducering i antallet af usikre punkter. Dog kan det ses at den har udfordringer omkring hjørner sammenlignet med figur 6.3e. Dette resultatet er vurderet til at være tilstrækkeligt, og det afspejler ground truth på figur 6.3e. For SLAM har performance vægtet meget, grundet at algoritmen skal eksekveres på en Raspberry Pi, med begrænset processerkraft. Af denne årsag er loop closure også blevet fra valgt for en forbedret performance på Raspberry Pi'en, hvilket er en algoritme som forbinder målingerne i forhold til tidligere målinger. Dette kræver meget processerkraft for at iterere igennem tidligere målinger og finde sektioner, som passer ind med de nye målinger. Efterfølgende korrigeres alle målingerne mellem de fundene sektioner og frem til den nye måling.

Det kan endvidere ses på figur 6.3c, og på figur 6.3b, hvor meget at vægtningen betyder for det endelige resultat, samt valg af features. Dette er på figur 6.1 vist i forhold til performance, hvor at det her på det endlige kort, tydeligt kan ses en visuel forskel på resultaterne.

## Resultat for SLAM

For at optimere ICP algoritmen blev feature baseret data anvendt, som var målinger der repræsenterede hjørner og lige linjer. Dette reducerer kompleksiteten for proceseren, og leverede stærke målinger. Dette kan se på figur 6.2, hvor det tydeligt fremgår. Ved at bruge alt den tilgængelige data fra målingerne vil det betyde at det ikke var muligt at eksekvere mens robotten kører, og fra figur 6.1, kunne effekten af feature baseret ICP også ses.

Fra figur 6.3 kan det ses at der blev skabt et kort, som er tilstrækkeligt til at lave ruteplanlægning, samt at resultatet var tæt på ground truth. Det kan dog ses at der mangler lidt korrigering af lokalerne, hvilket loop closure kunne have optimeret. Dette var grundet processerkraften ikke en mulighed, og er derfor blevet fravalgt.

I forhold til kravet for dimensionen, som skulle kunne kortlæges, kravspecifikation 2 i sektion 1.1, ses det på figur 6.3, at dette har kunne lade sigøre med en fornuftig kvalitet. Dette indebærer kortlægning i lokaler, for at validere SLAM algoritmen i forskellige scenarier, samt rotering om en søjle.

### 6.3 Partikelfilter

Ved at kortlægge det aktuelle rum som robotten kører i, kan en ruteplanlægning indledes for at køre til et specifikt punkt, der beskrives i henholdsvis sektion 6 og sektion 7. Dette stiller dog et krav til at kunne lokalisere, hvor robotten er i rummet ved opstart. Til at løse det problem kan et partikelfilter anvendes anvendes. Den tager udgangspunkt i at lave en masse partikler på et kortet, som bruges til at validere den aktuelle position. Dette er normalt gjort tilfældigt, og derfor kræver denne metode også mange partikler for at kunne lokalisere, hvor robotten befinner sig. For hver gang at robotten indsamler målinger om rummet, vil datasættet blive simuleret for hver partikel, og undersøge om datasættet passer til partikellen. Ved at fjerne de partikler som ikke passer, og tilføje nye omkring de partikler som potentielt kan lade sig gøre, vil det over flere iterationer resultere i en estimeret position.

For at reducere antallet af partikler som skal fordeles i rummet, indsamlies et datasæt med lidaren, for at indikere robottens start position. Det bearbejdes herefter i python, hvor Open CV's template matching metode anvendes. Dette vil give en indikation om hvor det er muligt at målingen kan lade sig gøre, og hermed begrænse området for partikler. Dette gør at mængden af partikler kan reduceres drastisk, og blive fokuseret omkring estimeringsspunktet. Det skal dog bemærkes at den kan tage fejle og derfor føjes tilfældige partikler også uden for det detekterede område. Resultatet for denne algoritme kan ses på figur 6.4, hvor der er testet 2 positioner, figur 6.4a og figur 6.4c, med respektive resultater figur 6.4b og figur 6.4d.

Template matching kan benytte flere forskellige parametre for template match modes. Den valgte parameter er TM\_CCOEFF\_NORMED og den laver en normaliseret correlation coefficient. Dette betyder at den belønner rigtige pixels, og straffer forkerte, som hermed reducere risikoen for falske detekteringer i tætte områder.

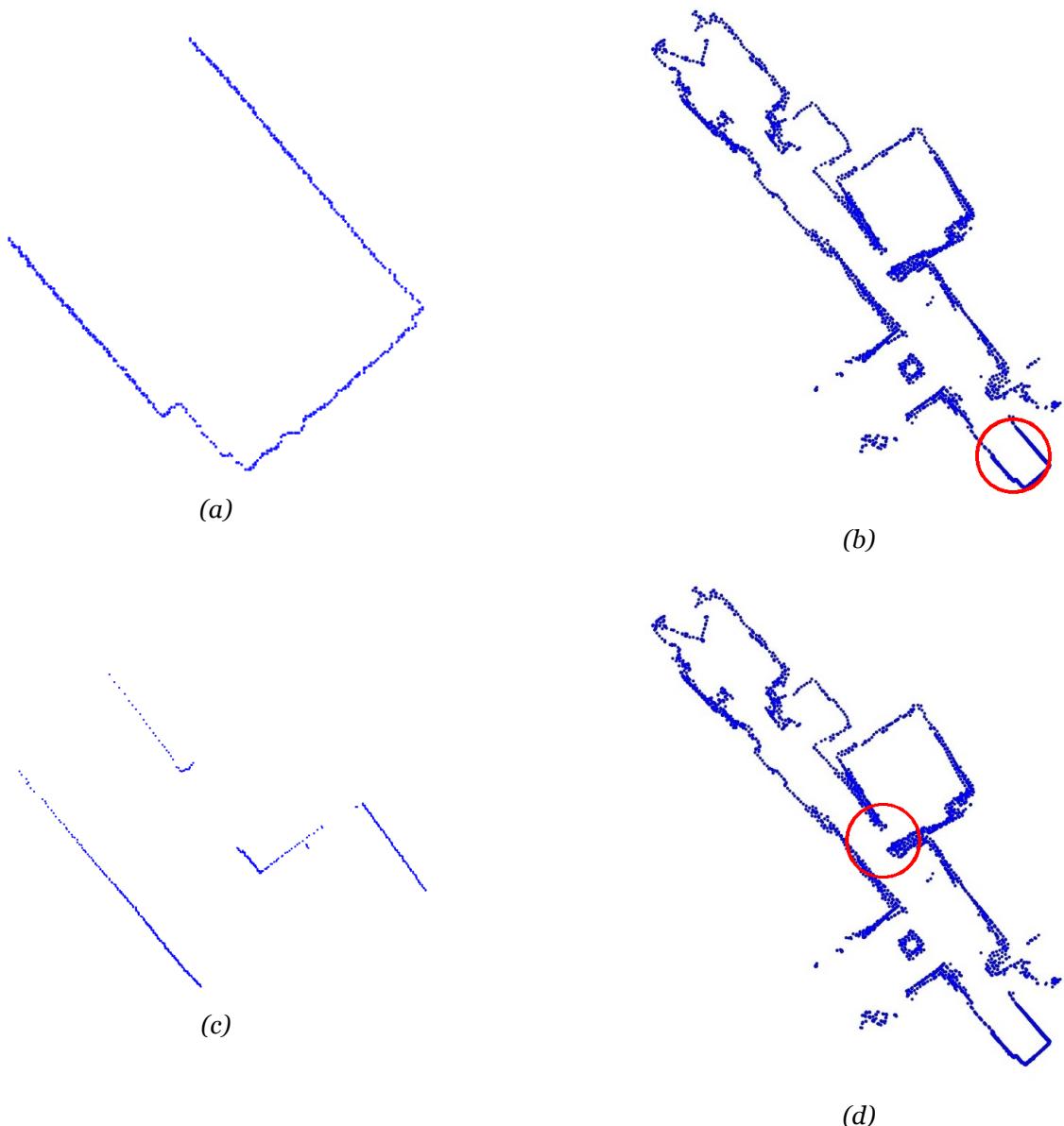


Figure 6.4: Test af lokalisering

Fra resultaterne på figur 6.4 kan det ses, at estimeringen med template matching indskrænker området. Anvendelsen af template matching har dermed reduceret området for partikelfilteret, hvilket vil medføre færre partikler og reducere i kompleksiteten for processen. Dermed kan partikelfilteret konvergere hurtigere mod en løsning. Grunden til at dette er hurtigere end den klassiske implementering af et partikelfilter er, at der fjernes mange tvivlsomme løsninger, som er uden for det detekterede område. Ulempen ses derimod i tilfælde af forhindringer, hvor at denne metode vil kunne have problemer med at finde det rette område, hvis støjen kan forsage at målingerne passer bedre i et område på kortet. Her ville det klassiske parikel filter også opleve problemer, men med en bredere søgning på kortet, vil det medføre at en mere robust løsning.

Grundet prioritering er implementeringen af partikelfilteret ikke blevet færdiggjort.

## 7 Rute planlægning

Til planlægning af ruten er robottens workspace anvendt. Det bruges for at forbedre robustheden for robotten, således at ruten er garanteret at være mulig for robotten, hvilket kan ses på billede figur 7.1a. Her ses at kortet er afgrænset med blå farve i forhold til robottens workspace, og derfor skal der ikke længere tages højde for bredden af robotten, da den hvide del sørger for en større marge ud til kanterne. Derudover bruges det også i tilfælde af forhindringer, for selvom forhindringer ikke krydser ruten, så kan de være inden for robottens workspace. Derfor betragtes dette som en essentiel del af ruteplanlægningen, som simpelt afgrænsrer robottens tilgængelige workspace. Robottens workspace er blevet udarbejdet ved at benytte brushfire metoden. Brushfire udarbejdes ved at man har et grayscale billede, se billede 7.1c, som har ground truth fra billede 7.1d. Man itererer henover billedet i to retninger, fremad og tilbage, hvor man i hver retning øger sin pixelværdi med en enkelt i forhold til den tidligere pixel. Som eksempel kan der ses på den fremadgående iteration. For hver pixel kigger man på den pixel til venstre og oppe over. Hvis disse pixels er af lavere værdi end den pågældende, så bliver den pågældende pixel én værdi højere end den laveste. Når det er gjort for alle pixels i begge retninger, såender man ud med billede figur 7.1b, hvor det gradvist øger sin lysstyrke, som desto længere væk fra vægge og forhindringer at man kommer. Der er benyttet en threshold på 10, så alle pixels, der har en højere værdi, bliver farvet blå. Koden kan ses i den vedlagte zip fil i mappen 'brushfire'.

