

# Semantische positietracking van een mobiele robot in zie- kenhuisgangen d.m.v. visie

**Olivier VAN DEN EDE**

Promotor(en): Prof. dr. ir. Toon Goedemé

Co-promotor(en): Filip Reniers

Masterproef ingediend tot het behalen van  
de graad van master of Science in de  
industriële wetenschappen: Electronica-ICT  
afstudeerrichting ICT

Academiejaar 2018 - 2019

©Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, kan u zich richten tot KU Leuven Technologiecampus De Nayer, Jan De Nayerlaan 5, B-2860 Sint-Katelijne-Waver, +32 15 31 69 44 of via e-mail [iiw.denayer@kuleuven.be](mailto:iiw.denayer@kuleuven.be).

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

# **Voorwoord**

Ik zou graag mijn promotoren Toon Goedemé en Filip Reniers bedanken voor het begeleiden van mijn thesis. Zij waren steeds bereikbaar voor vragen en feedback over de implementatie.

Ik zou ook graag de EAVISE en de 'Robotics Research Group' bedanken voor het mogelijk maken van deze thesis.

Als laatste wil ik mijn familie, vrienden en klasgenoten bedanken voor de steun tijdens mijn opleiding.

# **Abstract**

In deze thesis gaan we proberen om de positie van een robot te tracken door gebruik te maken van een RGB camera en een semantische kaart in de logistieke gangen van een ziekenhuis.

In deze thesis is er een volledige pipeline opgesteld die op basis van een input stream van RGB beelden opvallende kenmerken gaat zoeken in de ruimte zoals lampen, rookmelders en brandblusser. In combinatie met perspectiefpunctdetectie en informatie over deze gedetecteerde objecten op de semantische kaart zal er een schatting gemaakt worden van de actuele lokatie van de robot. We hebben onderzocht welke object detector en welke perspectiefpunt detectiemethode de beste resultaten oplevert, en op basis van deze gegevens de volledige pipeline geïmplementeerd in python.

# **Abstract**

In this thesis we try to track the position of a mobile robot inside the logistic hallways of a hospital by using a RGB camera and a semantic map.

We created a whole pipeline to detect notable features in the corridors such as lamps, smoke detectors and fire extinguishers bases on a stream of RGB images. With a combination of this object detector, a vanishing point detector and information from the semantic map about these objects we try to estimate the current location of the robot.

We researched which object detector and vanishing point detection method delivered the best results, and implemented the processing pipeline in python based on this information.

# Inhoudsopgave

<b>Voorwoord</b>	iii
<b>Abstract</b>	v
<b>Inhoud</b>	vii
<b>Figurenlijst</b>	viii
<b>Tabellenlijst</b>	ix
<b>Acroniemen</b>	x
<b>1 Probleemstelling</b>	1
<b>2 Literatuurstudie</b>	2
2.1 Indoor navigatie & visie . . . . .	2
2.2 Object detectie . . . . .	2
2.2.1 Traditionele beeldverwerking . . . . .	3
2.2.2 Convolutional neural network . . . . .	4
2.3 Object tracking . . . . .	5
2.4 Image segmentation . . . . .	6
<b>3 Uitwerking</b>	8
3.1 Object detectie . . . . .	9
3.1.1 Annotatie beeldmateriaal . . . . .	9
3.1.2 Training YOLOv2 . . . . .	10
3.2 Perspectiefpunt detectie . . . . .	11
3.2.1 Hoogste vloerpixel segmentatie . . . . .	11
3.2.2 Vloerlijn kruising . . . . .	11
3.2.3 Perspectieflijn kruising . . . . .	12

3.3 Omgevingsmeting . . . . .	13
3.3.1 Camera kalibratie . . . . .	14
3.4 Semantische kaart . . . . .	15
3.5 Lokalisatie . . . . .	17
<b>4 Resultaten</b>	<b>20</b>
4.1 Object detectie . . . . .	20
4.1.1 Training . . . . .	20
4.1.2 Snelheid . . . . .	20
4.1.3 Nauwkeurigheid . . . . .	22
4.2 Perspectiefpunt detectie . . . . .	23
4.2.1 Snelheid . . . . .	23
4.2.2 Nauwkeurigheid . . . . .	24
4.3 Lokalisatie . . . . .	25
4.3.1 Snelheid . . . . .	25
4.3.2 Nauwkeurigheid . . . . .	26
<b>5 Besluit</b>	<b>28</b>
<b>A CVAT naar YOLO conversie</b>	<b>31</b>

# Lijst van figuren

2.1	De lagen van een CNN volgens het YOLO [13] detection system.	4
2.2	Het SegNet [2] segmentatie netwerk.	7
3.1	Overzicht van het programma	8
3.2	ResNet segmentatie van 1 input frame.	11
3.3	Hoogste vloerpixel als perspectiefpunt	12
3.4	<b>Linksboven:</b> vloersegmentatie, <b>Rechtsboven:</b> Canny-edgedetectie, <b>Linksonder:</b> alle Hough lijnen, <b>Rechtsonder:</b> filtering van lijnen + resultaat	12
3.5	<b>Links:</b> Canny-edgedetectie op hele afbeelding. <b>Rechts:</b> Hough lijnen uit edge detection.	13
3.6	Schematische voorstelling omgevingsmeting	14
3.7	Detectie van het dambordpatroon voor camerakalibratie	15
3.8	Grafische voorstelling semantische kaart	16
3.9	Overzicht van de lokalisatie	18
4.1	Inferentie van YOLOv2 detector met 1 klasse op 100 afbeeldingen.	21
4.2	Inferentie van YOLOv2 detector op verschillende inputformaten. <b>Links:</b> 4 klassen, <b>Rechts:</b> 1 klasse	21
4.3	Grafische voorstelling IoU	22
4.4	Precision-Recall curves van YOLOv2 detector met mAP in de legende. <b>Links:</b> detector met 4 klassen. <b>Rechts:</b> detector met 1 klasse	23
4.5	Detectiesnelheid van 3 perspectiefpunt detectiemethoden op CPU en GPU	23
4.6	Afwijking t.o.v. perspectiefpunt voor 3 verschillende detectietechnieken.	24
4.7	Berekeningssnelheid van de actuele locatie	26
4.8	Gemiddelde profilering van het lokalisatieproces	26
4.9	Vergelijking van lokalisatie fouten	27