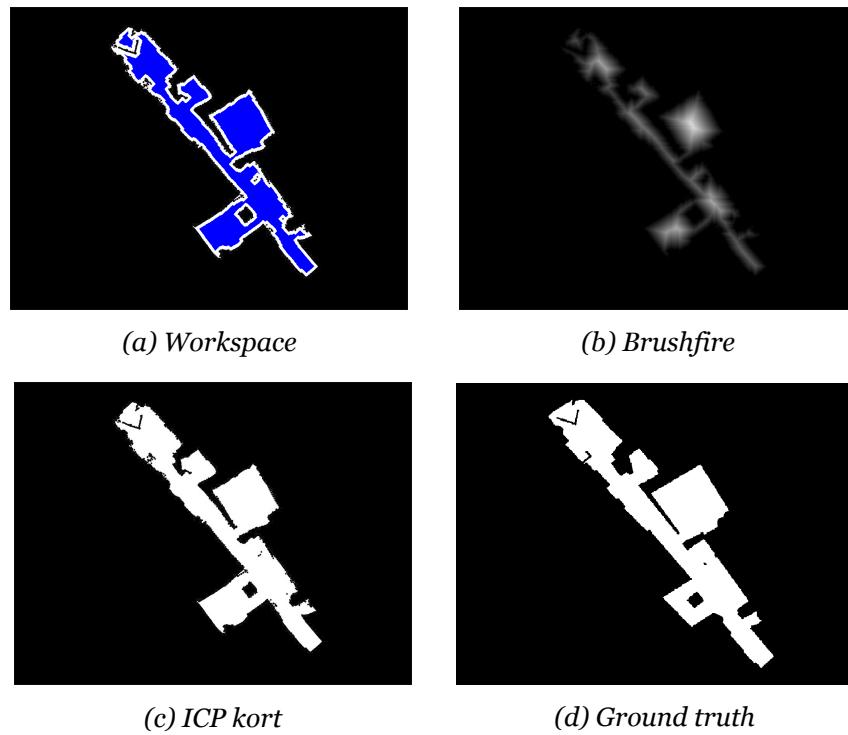


Figure 7.1: Workspace og Brushfire

Ved at lave et workspace på kortet er der blevet taget højde for robotten. Det vil simplificere ruteplanlægningen ved at fjerne utilgængelige områder og sikre en minimumsafstand til forhindringer.

## 7.1 Generaliseret Voronoi Diagram

Ud fra det kort, som er blevet genereret ønskes det at lave et generaliseret voronoi diagram, forkortes GVD. Formålet med GVD er at finde linjer, som er længst væk fra kendte forhindringer, så robotten kan navigere med størst sandsynlighed for at undgå at ramme objekter. GVD illustrerer medianen imellem to forhindringer, altså når afstanden imellem de to er lige store, så markeres det på kortet med en prik, og samlet illustrerer disse en linje.

På figur 7.2 ses seks billeder. Der tages udgangspunkt i figur 7.1c, der inden behandling er eroderet og udvidet (dilated), og ud fra dette, så tester vi tre forskellige algoritmer fra `skimage.morphology` [21]. Metoderne har den fælles ting, at de har til opgave at repræsentere et objekts topology, altså formens skelet. Det gør de ved at lave en én pixel bred illustration. De tre metoder, ses på figurene 7.2a, 7.2b og 7.2c. Af de tre metoder, så er skeleton lee den mest skrabet, mens medial axis skeletedannelse giver flest linjer, som går ud til hjørnerne. På figurene 7.2d, 7.2e og 7.2f er de tre roadmaps, som bliver sat oven på billedet. Der er med en  $3 \times 3$  kernel optalt antal pixels omkring hvert punkt for at beslutte, om punktet er et samlingspunkt, og der derfor skal placeres et punkt på GVD kortet. Når man kigger på de forskellige algoritmer, så kan man se, at de har tre niveauer. `Skeletonize Lee`'s method dækker mindst, `skeletonize` er noget bedre, da den er bedre til at komme ud til hjørnerne, hvor `Medial Axis Skeletal dannelse` ender med at komme ud kanten for mange steder, da den anser cirklerne fra punkterne som hjørner.

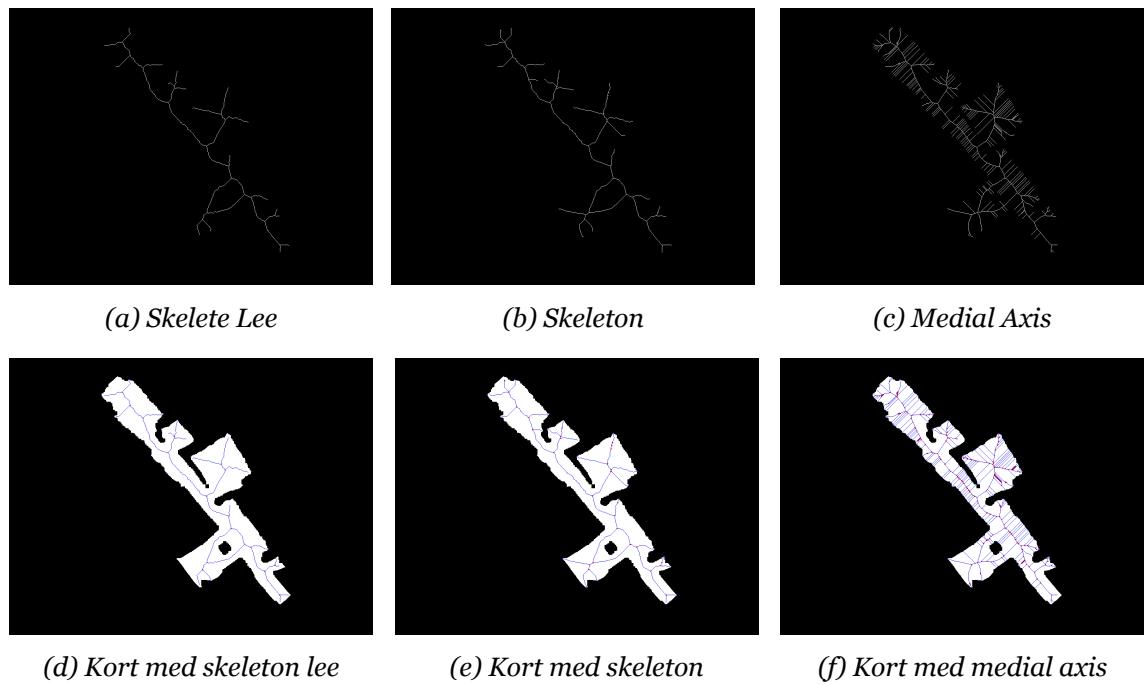


Figure 7.2: GVD

Det er blevet besluttet, at der benyttes skeletonize til at konstruere et GVD. Metoden sørger for at ramme det bedste antal af linjer, så algoritmen næsten når ud til alle hjørner. Metoden hedder ikke GVD, men tilgangen til konstruktion af roadmap'et og slutresultatet er det samme. Metoden er ikke perfekt, men de fejl, som den så laver er små og det værste scenario vil blot blive en længere behandlingstid af algoritmen til er bestemme vejen. Desto mindre ujævnheder og skaeve hjørner, der er, desto bedre er Medial Axis Skeletdannelse. Hvis der benyttes et mindre komplekst billede, hvor der er lodrette og vandrette linjer, så formår Medial Axis Skeletdannelse at leve et helt korrekt GVD af kortet. Se bilag A for illustrationer af et kort med lodrette og vandrette linjer, samt algoritmen på ground truth billedet. Koden kan ses i den vedlagte zip fil i mappen 'voronoi'.

## A\* algoritmen

Med noderne fra GVD algoritmen, kan der laves en søgning, som finder den korteste rute mellem to punkter. Til at lave denne rute kan ruteplanlægningsalgoritmer som Dijkstra og A\* anvendes. Begge tager udgangspunkt i at finde den korteste rute mellem 2 punkter, hvor ruten sker gennem noder. Ved Dijkstra tages udgangs punkt i at finde forbindelser mellem noder, således at der er den korteste rute fra start ud til en hver node. Dette gøres ved at vægte afstanden mellem noderne. Hertil er vægten  $g$ , som betegner den akkumulerede vægt fra start til den pågældende node. Dijkstra algoritmen vælger for hver iteration den node med den laveste vægtning, og udforsker kortet til at start er forbundet med alle noderne. De samme principper bygger A\* på, som endvidere bruger vægten  $h$ , som er den heuristske funktion. Den heuristske funktion estimerer vægten fra noden til målet, og med denne tilføjelse bliver ruteplanlægningsalgoritmen betydeligt hurtigere, eftersom noderne også bliver vægtet i forhold til målet. Dette gør at algoritmen vil forsøge at bevæge sig i retning af målet og undlade udforskning af noder i en anden retning, hvis det kan undgås. Derfor er A\* også at foretrække som ruteplanlægningsalgoritmen i dette projekt. Fremgangsmåden af A\* kan ses som pseudokode i algoritme2. Her fremgår det klart, at A\* kun validere noder som er omkring udforskede noder, og derfor er optimal for projektet.

---

**Algorithm 2** A\* algoritme

---

```

Input: start_node, target_node, H           ▷ H is the heuristic function
Output: path
Ensure: start_node ≠ null
Ensure: target_node ≠ null    Ensure: H ≠ null

1: procedure A*
2:   openList = null
3:   closedList = null
4:   openList.add(start_node)                  ▷ start_node F value = 0
5:   while (openList ≠ empty) do
6:     current_node = min(openlist)            ▷ Find the smallest F
7:     openList.remove(current_node)
8:     closedList.add(current_node)
9:     if (current_node = target_pos) then
10:      return path                          ▷ Path is found
11:      end if
12:      child_nodes = current_node(adjacent nodes) ▷ List of adjacent nodes
13:      while (node in child_nodes ≠ null) do
14:        F_score = g_score(current_node) + H(current_node, node)
15:        if (F_score < openlist(node)) then      ▷ New shortest path to the node
16:          Update closedList
17:          if (node not in openList) then
18:            openList.add(node)
19:          end if
20:        end if
21:      end while
22:    end while
23:    return failure                         ▷ No path has been found
24: end procedure

```

---

A\* og GVD har en god synergি, da A\* hurtigt kan søge efter den korteste rute imellem to punkter, og GVD begrænser antallet af punkter, som den kan søge på, ved kun at ligge en rute med den største afstand til forhindringerne. Det reducerer antallet af områder, som A\* kan søge markant, samtidigt med at robotten har den mindste sandsynlighed for at ramme kendte objekter.

## 7.2 Cubic spline

Efter at have fundet en rute af punkter placeret for hver retningskifte, er en mere hensigtsmæssig rute blevet udarbejdet. Dette er gjort ved anvendelse af en cubic spline, som forbinder punkterne med stykkevis tredjeordens polynomier. Dette jævner herved svingene ud, og medfører at robotten kan have en mere jævn kørsel. Dette kan ses på figur

5.3, hvor at reference ruten er efterbearbejdet med cubic splines. Resultatet ved at anvende cubic spline kan ses på figur 7.3. Billedet viser et eksempel over hvordan punkter kan forbindes. og her ses fordelen ved at anvende cubic spline tydeligt. Hver gang at der kommer et hårdt sving kræver det at robotten stopper bevægelse på det ene hjul, og derfor medfører en lineær forbindelse mange stop og starter. Ved spline ses det derimod at robotten skal køre en støre omvej for at kommer gennem punkterne. At minimere ruten og samtidigt sørge for bløde overgange mellem punkter, er derfor fordelagtigt for robotten, for at sikre den hurtigste og meste jævne kørsel. Dertil formår en cubic spline at løse opgaven, hvor at den både holder en kort afstand til punkterne, og undgår unødvendige store sving[22].

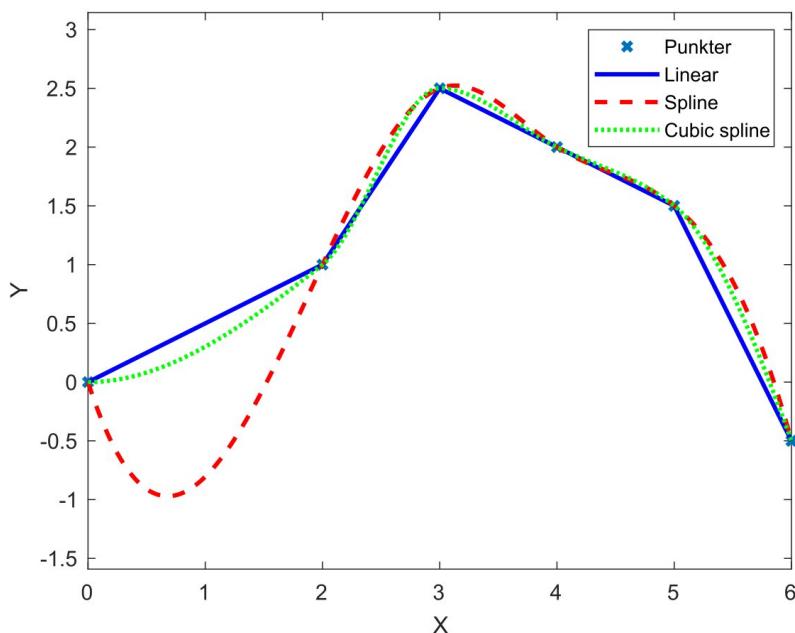


Figure 7.3: Sammenligning mellem lineær, spline og cubic splic, med fiktive data punkter

### 7.2.1 Path coverage

Planen var at anvende en path coverage algoritme, som optimalt kunne lave en rute, til at kører rundt i søgen efter BLE sensorene. En sådan algoritme der kunne være anvendt er en Cell Decomposition efterfulgt af Coverage Planning for hver celle. Således ville robottens kammera kunne opfange alle synlige overflader i lokalet. Dette er essentielt for at sikre at BLE sensorene ikke kan blive overset grundet de hænger på en overflade bagved robotten.

# 8 Forhindringsdetektering

## 8.1 Lidar

Til forhindringsdetekteringen bruges både lidaren og kameraet. Lidaren virker ved at den foretager målinger, som kan sammenlignes med kortet og derved finde forhindringer. Derfor vil alle detekteringer, som afviger fra kortet blive betragtet som værende forhindringer, og et nyt midlertidigt kort vil blive dannet. Dette gør også at det er vigtigt at målingerne er gode, for i tilfælde af støj vil støjmålinger fremgå som forhindringer. Derfor bruges en threshold, som filtrerer hovedparten af støjen væk, og efterlader kun reflekterende overflader som et problem. Dette nye kort bruges til validere, om robottens rute er fri for forhindringer. Hvis det ikke er tilfældet vil en ny rute blive dannet for at omgås forhindringen.

Lidarens forhindringsdetektering har både fordele og ulemper. Først og fremmest er data målingerne både simple og hurtige at processere, som herved giver en klar indikation om omgivelserne. Ulempen derimod er, at der sættes krav til forhindringerne. Disse forhindringer skal være højre end 12.5 [cm], og bredere end 1 [cm]. Dette skyldes at lidaren mäter horisontalt ved 12.5 [cm] højde, og hvis der opfanges for få data punkter, er det svært at vurdere om det skyldes støj. Derved kan den simple threshold fjerne meget af støjen. Yderligere må forhindringerne ikke reflektere laserne, eftersom dette vil påvirke datamålingerne.

På figur 8.1 kan et eksempel ses på hvordan forhindringer vil blive betragtet. Her ses det på figur 8.1a at den ene boks er for lav og ikke er detekteret på figur 8.1b. Datasættet fra figur 8.1b vil kunne sammenlignes med kortet, til at detektere forhindringerne.

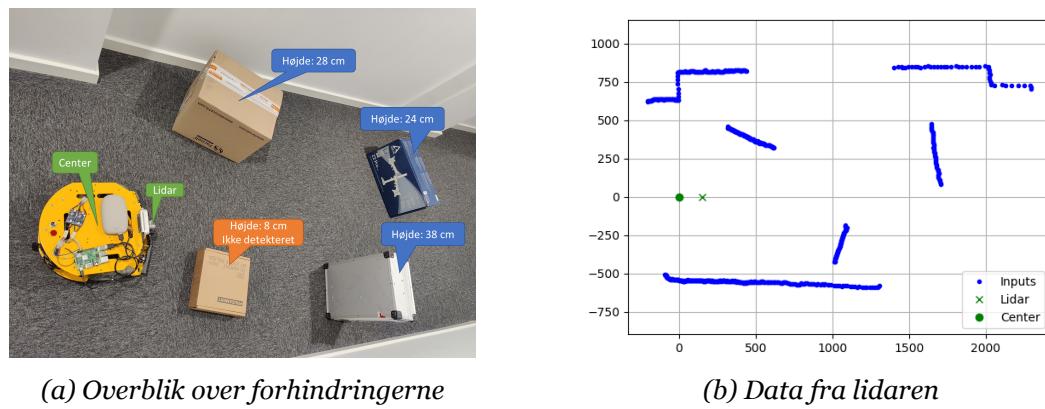


Figure 8.1: Test af forhindringsdetektering

Lidar løsningen er effektiv og lever et fornuftigt argument til at tage beslutninger i forhold til forhindringer. Dette kan se på figur 8.1, hvor den laver robuste detekteringer af forhindringer. Trods de gode resultater har lidar løsningen også ulemper. Den mest åbenlyse

ulempe er i forhold til, at den ikke kan detektere objekter under 12.5 [cm], hvilket kan give nogle udfordringer, som ses på figur 8.1. Derudover ses en ulempe i tilfælde af reflekterende overflader, hvor støj bliver et større problem for målingerne. Derfor er det nødvendigt at tilføje en anden sensor typer, for at opnå en sikker kørsel uden at kolidere med forhindringer. Dette er gjort i form af et kamera.

## 8.2 Kamera

Kamerets opgave er yderligere at formindsker risikoen for sammenstød. Kameraet sidder i en højde af 15 [cm] over jorden, og 5.5 [cm] bag ved front bumperen. Det betyder at, når robotten kører, så kan den først se underlaget 45 [cm] ude foran sig. Hvis man drejer tæt omkring et hjørne, eller hvis objektet på anden vis kommer ind under synsfeltet, vil robotten overse det og muligvis støde ind i det. Objekter over 12.5 [cm] i højde vil blive opfanget af lidar sensoren, og ellers vil de støde imod bumpersensoren. Der bliver gennemgået nogle af de valg af processor, som bruges til at komme frem til en løsningsmulighed i de følgende afsnit.

### Kantdetektering og sløring

Under udarbejdning af algoritmen til objektundgåelse er der blevet testet med forskellige typer af kantdetektering. Det første forsøg blev imellem Canny, Sobel og Laplacian kantdetektering. På figur 8.2 ses tre billeder med forskellige typer af kantdetektering, som alle har fået sløret billede forinden, med et gaussian filter. Af de tre billeder fremstår billede 8.2b meget u tydeligt og med mange små 'korn' på billedet, hvilket ikke er særlig attraktivt, hvis man tager udgangspunkt i algoritmen skulle have fjernet disse. Billede 8.2c har nogle fine linjer langs væggen, men fejler, når den skal detektere langt væk. Billede 8.2a har det bedste resultat af de tre, og derfor er det valgt at gå videre med Canny kantdetektering.