# **Lijst van tabellen**

3.1 Annotaties in training en validatie set . . . . .	9
3.2 YOLOv2 training hyperparameters . . . . .	10
4.1 Computer specificaties . . . . .	20

# Acroniemen

**AGV** Autonomoos Geleid Voertuig. 1

**CNN** Convolutional Neural Network. 4, 5, 6, 7, 9

**CPU** Central Processing Unit. 21, 23, 24

**CVAT** Computer Vision Annotation Tool. 9

**EM** Expectation-Maximization. 6

**GPU** Graphical Processing Unit. 21, 23, 24

**HOG** Histogram of Oriented Gradients. 3

**HSI** Hue Saturation Intensity. 3

**IoU** Intersect Over Union. 22

**LSTM RNN** Long Short Term Memory Recurrent Neural Network. 6

**mAP** Mean Average Precision. 22

**OCR** Optical Character Recognition. 2, 3

**OSM** OpenStreetMap. 15, 17

**RANSAC** Random Sample Consensus. 3, 6

**ReLU** Rectified Linear Unit. 5

**RGB** Rood Groen Blauw. 1, 2, 4, 7, 8, 9, 13, 25

**ROI** Region Of Interest. 5, 6

**ROLO** Recurrent YOLO. 6

**SIFT** Scale-Invariant Feature Transform. 3, 5

**SVM** Support Vector Machine. 3, 6

**YOLO** You Only Look Once. 5, 6, 10

# **Hoofdstuk 1**

## **Probleemstelling**

Ziekenhuizen kampen al langer met personeeltekorten en een hoge werkdruck voor het zorgpersoneel. Een deel van dit probleem komt doordat ze ook instaan voor de textiellogistiek en de goederenstroom. Dit probleem zou aangepakt kunnen worden met behulp van automatisatie van de transporten van textiel, karren en bedden. Deze automatisatie staat momenteel nog niet zo ver, omdat vergeleken met de industrie het moeilijk is om de volledige infrastructuur aan te passen, en deze aanpassingen meestal niet overweg kunnen met het bestaande logistiek materiaal. Een Autonomo Geleid Voertuig (AGV) zou gebruikt kunnen worden om het transport van karren en bedden te automatiseren binnen de logistieke gangen van het ziekenhuis.

Dit voertuig moet vanzelf kunnen navigeren in de gangen van een ziekenhuis en weten waar het zich op elk moment bevindt. Om dit te realiseren wordt het voertuig uitgerust met een aantal sensoren en een Rood Groen Blauw (RGB) camera om de omgeving te observeren. Voor navigatie beschikt het AGV over een semantische kaart. Dit is een kaart waarop aangeduid staat wat voor objecten er te zien zijn (muren, deuren, bordjes, verlichting, ..) samen met de afmetingen, positie en oriëntatie van deze tags.

Het doel van deze masterproef is het onderzoeken welke objecten/features er aanwezig zijn in de logistieke gangen van een ziekenhuis en op basis daarvan beeldverwerkingstechnieken te zoeken die geschikt kunnen zijn voor detectie en tracking van deze objecten. Deze detecties kunnen dan gebruikt worden om de locatie van de robot te bepalen op basis van de kaart. Vervolgens is het de bedoeling dat de robot vertrekt vanop een gekende locatie en d.m.v. zijn kaart en de objecten die hij detecteert in zijn omgeving autonoom kan navigeren naar een eindpunt.

# **Hoofdstuk 2**

## **Literatuurstudie**

### **2.1 Indoor navigatie & visie**

Op visie gebaseerde navigatie is een onderwerp dat zeer vaak onderzocht wordt. Oudere onderzoeken zoals [20] maken gebruik van een robot met een RGB camera die zonder kaart informatie navigeert. De enige informatie die gegeven wordt is een eenvoudige object beschrijving van de gang en een beschrijving van een deur met een deurnummer ernaast. Met enkel een deurnummer als doel vertrekt de robot door de gang, en houdt zichzelf parallel met de muren door gebruik te maken van andere sensoren. Eens er een deur in beeld komt, worden er een aantal features(randen) herkent in het beeld. Nadat hun algoritme de deuren herkend kan er via Optical Character Recognition (OCR) op het deurnummer worden nagegaan of het doel bereikt is. Dit is uiteraard een zeer eenvoudige techniek omdat de robot geen begrip heeft van de omgeving, en moeilijk plaatsen t.o.v elkaar kan onderscheiden.

Nieuwere technieken zoals [8] maken gebruik van RGB-D camera's zoals bijvoorbeeld een kinect waardoor ze ook over diepte-informatie beschikken. Die diepte info kan dan gebruikt worden om heel de omgeving in 3d te mappen en op basis van de effectief gemeten positie te navigeren. Een andere manier om een 3d representatie van de omgeving te verkrijgen zoals [16] is gebruik te maken van stereovisie. Hierbij wordt de informatie van 2 RGB camera's die op een vaste afstand van elkaar staan gecombineerd om diepte informatie te verzamelen.

In dit onderzoek gaan we ons echter beperken tot één enkele RGB camera.

### **2.2 Object detectie**

Een belangrijk aspect van dit onderzoek is het detecteren van individuele objecten in het beeld van één enkele RGB camera. De te detecteren objecten zijn op voorhand vastgelegd, en zijn afhankelijk van de ruimte waarin de robot zich bevindt.