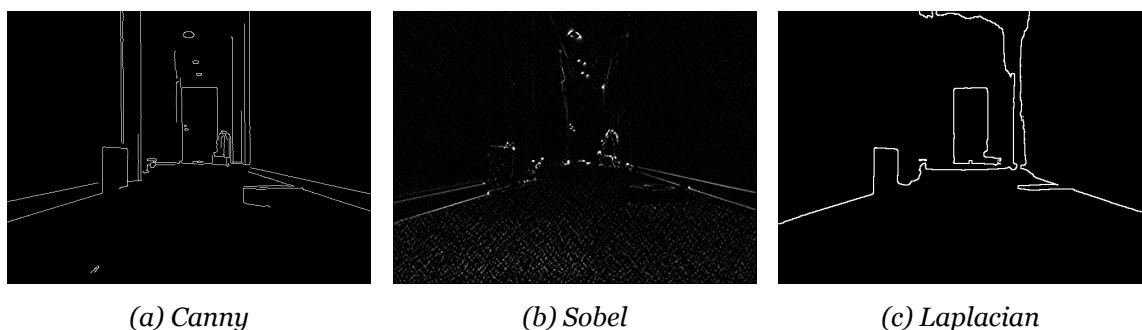


Figure 8.2: Test af kantdetekteringstyper

Den næste beslutning er omkring typen af sløring, som billede skal have inden det gennemgår en kantdetektering. På figur 8.3 ses fire identiske billeder, som hver har deres overlag med objektundgåelsesalgoritmen. Af de fire billeder klarer billede 8.3a sig dårligst, og det er på baggrund af at de mange små 'korn' fylder for meget i billede. Billede 8.3b

klarer det markant bedre, men fejler lidt på nogle kanter, så det område, som burde være tilgængeligt rammer begge forhindringer, og det samme gælder for billede 8.3d. Billede 8.3c klarer sig helt klart bedst, og undgår alle forhindringer på billedet. Det er ikke et enestående scenarie, og det er sjælendt at de andre klarer sig bedre, så derfor er det blevet besluttet kun at benytte median blur til sløring af billedeerne.

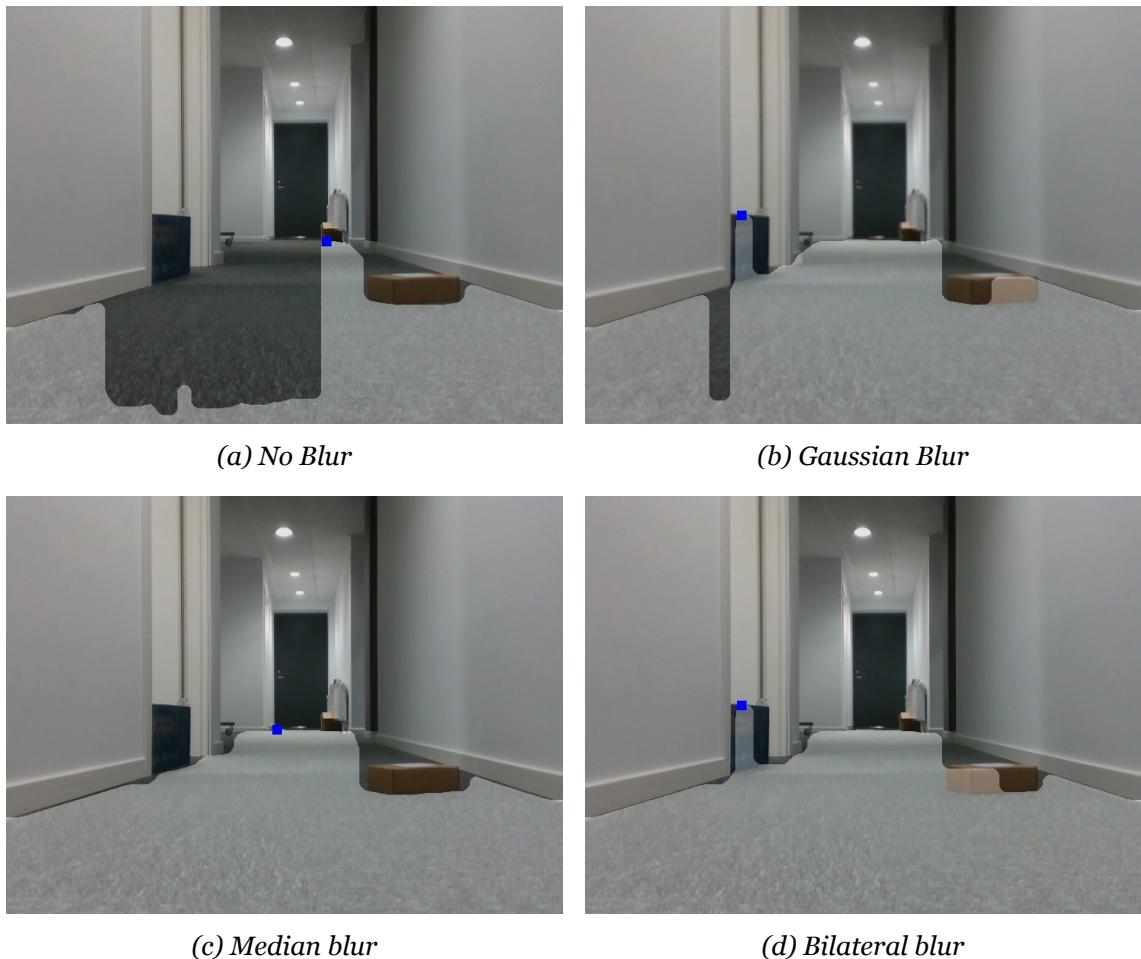


Figure 8.3: Test af Canny kantdetektering

## Objektundgåelsesalgoritmen

På figur 8.4 ses seks billeder. Disse seks billeder viser de seks trin, der er i objektundgåelsesalgoritmen. Billede 8.4a viser det originale billede, som vi tager udgangspunkt i. Billede 8.4b viser en kantdetektering af billedeet, hvor de vigtigste features findes, der skal benyttes til at danne et korrekt kort over forhindringer forude. Efter billedeet er blevet sløret, så benyttes Canny kantdetekteringsmetoden, som er fra OpenCv. Når kantdetekteringen er på plads, så udføres en sidefill, se billede 8.4c. Sidefill kører fra venstre mod højre, og for hver kolonne fylder den fra bunden med hvide pixels indtil den rammer en hvid pixel fra billedet med kantdetektering.

Sidefill har lidt problemer, hvis en kant ikke er fuldendt fra billede 8.4c. For at minimere de problemer, som det skaber, så behandles billedeet med en horizontal erodering og ud-

videlse (dilate), som ses på billede 8.4d. Den bruges til at lukke huller, når der detekteres små 'korn' på overfladen. En bivirkning af det er, hvis en kant starter lodret fra bunden af billedet. Det betyder så at f.eks. pakken på dette billede bliver anset som gulv. Det kan ses i højre side af billede 8.4c. Til sidst laves en erodering igen, så den fyldte del ikke går ind over de kantdetekteringer, som den tidligere har sat.

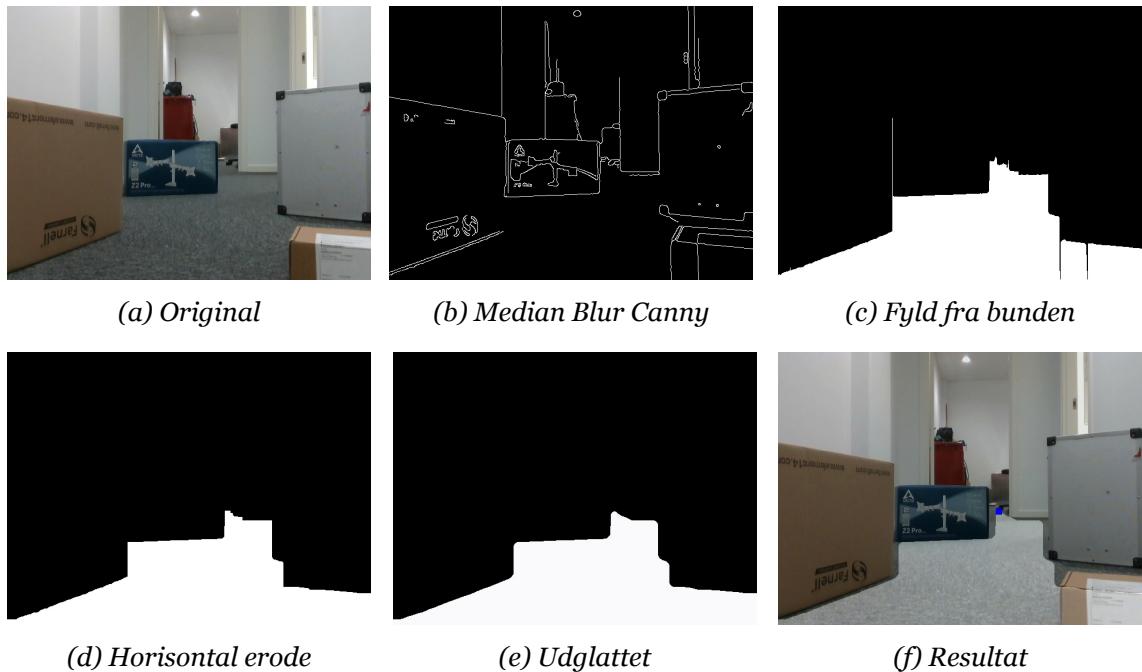


Figure 8.4: Gennemgang er objektundgåelsesalgoritme

Udglatningen af billedet bliver gjort efterfølgende, for at få et simplerere resultat, som ses på billede 8.4e. Det er udelukkende af æstetiske årsager, og efterfølgende, på baggrund af tests for behandlingstid, er det erfaret, at algoritmen for udglatningen er meget ressourceneffektiv. Udglatningen benytter sig af `ImageFilter.ModeFilter` modulet fra `Pillow` med en kernel på  $13 \times 13$ . Denne metode finder den hyppist forekommende pixel i en kernel og erstatter den pixel, som den behandler med denne, som gør den praktisk til at afrunde hjørner[23]. Ud fra algoritmen af behandling af billedet, oplevede vi at den var lidt langsom, når den kørte udglatningen, og vi har derfor på Raspberry Pi 'en testet behandlingstiden for 17 billeder og taget gennemsnittet. Behandlingstiden for billederne ligger indenfor en  $1/10$  af hinanden, så der er ikke en stor variation imellem billederne. Den sammenlagte tid for testen falder fra et gennemsnit på 2.24 [sek] til 0.35 [sek], hvilket er en forbedring på 640 [%]. Af den årsag er det blevet besluttet at fjerne smoothing for at optimere tiden, som det tager at behandle objektundgåelsen.

Det endelige resultat ses på billede 8.4f. Inden de to billeder kombineres, så findes den første instans af en hvid pixel på billede, og her tilføjes en prik, for at markere hvad robotten ser som værende fjerneste punkt på gulvet. Afslutningsvis laves der en bitvis sammenfletning af det originale billede og det udglattede billede for at vise resultatet.

Denne objektundgåelsesalgoritme fanger ikke alle forhindringer korrekt, og den fejler ind imellem. Det er derfor planen at processen skal samarbejde med lidar sensoren, men dette er ikke bliver implementeret grundet mangel på tid og prioritering. Flere resultater kan ses i bilag B. Koden kan ses i den vedlagte zip fil i mappen 'obstacleavoidance'.

En forbedringsmulighed for kameraet kunne være at placere det tættere på jorden, hvorfra man ville kunne minimere afstanden foran robotten, hvor objekter er skjult for kameraet. Men som udgangspunkt, er det ikke et problem, som vi forventer at støde på særligt ofte, hvis overhovedet. Det skyldes ruteplanlægningsalgoritmen som vil bestræbe robotten på at køre midt imellem forhindringer.

Lidaren kan bruges til at opdatere kortet, der navigeres efter og afmåle afstand til forhindringerne. Den har begrænset data, men er derimod hurtig at afvikle på processeren. Til at opsamle yderligere informationer omkring forhindringerne anvendes et RGB kamera. Da kameraets behandlingstid alene er for langsom til at være realtid, så er den samlede process for kamera og lidar heller ikke i realtid. Dette lever ikke op til kravet fra kravspecifikation 4 i sektion 1.1 omkring realtime.

## Homografi

For at mappe kameraets billede over til data, som kan samarbejde med lidar sensoren, så skal der defineres en homografi for mapning imellem kameraets billedplan over til gulvet. En af forudsætningerne for at undgå objekter er, at robotten kører på en flad overflade, ergo der skal ikke tages højde for ujævne overflader / offroading. Det antages derfor at homografien for en enkelt position vil kunne anvendes under kørsel. Der tages forbehold for krumninger eller trin og disse medregnes ikke, som tilstedevarende i testmiljøet, og objekter vil blive frasorteret af undgåelsesalgoritmen. Det er en opgave, som ikke nåede at blive udført, da det kræver at kameraet var blevet kaliberet inden homografien laves, ellers vil det give forkerte resultater.

## Transfer learning til objektundgåelse

Neurale netværk eller transfer learning kan bruges til Objektundgåelse. Det kræver at netværket trænes med de objekter, som man forventer at se på billedet. En tilgang til det vil være at indramme alle forhindringer, som ikke er væggen på billedeerne, så modellen for det neurale netværk også leder efter forhindringer. En anden tilgang er at benytte semantic segmentering, som tildeler alt på billedet en etiket. Etikkerne kan uddover at være forhindringer og BLE sensorer, også være gulve, vægge og døre. Alle disse bidrager med information til at finde rundt i rummet. Disse to processor vil dog begge kræve endnu mere af det neurale netværk, som i forvejen er ressouretungt.

## 9 Konklusion

Med LQR og PID regulatorer kan robotten køre robust efter hensigten fra en hastighed på  $0.5 \frac{\text{km}}{\text{t}}$ . Kører robotten langsommere end det, har PID regulatoren udfordringer grundet motorene ikke kører ens, samt robotten vejer mere i den ene side, og derfor kræver mere kraft for at starte. Grundet mekaniske slack i enkoderne er SLAM implementeret, som anvender lidar målinger, der afhjælper afvigelser i målingerne.

Det blev i de tidlige faser af projektet konkluderet, at et 3D kamera ikke var en optimal løsning for kortlægning en Raspberry Pi. Dette skyldes at 3D kammeraet indsamler rigtig mange data målinger for hvert frame, og det derfor er tungt at processere. Kamera målingerne kan reduceres fra 3D til 2D, men her viste lidaren store fordele ved at have et brede synsfelt end kammeraet. Ved at bruge en lidar form af SLAM, og anvende informationerne omkring positionen fra enkoderne, så er det muligt at opnå et kort som kan bruges til navigering. Hertil er datamålingerne også vægtet og der er udført featureudtrækning. Dette har reduceret kompleksiteten og øget robustheden for kortlægningen.

Forhindringsdetekteringen leverer detekteringer foran robotten med fejlbehæftelser i. Hvis objektet er for tæt på robotten, og bunden ikke er inde i billedet, kan objektet fejldejtes, så objektet skal minimum være 45 [cm] foran robotten for at mindske fejldetekteringer. Der er derfor plads til forbedring af metoden i objektundgåelse. Dog assisteres den af lidar sensoren, og denne finder forhindringer, hvis de er over 12.5 [cm] høje. En mapning af kameradata over på lidar data vil kunne samle dataene og forbedre objektundgåelsen, men dette kræver yderligere arbejde, som ikke er udført i projektet.

Ved at benytte segmentering sammen med transfer learning opnås der en detekteringspræcision på 98.36 % med BLE sensorer på en afstand op til 4 m væk, hvilket er et tilfredsstilende resultat.

Der er blevet udviklet et produkt med mange tilfredsstillende kvaliteter baseret på ovenstående resultater. Systemet fungerer dog ikke sammen i helhed, hvilket vil kræve en kraftigere processor og at et main program skal kunne kontrollere processerne.

# Litteraturliste

- [1] Bionic System Solutions. *BSS Aps, Adresse Østre Stationsvej 41P 5000 Odense C, CVR: 38378007.* url: <https://bionicsystemsolutions.com/>. (accessed: 28/10-2021).
- [2] Det Tekniske Fakultet Syddansk Universitet. *Afgangsprojekt på diplomingeniørudannelsen.* url: [https://mitsdu.dk/da/mit\\_studie/diplomingenioer/diplomingenioer\\_i\\_robotteknologi/eksamen/afsluttendeopgaver/diplomingenioer](https://mitsdu.dk/da/mit_studie/diplomingenioer/diplomingenioer_i_robotteknologi/eksamen/afsluttendeopgaver/diplomingenioer). (accessed: 28/12-2021).
- [3] Nexus Robot. *2WD Mobile Robot Kit, Nexus Robot.* url: <https://www.robotshop.com/media/files/pdf/datasheet-10004.pdf>. (accessed: 04/10-2021).
- [4] Hokuyo. *Lidar, URG-04LX, datablad.* url: [https://www.hokuyo-aut.jp/dl/Specifications\\_URG-04LX\\_1513063395.pdf](https://www.hokuyo-aut.jp/dl/Specifications_URG-04LX_1513063395.pdf). (accessed: 05/10-2021).
- [5] Intel Corporation. *3D kamera, Intel Realsense D435, datablad.* url: <https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf>. (accessed: 06/10-2021).
- [6] Intel. *Intel Realsense Specifications.* url: <https://www.intelrealsense.com/depth-camera-d435/>. (accessed: 16/12-2021).
- [7] Cheng H.D. et al. *Color image segmentation: advances and prospects.* url: <https://www.sciencedirect.com/science/article/pii/S0031320300001497>. (accessed: 30/12-2021).
- [8] Shelby Zach and Jongboom Jan. *Edge Impulse.* url: <https://www.edgeimpulse.com>. (accessed: 28/12-2021).
- [9] Howard Andrew et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* url: <https://arxiv.org/pdf/1704.04861.pdf>. (accessed: 28/12-2021).
- [10] Becker Dan S. *ReLU.* url: <https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning>. (accessed: 02/01-2022).
- [11] Sahoo Sabyasachi. *Residual Blocks.* url: <https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec>. (accessed: 31/12-2021).
- [12] Sandler Mark et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks.* url: <https://arxiv.org/pdf/1801.04381.pdf>. (accessed: 27/12-2021).
- [13] Sears Collins Addison. *LQR simulering implementering.* url: <https://automaticaddison.com/linear-quadratic-regulator-lqr-with-python-code-example/>. (accessed: 07/12-2021).
- [14] Beauregard Brett. *Arduino PID library.* url: <https://github.com/br3ttb/Arduino-PID-Library>. (accessed: 07/12-2021).
- [15] Triantafyllou Prof. Michael. *LQR lecture.* url: <https://ocw.mit.edu/courses/mechanical-engineering/2-154-maneuvering-and-control-of-surface-and-underwater-vehicles-13-49-fall-2004/lecture-notes/lec19.pdf>. (accessed: 22/12-2021).

- [16] aa4cc. *L4.4 - Discrete-time LQ-optimal control - infinite horizon, algebraic Riccati equation.* url: <https://www.youtube.com/watch?v=wLTx0HiK6G0>. (accessed: 22/12-2021).
- [17] MathWorks. *SLAM (Simultaneous Localization and Mapping)*. url: <https://se.mathworks.com/discovery/slam.html>. (accessed: 23/11-2021).
- [18] Pomerleau François et al. *Comparing ICP Variants on Real-World Data Sets*. url: <https://stephane.magnenat.net/publications/Comparing%20ICP%20Variants%20on%20Real-World%20Data%20Sets%20-%20Pomerleau%20et%20al.%20-%20Autonomous%20Robots%20-%202013.pdf>. (accessed: 14/12-2021).
- [19] Stachniss Cyrill. *Mobile Sensing And Robotics*. url: <https://www.ipb.uni-bonn.de/msr2-2021/>. (accessed: 14/12-2021).
- [20] Gao Haiming et al. *A line segment extraction algorithm using laser data based on seeded region growing*. url: <https://journals.sagepub.com/doi/pdf/10.1177/1729881418755245>. (accessed: 09/12-2021).
- [21] scikit. *Skeletonize*. url: [https://scikit-image.org/docs/dev/auto\\_examples/edges/plot\\_skeleton.html#sphx-glr-download-auto-examples-edges-plot-skeleton-py](https://scikit-image.org/docs/dev/auto_examples/edges/plot_skeleton.html#sphx-glr-download-auto-examples-edges-plot-skeleton-py). (accessed: 26/12-2021).
- [22] mathworld. *Cubic spline*. url: <https://mathworld.wolfram.com/CubicSpline.html>. (accessed: 23/11-2021).
- [23] Pillow. *Smoothing corners*. url: <https://pythontic.com/image-processing/pillow/mode%20filter>. (accessed: 14/12-2021).