In de logistieke gangen van een ziekenhuis zijn er heel wat objecten die we kunnen detecteren, een kleine selectie van deze objecten zijn:

- Pictogrammen;
- Brandblussers;
- Deurklinken.

Voor deze objecten gaan we kijken naar detectietechnieken uit de traditionele beeldverwerking, en naar meer *state-of-the-art* technieken.

### 2.2.1 Traditionele beeldverwerking

In openbare gebouwen zijn er heel wat pictogrammen te vinden zoals bijvoorbeeld nooduitgang, hoogspanning en brandblusser. Deze pictogrammen hebben steeds een specifieke vorm, kleur en symbool. De literatuur over pictogramdetectie is schaars. De techniek die voorgesteld wordt door [18] gebruikt een zeer eenvoudige edge detectie met OCR om eventuele letters op pictogrammen te lezen, dit is echter minder relevant omdat niet op elk pictogram tekst aanwezig is. Pictogrammen kunnen echter wel vergeleken worden met verkeersborden die bijna dezelfde kenmerken hebben. De aanpak van [7] is om 2 soorten features in een beeld te onderscheiden. Enerzijds detecteren ze vormen op basis van kleurranden en anderzijds wordt de afbeelding omgezet naar Hue Saturation Intensity (HSI) waaruit enkel de hue gebruikt wordt. De hue is de belangrijkste component voor het onderscheiden van kleuren omdat er zo geen rekening wordt gehouden met de hoeveelheid licht en schaduwen. Een recent onderzoek [21] bouwt voort op deze technieken, maar hier berekenen ze de Histogram of Oriented Gradients (HOG) features van het beeld. Vervolgens wordt er gebruik gemaakt van een Support Vector Machine (SVM) om te bepalen waar er zich een match bevindt.

Een SVM is een binaire classificatie techniek die gebruikt kan worden om na te kijken bij welk klasse een feature het dichtst aanleunt. De scheiding tussen de 2 klassen wordt voorgesteld door een hypervlak dat bepaald wordt door middel van voorbeelden in de trainingsfase. Een feature wordt geklassificeerd door te kijken aan welke kant van het hypervlak hij het beste past.

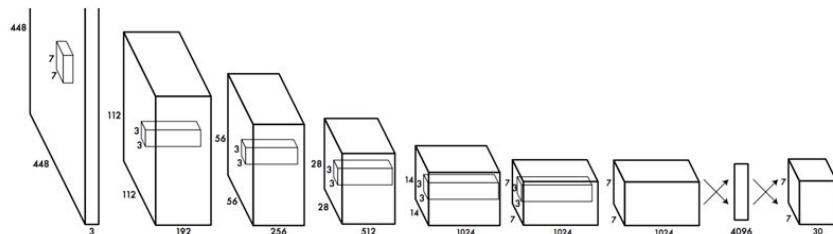
De vorm en kleur features kunnen dan gecombineerd worden om de positie van een mogelijke match te vinden. Eens er een mogelijke boundig box rond de mogelijke match gevonden is, kan er geprobeerd worden een template te matchen om het effectieve pictogram te achterhalen. Het grootste probleem bij de techniek van [7] is dat hun gebruikte template matching techniek niet robuust is tegen schaal invariantie. Bij [21] maken ze voor de herkenningsfase gebruik van Scale-Invariant Feature Transform (SIFT)[11] features en kleur informatie. Hierdoor is het probleem van schaal invariantie grotendeels opgelost. Hierbij worden de SIFT features van de kandidaat matches en de templates vergeleken, en wordt er een gemiddelde genomen van de verschillen tussen hue, saturation en value. Door middel van Random Sample Consensus (RANSAC) en een threshold wordt er bepaald welke matches gebruikt worden. Deze techniek zou gebruikt kunnen worden voor het detecteren van pictogrammen.

## 2.2.2 Convolutional neural network

De laatste jaren in het domein van beeldverwerking wordt er steeds meer gegrepen naar deep learning technieken. Dit komt omdat de rekenkracht van computers steeds beter en beter wordt, en de resultaten die bekomen worden met neurale netwerken de traditionele manieren overtreffen op verschillende vlakken. Een deep learning techniek die veel gebruikt wordt in de beeldverwerking is een Convolutional Neural Network (CNN).

Een CNN is een supervised deep learning techniek die gebruikt kan worden om verschillende beeldverwerkende taken uit te voeren. Als een traditioneel neuraal netwerk gebruikt zou worden voor een afbeelding van 448 bij 448 pixels, zou er bij elke inwendige laag een paar miljoen variabele gewichten aanwezig moeten zijn, dit zou leiden tot een veel te complex netwerk. Daarom wordt er gebruikgemaakt van een CNN, dit is eigenlijk een neuraal netwerk waarbij niet alle neuronen met elkaar verbonden zijn.

Een CNN kan bestaan uit meerdere lagen die meestal een combinatie zijn van 'convolutional-layers' en 'fully connected-layers'. Elk van deze lagen bevat een aantal neuronen met elk een eigen set van gewichten. Het doel van een CNN is om de gewichten zodanig bij te stellen dat data die aan de eerste laag (de input laag) gegeven wordt een verwacht resultaat geeft aan de laatste laag (de output/classificatie laag). Deze laatste laag kan men de classificatielaag noemen, en geeft een representatie van wat het netwerk denkt dat er aan de input staat. In figuur 2.1 is een voorbeeld zichtbaar van een CNN met de verschillende soorten lagen.



**Figuur 2.1:** De lagen van een CNN volgens het YOLO [13] detection system.

Een 'convolutional-layer' of convolutie-laag is een laag die een convolutie operatie uitvoert op zijn input. Dit wil zeggen dat de input verdeeld word in regio's van bijvoorbeeld 7x7 pixels, deze 49 pixels zijn verbonden met 1 neuron in de volgende inwendige laag. Als dit 7x7 masker wordt opgeschoven met 1 pixel, verkrijgen we de input voor aan andere neuron in de volgende laag. Dit proces kan gebeuren in meerdere dimensies tegelijkertijd, op dat moment spreekt men van een tensormasker. Een voorbeeld van een convolutie in meerdere dimensies tegelijkertijd is een RGB afbeelding die in een keer door het netwerk gaat, waarbij de dimensies de rode, groene en blauwe pixels van het beeld zijn.

De verschillende inwendige lagen van een CNN zullen na training op zoek gaan naar features. Deze features kunnen eenvoudig zijn zoals randen en lijnen, maar kunnen ook complexer zijn specifiek voor de getrainde data. Elke laag genereert dus een soort featuremap, die gebruikt wordt als input voor de volgende laag van het netwerk.

Om uiteindelijk een classificatie te verkrijgen moet er een dimensievermindering doorgevoerd worden, dit wordt gedaan door 'pooling layers' aan het netwerk toe te voegen na elke convolutie laag. Dit heeft ook als effect dat de featuremaps vereenvoudigd worden, en het aantal gewichten beperkt blijft.

Een CNN kan pas gebruikt worden nadat het getraind is. Voor de training van een netwerk zijn er 2 dingen noodzakelijk: veel voorbeeld data en per voorbeeld de verwachte output (label). Bij het trainingsproces wordt alle inputdata aangelegd, en wordt er gekeken wat het netwerk aan zijn output heeft. De loss functie is een maat van hoe goed een netwerk een voorspelling kan doen van de input data, met andere woorden een vergelijking tussen de input en de output. Trainen van een netwerk is het optimaliseren van de gewichten bij de neuronen in elke laag van het netwerk, bij een optimaal resultaat is de loss functie minimaal. Dit is een complex probleem dat enkel kan lukken indien er genoeg trainingsdata ter beschikking is. Trainen kan gedaan worden d.m.v 'backpropagation'. Backpropagation is het steeds een klein beetje aanpassen van de gewichten bij de neuronen in het netwerk om de classificatie ten gevolge van de input, en het trainingslabel beter op elkaar af te stemmen. Dit proces werkt door de kettingregel toe te passen, en voor elke laag een voor een de gewichten te zoeken.

Een belangrijke laag in een CNN is de Rectified Linear Unit (ReLU). Deze laag voegt een niet lineaire functie toe tussen de convolutielagen, dit is nodig omdat de convolutie een lineaire operatie is, en een eigenschap van lineaire operators laat toe om deze te combineren tot 1 operator. Dit zou zeggen dat alle convoluties in een netwerk samengesteld kunnen worden tot 1 laag, en dus het voordeel van meerdere lagen wegnemen. Het toevoegen van een ReLU laag tussen elke convolutielag voorkomt dit probleem.

Een voorbeeld van een CNN is het 'You Only Look Once (YOLO) detection system' [13]. Het YOLO netwerk is opgebouwd uit 24 convolutielagen en 2 'fully connected' lagen. Dit netwerk heeft een uitgebreide training gehad op de ImageNet dataset<sup>1</sup> en kan gebruikt worden om objectdetectie en -classificatie te doen door één keer de input afbeelding door het netwerk te laten gaan. Door middel van een hertraining kan deze detector leren om alle objecten te detecteren en te classificeren en dus een mogelijke detector zijn voor onze toepassing.

Zoals [10] voorstelt is het niet moeilijk om het 'YOLO detection system' een hertraining te geven om deuren te herkennen. Zo kan dit ook toegevoegd worden aan de lijst met te detecteren kenmerken.

## 2.3 Object tracking

Object tracking of het volgen van objecten heeft als doel het bepalen van de positie van hetzelfde object over meerdere frames heen. In het geval van dit onderzoek kan het een indicatie geven van relatieve posities t.o.v. objecten die zich in de gangen bevinden. Een grote moeilijkheid bij het volgen van objecten die stilstaan t.o.v. de camera is dat ze veranderen in grootte, oriëntatie en perspectief. [23] stelt voor om gebruik te maken van SIFT voor het volgen van objecten. Ze zoeken een Region Of Interest (ROI) op het eerste frame waarop ze een kleurhistogram en SIFT

<sup>1</sup><http://www.image-net.org>

features berekenen. Op het volgende frame worden dezelfde bewerkingen uitgevoerd in een regio die net iets groter is dan de originele ROI. Een overeenkomstregio wordt dan berekend door middel van een Expectation-Maximization (EM) algoritme. Volgens [3] is het beter om gebruik te maken van het KLT feature algoritme [19]. Hiermee wordt er een transformatie berekent waardoor de ROI tussen de 2 frames gelijkaardig wordt. De initiële transformatieparameters worden berekend via RANSAC.

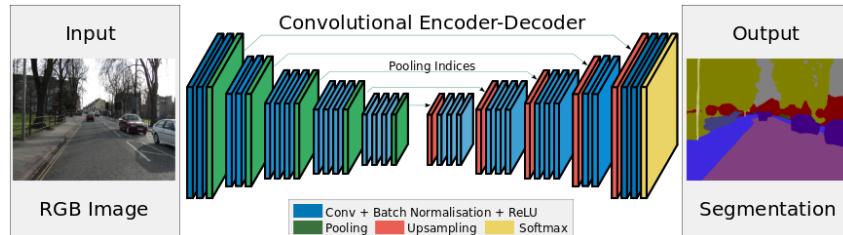
[12] heeft een nieuwe techniek ontwikkeld om object tracking te combineren met object detectie CNN. Ze hebben een uitbreiding gemaakt op het YOLO detection system genaamd Recurrent YOLO (ROLO). De uitbreiding bevat een extra Long Short Term Memory Recurrent Neural Network (LSTM RNN) geplaatst achter de detectie fase van het originele netwerk. Een LSTM RNN is een neuraal netwerk geoptimaliseerd voor het maken van beslissingen op tijdsgebaseerde data. De data die aan het extra netwerk gegeven wordt is een van de tussenresultaten van het YOLO netwerk. Het systeem blijkt zeer goed te werken voor het volgen van objecten zelfs bij occlusies in één van de beelden.