## A Generaliseret Voronoi Diagram

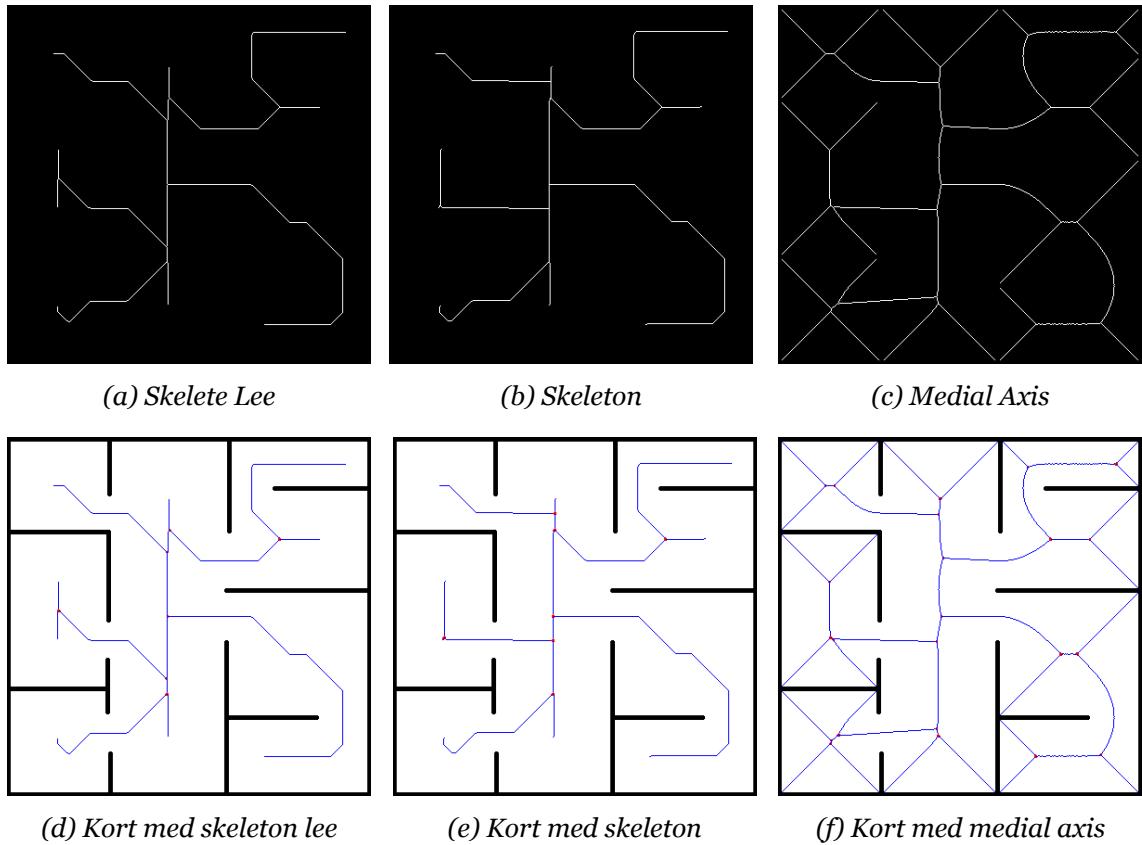


Figure A.1: GVD.

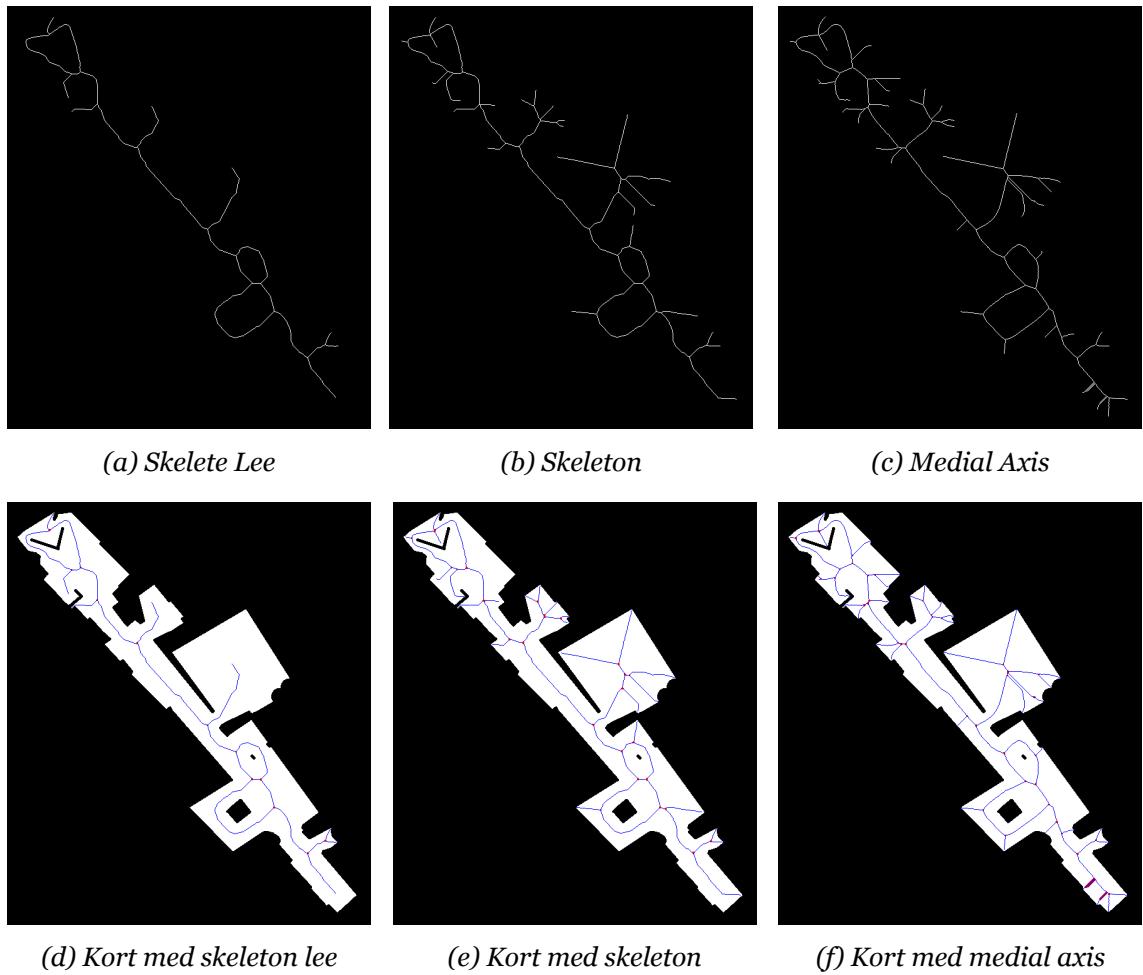


Figure A.2: GVD.

## B Kode for segmentering

### Objektundgåelse



Figure B.1: Billed 1



Figure B.2: Billed 2



Figure B.3: Billede 3



Figure B.4: Billede 4



Figure B.5: Billede 5

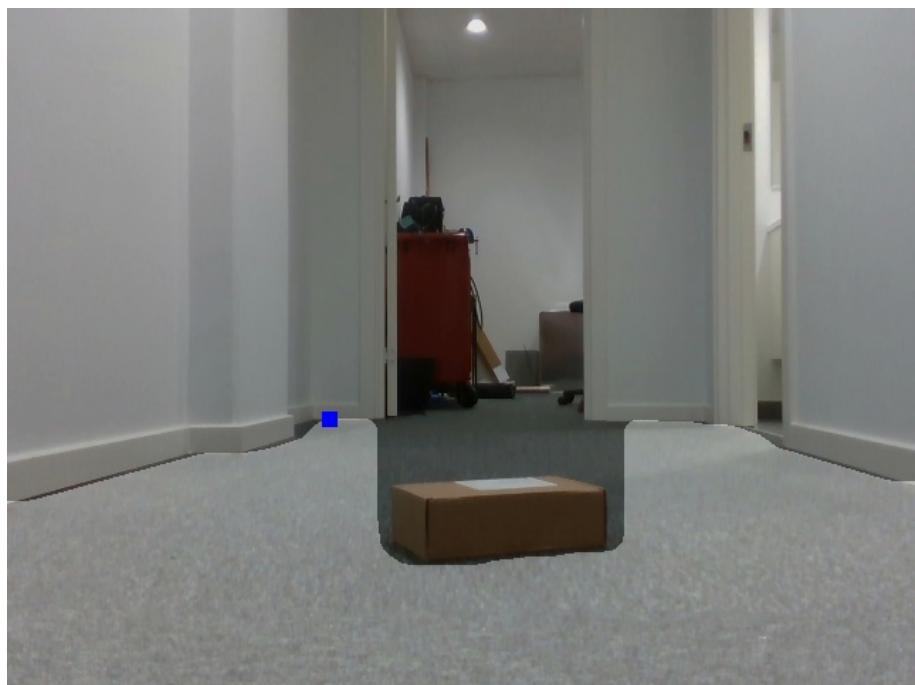
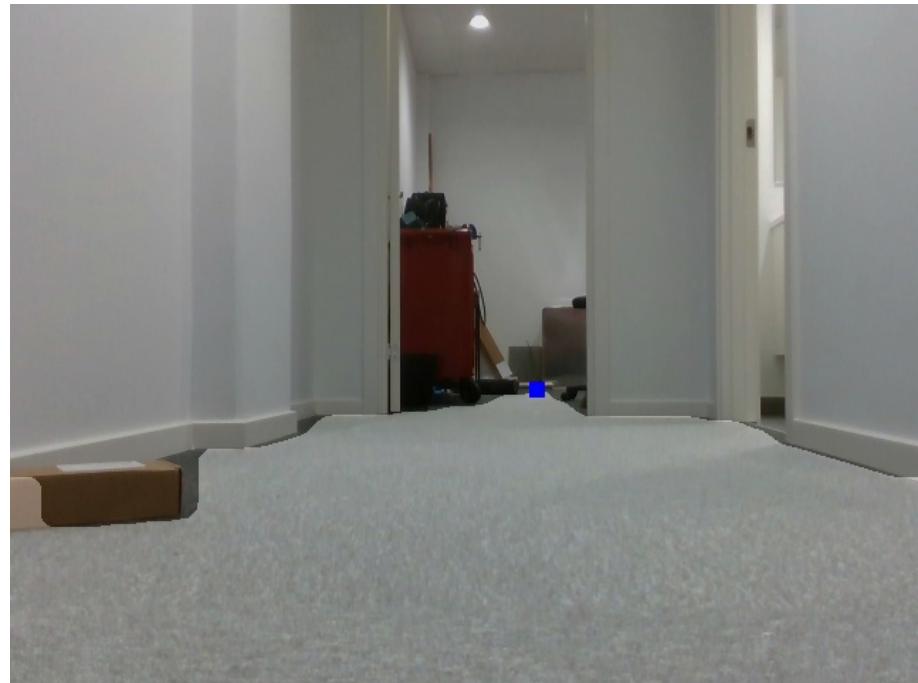
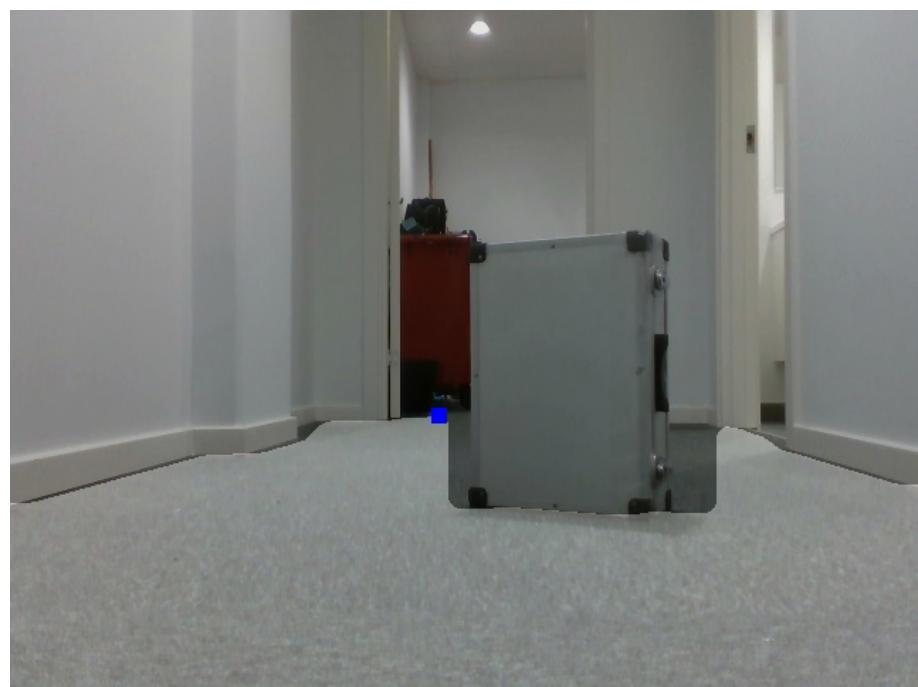