## 2.4 Image segmentation

Het correct segmenteren van de beelden zal een belangrijke rol spelen. Niet in elk beeld zal er een distinctief object aanwezig zijn om te detecteren. Daarom is het belangrijk om de vloer van de muren te kunnen onderscheiden. Een eenvoudige aanpak zou kunnen zijn om via K-means een verdeling van een beeld te doen en met een soort regressie de regio's te labelen. Volgens [22] werkt de K-means aanpak met een op textuur en kleur gebaseerde aanpak redelijk goed, maar wordt steeds de muur verbonden met het plafond omdat van kleur en textuurgelijkenissen. Hun regressie gebaseerde labeling techniek blijkt dus een slechte oplossing. Verder zoals [9] aangeeft zijn reflecties en overbelichting eigenschappen van indoor omgevingen die het moeilijk kunnen maken om een correcte segmentatie te doen. [9] stelt een techniek voor die begint met het detecteren van verticale en horizontale lijn segmenten. Dit doen ze door eerst een Canny edge detector[4] toe te passen en vervolgens een line fitting. Een zelf geleerde SVM classifier verdeelt alle lijnsegmenten in twee categorieën: horizontaal en verticaal. De vluchtlijnen van de gang worden hierbij onderverdeeld in de horizontale categorie. Alle lijnstukken krijgen een score via een reeks van operaties waarna enkel de beste lijnen bijgehouden worden. Op basis van de kleur van de vlakken tussen de lijnstukken kan een segmentatie gemaakt worden. Dit geeft een resultaat waarbij de vloer meestal een mooi homogeen geheel is, maar de muren in meerdere vlakken gesegmenteerd worden door eventuele kleurverschillen en objecten aan de muur.

Een andere manier om de vloer te segmenteren is voorgesteld in [15]. Zij doen een superpixel segmentatie volgens het SLIC algoritme [1], vervolgens bekijken ze de randen van de superpixels. Na observaties blijkt dat de randen van superpixels onregelmatig worden bij objectovergangen. Door het aanduiden van een paar vloerpixels kan hun algoritme superpixels aanduiden die tot de vloer behoren. Deze aanpak geeft een goede schatting van vrije ruimte op de vloer, maar is minder bruikbaar voor segmentatie van muren.

Een meer recente technologie om afbeeldingen te segmenteren is gebruik te maken van een CNN. Het netwerk voor segmentatie is verschillend van een traditioneel CNN voor bijvoorbeeld object detectie. Een voorbeeld van een segmentatiennetwerk is te zien in figuur 2.2.



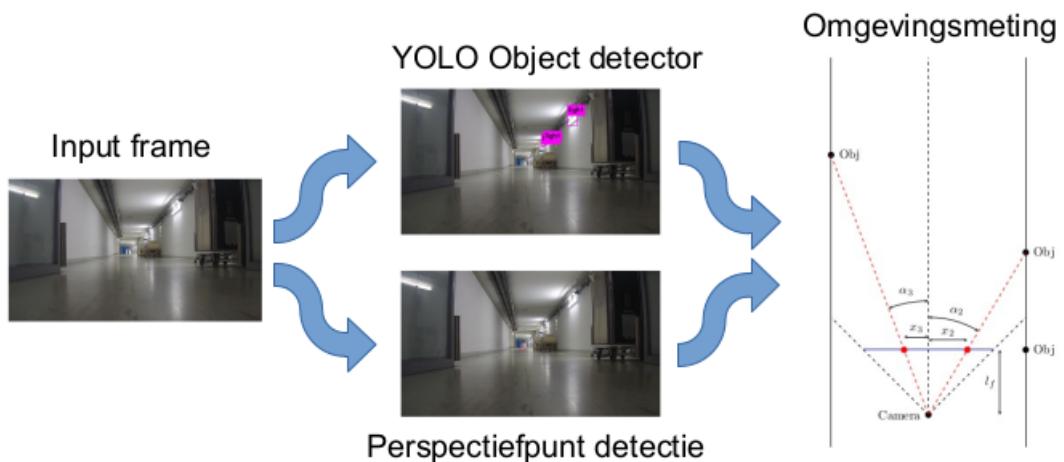
**Figuur 2.2:** Het SegNet [2] segmentatie netwerk.

Het segmentatiennetwerk SegNet [2] is een combinatie van convolutielagen en pooling layers. Er zijn geen fully connected layers aanwezig zoals het geval is bij een classificatie netwerk. De bedoeling van het SegNet netwerk is om als output opnieuw een afbeelding te genereren. Hiervoor zijn de lagen opgebouwd als een zandloper. Op deze manier is de output even groot als de oorspronkelijke afbeelding. Een segmentatiennetwerk wordt getraind op gelijkaardige manier aan een traditioneel CNN met als verschil dat de labeling gebeurd op pixelbasis aangezien de output even groot is als de input van het systeem. Het systeem geeft als output een label voor elke pixel uit de afbeelding onderverdeeld in de klassen waarmee het systeem getraind is.

Het SegNet netwerk is getraind op de SUN RGB-D [17] dataset. Deze dataset bevat een groot aantal indoor scenes, waarbij er onder andere segmentatie klassen zijn voor muren, vloeren en plafonds. De training is gebeurd met enkel de RGB gegevens van de dataset. Deze segmentatietechniek zou nuttig kunnen zijn voor dit onderzoek om in beelden bijvoorbeeld de vloer te kunnen onderscheiden.

# Hoofdstuk 3

## Uitwerking



Figuur 3.1: Overzicht van het programma

Het doel van deze masterproef is om op basis van 1 RGB camera en een semantische kaart een robot autonoom te laten navigeren in de logistieke gangen van een ziekenhuis. Om dit te implementeren, is er een pipeline opgesteld. Deze pipeline is onderverdeeld in een aantal stappen die in de volgende hoofdstukken uitgebreid besproken worden:

- Object detectie (3.1);
- Vluchtpunt detectie (3.2);
- Omgevingsmeting (3.3);
- Semantische kaart (3.4);
- Lokalisatie (3.5).

### 3.1 Object detectie

Zoals besproken zal worden in hoofdstuk 3.4, bevat de semantische kaart van het ziekenhuis een beschrijving van objecten/features die aanwezig zijn in de gangen. Deze objecten zijn bijvoorbeeld:

- Lampen;
- Brandblussers;
- Pictogrammen;
- Rookdetectors;
- Deurklinken.

De gebruikte dataset zijn 2 video's opgenomen vanop een robot met de waylens horizon<sup>1</sup> RGB camera. In dit beeldmateriaal zijn er een aantal van bovenstaande objecten zichtbaar, en kan er een detector gemaakt worden om deze features te herkennen.

De gekozen object detector is YOLOv2[14]. Dit is een zeer veel gebruikt object detectie/classificatie convolutioneel neuraal netwerk dat toelaat om objecten van een geleerde klasse te herkennen en lokaliseren in nieuwe beelden. De belangrijkste reden dat we deze detector gekozen hebben, is omdat op een grafische kaart deze real-time kan werken.

#### 3.1.1 Annotatie beeldmateriaal

Om een object-detector te maken gebaseerd op een CNN moet al het beeldmateriaal geannoteerd worden door bounding-boxes te tekenen rondom de kenmerkende objecten. Elk van deze bounding-boxes moet ook voorzien worden van een bijhorend label. Er is gekozen om de dataset op te splitsen in 2 delen, een training en een validatie set. De 2 datasets omvatten verschillende delen van de gangen in het gebouw. Het splitsen is nodig om door middel van de validatie set te valideren hoe goed de detector presteert op niet eerder gezien beelden. In tabel 3.1 is een opsomming gemaakt van alle annotaties in de training en validatie set.

**Tabel 3.1** Annotaties in training en validatie set

	Aantal frames	Rookmelder	Deurklink	Pictogram	TL-lamp	Totaal
Trainingsset	899	1016	147	340	5260	6763
Validatieset	711	1130	180	408	992	2710

Voor het aanduiden van de annotaties maakten we gebruik van de OpenCV Computer Vision Annotation Tool (CVAT)<sup>2</sup>. De output van de CVAT tool is een XML-bestand met voor elke afbeelding de coördinaten van de objecten. Een voorbeeld output voor 1 enkele afbeelding is te zien in listing 3.1.

<sup>1</sup><https://waylens.com/horizon/techSpecs/>

<sup>2</sup><https://github.com/opencv/cvat>

**Listing 3.1:** Voorbeeld CVAT output

```
<image id="24" name="hospital_corridors_025.png" width="1280" height="720">
    <box label="door_handle" xtl="863" ytl="237" xbr="881" ybr="248"/>
    <box label="light" xtl="584" ytl="254" xbr="608" ybr="265"/>
    <box label="light" xtl="435" ytl="321" xbr="448" ybr="329"/>
    <box label="exit_sign" xtl="590" ytl="298" xbr="604" ybr="307"/>
</image>
```

De trainingsdata die aan het YOLO netwerk geleverd moet worden is in een volledig ander formaat dan het verkregen XML bestand, daarom hebben we een conversiescript geschreven om dit om te zetten naar het YOLO formaat. Het conversiescript bevindt zich in bijlage A. Het formaat dat YOLO verwacht is per input afbeelding een .txt bestand met daarin per lijn een bounding box. Het YOLO formaat is beschreven in 3.1 waarbij  $\langle x \rangle$  en  $\langle y \rangle$  het centerpunt van een bounding box zijn. Alle waarden zijn genormaliseerd, dit wil zeggen dat om het absolute  $(x, y)$  coördinaat te bekomen, de waardes vermenigvuldigd moeten worden met de breedte en de hoogte zoals geïllustreerd in 3.2.

$$\langle x \rangle \langle y \rangle \langle width \rangle \langle height \rangle \quad (3.1)$$

$$\begin{aligned} x_{abs} &= \langle x \rangle \cdot \text{image\_width} \\ y_{abs} &= \langle y \rangle \cdot \text{image\_height} \end{aligned} \quad (3.2)$$

### 3.1.2 Training YOLOv2

Voor training en later ook detectie met YOLOv2 is er gebruikt gemaakt van een aangepaste versie van de darknet<sup>3</sup> implementatie in C. We zijn vertrokken van het yolo-voc.2.0.cfg configuratiebestand waar we het aantal te detecteren klassen hebben aangepast naar 4 (light, smoke\_detector, exit\_sign en door\_handle), en het aantal filters naar 25. Dit nieuw configuratiebestand specificert de netwerklayout, deze layout is een YOLOv2 netwerk waarbij nu de laatste lagen aangepast zijn om slechts 4 objecten te classificeren. De belangrijkste parameters zijn weergegeven in tabel 3.2.

Het trainen gebeurt nu op basis van de trainingsset zoals weergegeven in tabel 3.1. De performance van het netwerk zal besproken worden in hoofdstuk 4.

**Tabel 3.2** YOLOv2 training hyperparameters

Naam	Batch size	Batch subdivisions	Learning rate	Maximum batches	Classes	Filters
Waarde	64	8	0.0001	45000	4	25

<sup>3</sup><https://github.com/AlexeyAB/darknet>



**Figuur 3.2:** ResNet segmentatie van 1 input frame.

## 3.2 Perspectiefpunt detectie

Ik hoofdstuk 3.3 zullen we bespreken hoe het lokaliseren werkt, om dit te realiseren wordt er gebruik gemaakt van het perspectiefpunt. Het detecteren van het vluchtpunt of perspectiefpunt kan gebeuren op verschillende manieren, voor dit onderzoek hebben we drie verschillende technieken uitgevonden en geïmplementeerd, en deze zullen verder besproken worden.