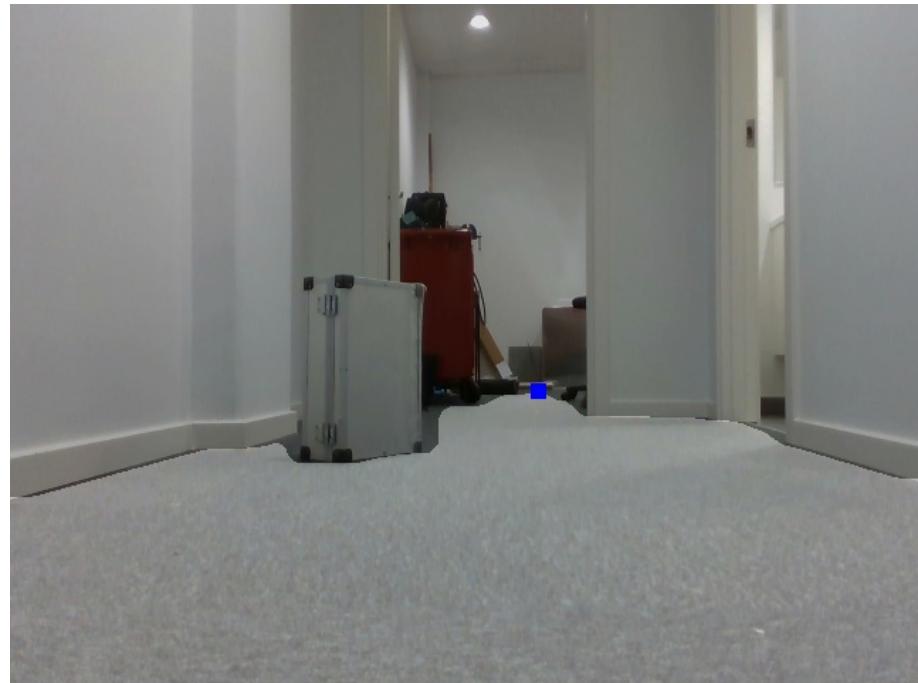
Figure B.6: Billede 6



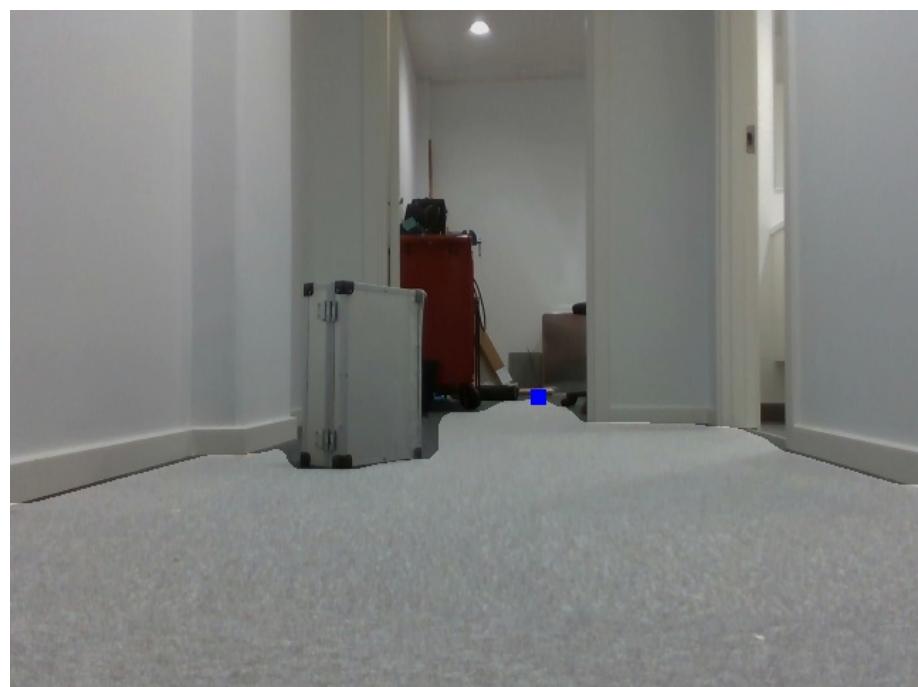
*Figure B.7: Billede 7*



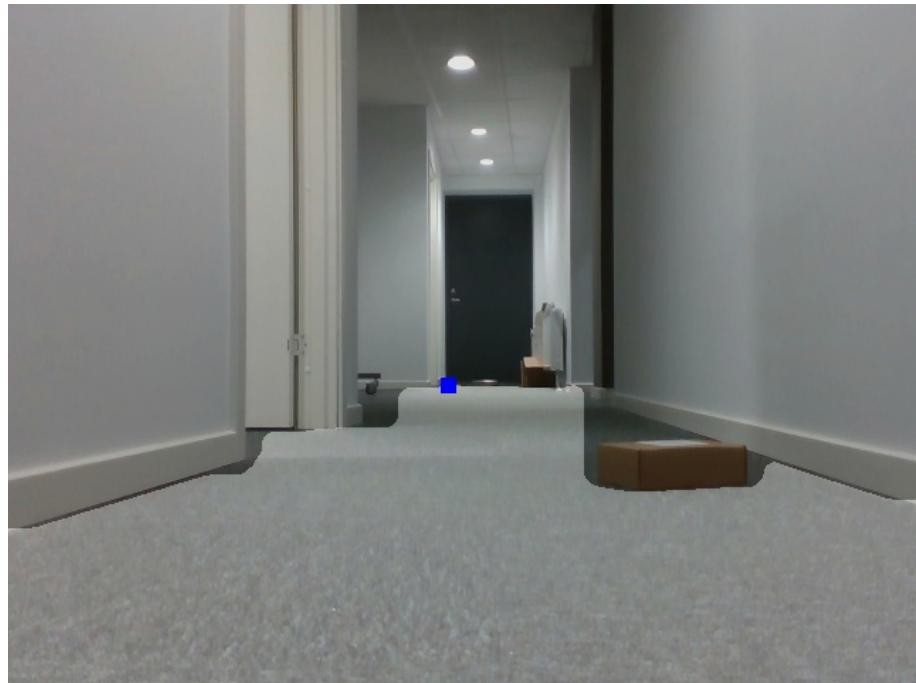
*Figure B.8: Billede 8*



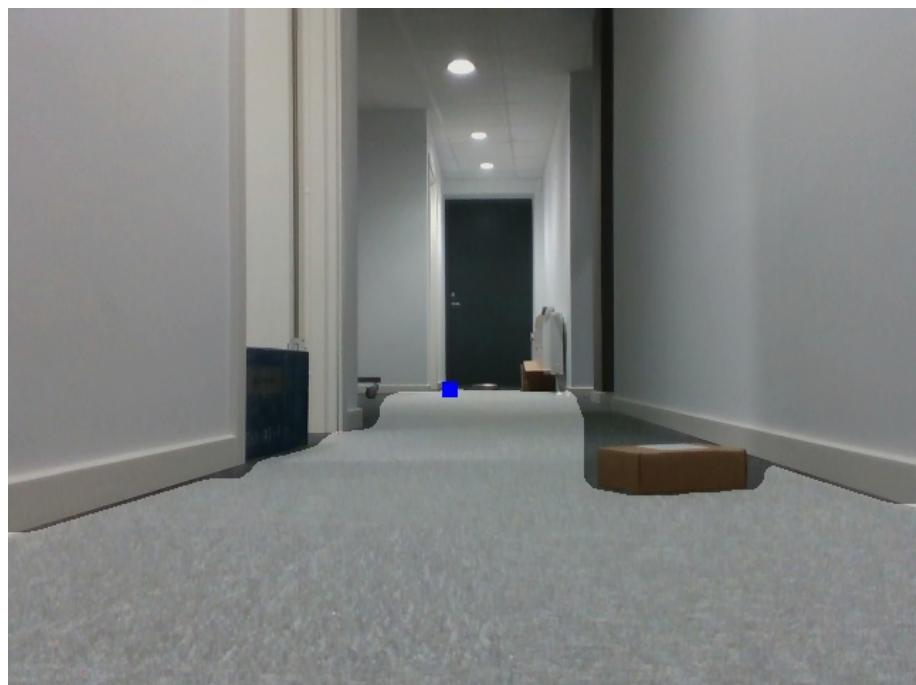
*Figure B.9: Billede 9*



*Figure B.10: Billede 10*



*Figure B.11: Billede 11*



*Figure B.12: Billede 12*



Figure B.13: Billede 13

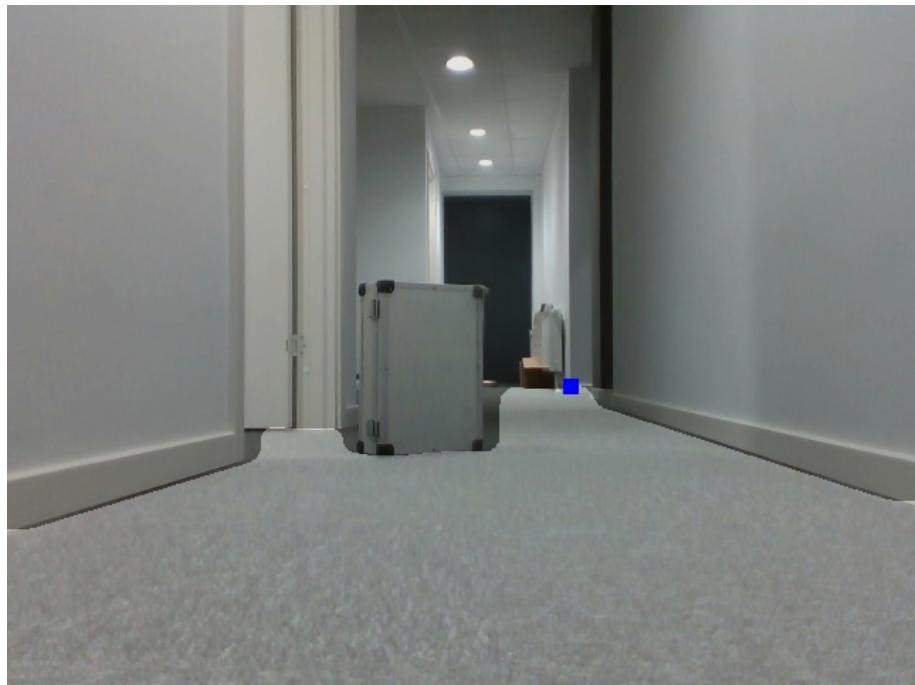
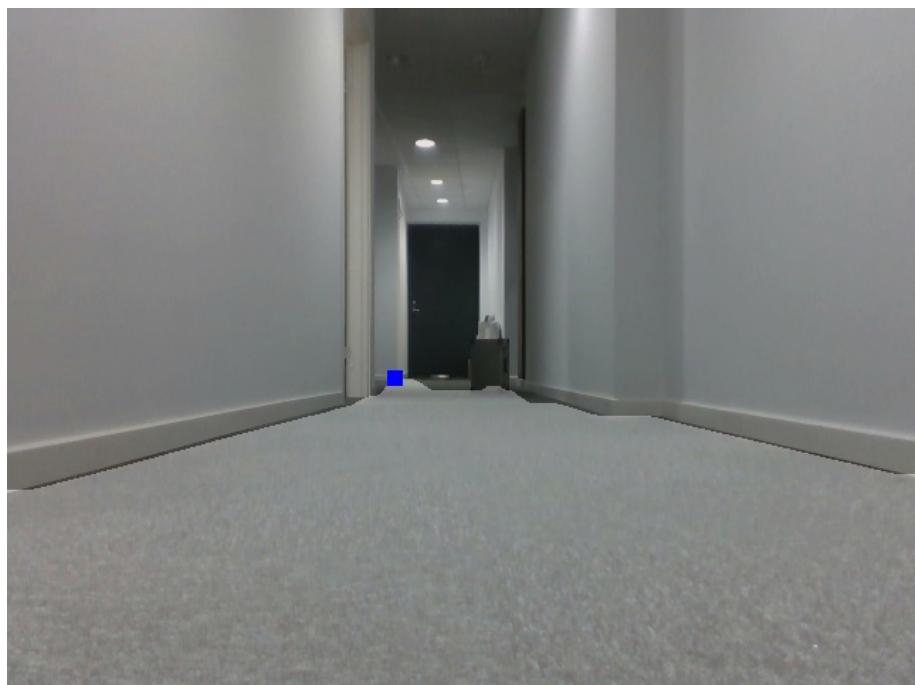


Figure B.14: Billede 14



*Figure B.15: Billede 15*



*Figure B.16: Billede 16*



*Figure B.17: Billede 17*

## C Resultater fra Objektdetekteringstests

Resultater

Epoker: 20

Læringskurve: 0.010

Scores:

Præcisionsscore 94.9%, test 96.72%