De performantie en nauwkeurigheid van de 3 methoden zal besproken worden in hoofdstuk 4.

### 3.2.1 Hoogste vloerpixel segmentatie

De eerste techniek maakt gebruik van het DeepLab-ResNet [5] segmentatiennetwerk, gelijkaardig aan de beschrijving in hoofdstuk 2.4. Het netwerk is geïmplementeerd in TensorFlow en voorge-traind op indoor scenes zodat het klaar is voor gebruik in python.<sup>4</sup>

Het netwerk kan gebruikt worden om in een afbeelding elke pixel onder te verdelen in een aantal klassen zoals: muur, vloer, boom, meubel en trap. Voor het detecteren van het perspectiefpunt hebben we gekozen om enkel de vloerpixels te gebruiken. Het resultaat van de segmentatie is te zien in figuur 3.2 waarbij groen vloerpixels zijn, roze muurpixels, grijs plafondpixels en blauw meubelpixels.

In eerste instantie nemen we van de gesegmenteerde data de hoogste vloerpixel, en deze gebruiken we als perspectiefpunt. Dit illustreren we in figuur 3.2

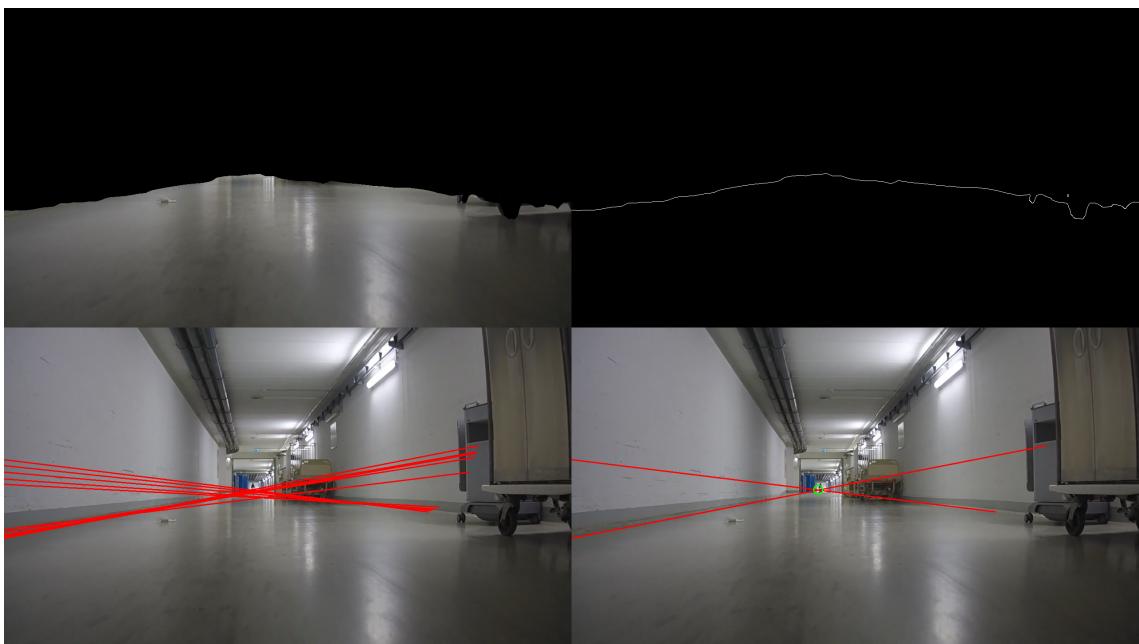
### 3.2.2 Vloerlijn kruising

Voor deze techniek maken we gebruik van dezelfde segmentatiedata als in hoofdstuk 3.2.1, maar wordt er een masker gemaakt met enkel de vloerpixels. Op dit masker wordt door middel van een Canny-edgedetector de rand van de vloer gedetecteerd, dit proces is gevisualiseerd in figuur 3.4. Vervolgens kunnen er door middel van de Hough-transformatie een aantal rechten gevonden worden die raken aan zoveel mogelijk punten met de rand van de vloer. Zoals te zien in figuur 3.4 worden er meerdere rechten gevonden, en deze moeten gefilterd worden omdat ze ongeveer dezelfde

<sup>4</sup><https://github.com/hellochick/Indoor-segmentation>



**Figuur 3.3:** Hoogste vloerpixel als perspectiefpunt

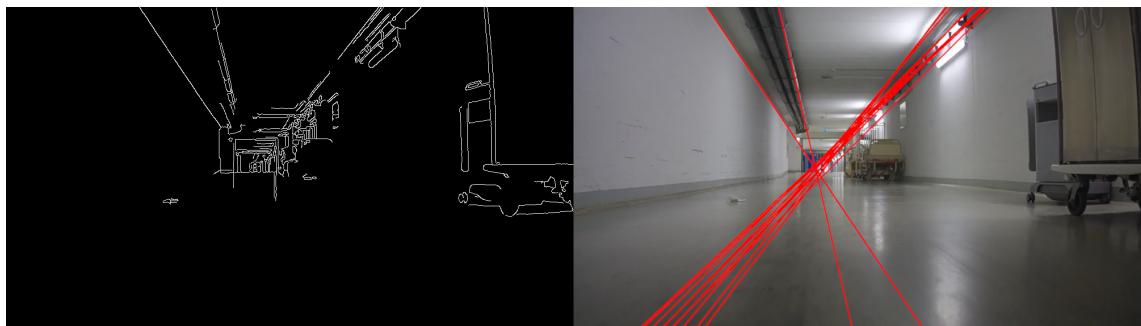


**Figuur 3.4:** **Linksboven:** vloersegmentatie, **Rechtsboven:** Canny-edgedetectie, **Linksonder:** alle Hough lijnen, **Rechtsonder:** filtering van lijnen + resultaat

lijn voorstellen. Dit gebeurt door de richtingscoëfficiënt van alle lijnen met elkaar te vergelijken, en indien het verschil te klein is, wordt de lijn met de laagste richtingscoëfficiënt verwijderd. Het optimale resultaat is dat er 1 lijn overblijft aan elke kant van de vloer, zodat deze de vluchtlijnen voorstellen. Indien dit het geval is, is de kruising van de 2 rechten het perspectiefpunt.