Præcisionsscore 73.5%, test 72.13%

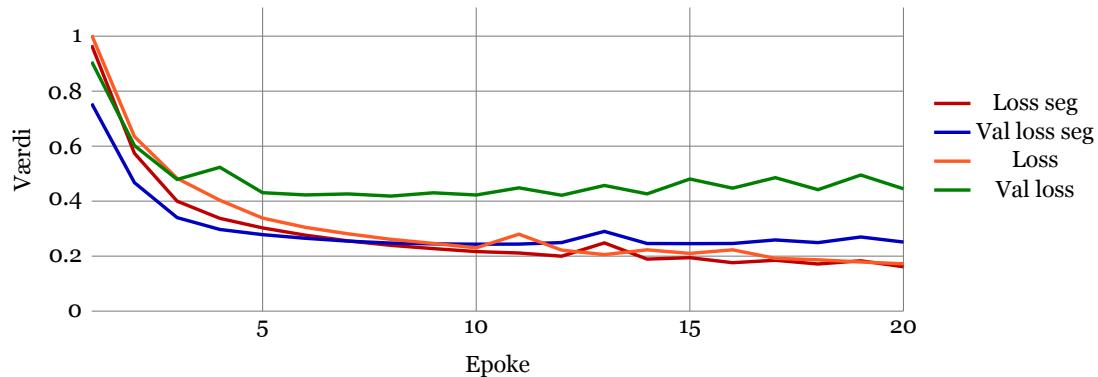


Figure C.1: Edgeimpulse

Epoker: 25

Læringskurve: 0.015

Scores:

Præcisionsscore 93.8%, test 93.44%

Præcisionsscore 66.2%, test 37.70%

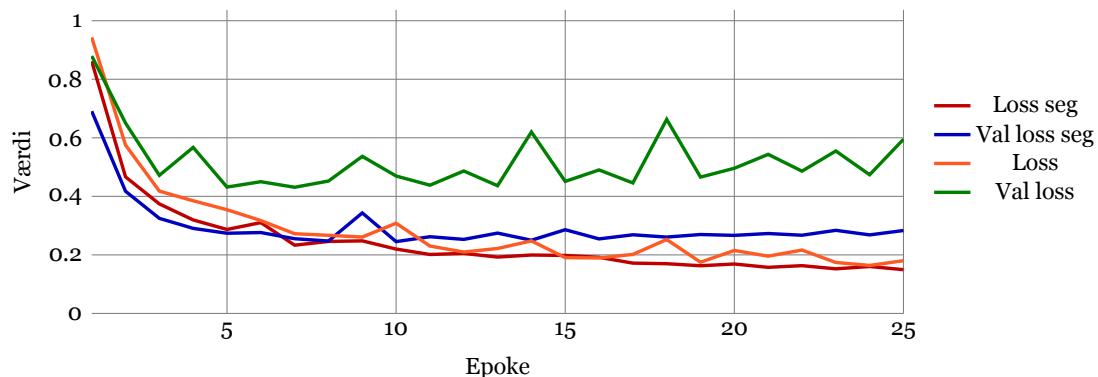


Figure C.2: Edgeimpulse

Epoker: 40

Læringskurve: 0.005

Scores:

Præcisionsscore 95.3%, test 90.16%

Præcisionsscore 75.9%, test 70.49%

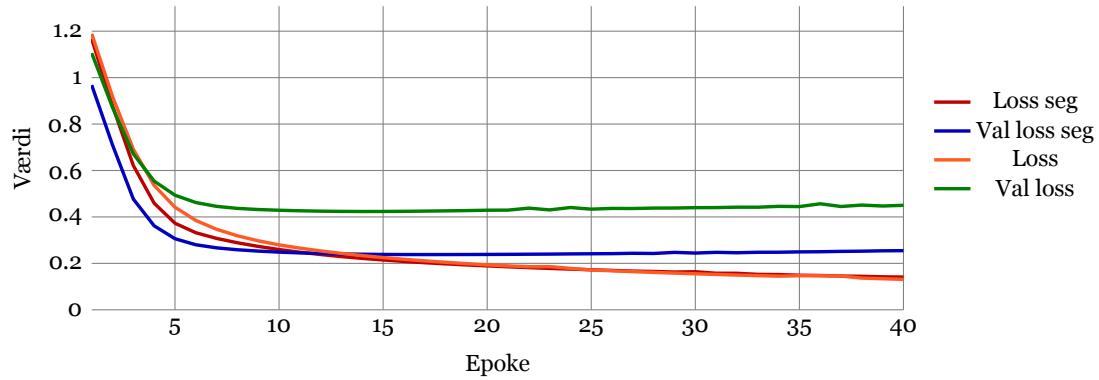


Figure C.3: Edgeimpulse

Dette projekt er udført i samarbejde med Bionic System Solutions.

Bionic System Solutions  
Østre Stationsvej 41P  
5000 Odense C  
<https://bionicsystemsolutions.com>



Syddansk Universitet  
Campusvej 55  
5230 Odense M  
[www.sdu.dk/mmmi](http://www.sdu.dk/mmmi)