### 3.2.3 Perspectieflijn kruising

Deze techniek gelijkt sterk op de beschreven methode uit 3.2.2 met als verschil dat hier geen gebruik gemaakt wordt van vloersegmentatie. De gehele afbeelding wordt omgezet naar grijswaarden voordat het door een Canny-edgedetector gehaald wordt met een variabele threshold gebaseerd op de mediaan van de grijswaarden. De edge detectie wordt wederom gebruikt om door middel van de Hough-transformatie een aantal aanliggende rechten te zoeken die zoveel mogelijk punten snijdt.



**Figuur 3.5:** Links: Canny-edgedetectie op hele afbeelding. Rechts: Hough lijnen uit edge detection.

We zijn op zoek naar perspectieflijnen, dus alle horizontale en verticale lijnen worden onmiddellijk verwijderd net als rechten waarvan de richtingscoëfficiënt niet genoeg van elkaar afwijkt. Dit is weergegeven in figuur 3.5.

Indien er slechts een paar lijnen overschieten na filtering, kan gesteld worden dat het perspectiefpunt te vinden is op de kruising van de rechten. In het geval dat er nog meerdere kruispunten zouden overschieten, kan er een gemiddelde positie berekend worden van deze punten.

### 3.3 Omgevingsmeting

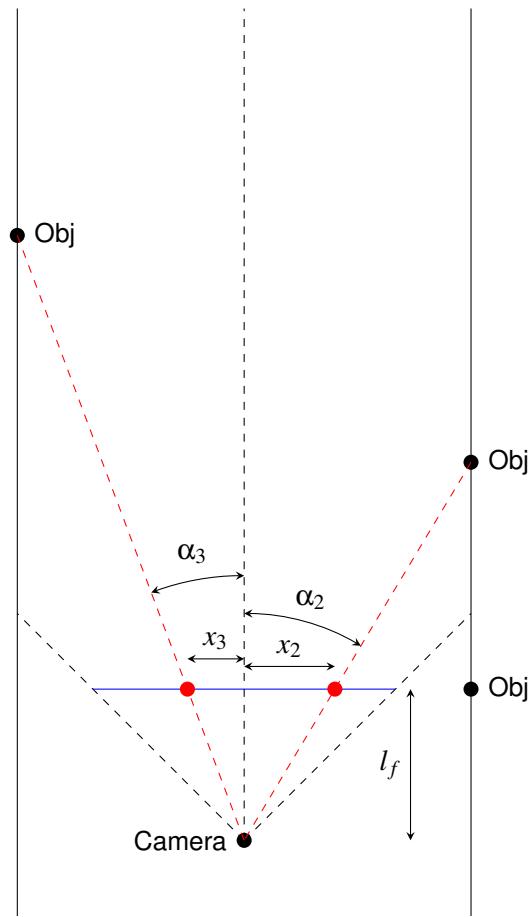
Door middel van de RGB camerabeelden genomen van het standpunt van de robot, willen we een representatie maken van de omgeving. Deze representatie zal later gebruikt worden om de actuele locatie van de robot te volgen. Voor het opbouwen van de omgeving maken we gebruik van de object detector beschreven in hoofdstuk 3.1. De detector geeft een lijst terug van objecten met voor elk object een begrenzende rechthoek en een klasse. Elk object heeft een unieke locatie in het beeld, en dus ook in de ruimte.

In figuur 3.6 is een schematische 2D voorstelling te zien van een gang waarin er zich 3 objecten bevinden, 2 van deze objecten bevinden zich binnen de kijkhoek van de camera, en worden door middel van de rode stippenlijnen geprojecteerd op het blauwe cameravlak.

De rode stippen zichtbaar in 3.6 zijn de coördinaten van de objecten zoals ze gedetecteerd worden door de object detector. Het doel is om op basis van deze coördinaten de hoek  $\alpha_i$  tussen de optische as van de camera en het object in de ruimte te bepalen.

Hiervoor komt het perspectiefpunt van pas, het perspectiefpunt ligt in de figuur op een oneindige afstand op de zwarte stippelijn(optische as). De pixelafstand volgens de x-as van de optische as tot het centrum van de bounding box van een object is gedefinieerd als  $x_i$ .

Formule 3.3 toont dat voor het verband tussen  $x_i$  gemeten in pixels en  $\alpha_i$  een extra constante parameter  $l_f$  nodig is.  $l_f$  is één van de camera specifieke parameters, die de afstand tussen de camerasensor en het beeldvlak definieert, namelijk de focale lengte of brandpuntsafstand. Deze parameter kan verkregen worden door middel van een camerakalibratie, en moet omgerekend



**Figuur 3.6:** Schematische voorstelling omgevingsmeting

worden naar pixels.

$$\alpha_i = \tan^{-1}\left(\frac{x_i}{l_f}\right) \quad (3.3)$$

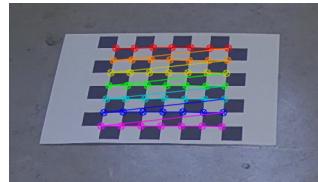
### 3.3.1 Camera kalibratie

Elke camera heeft een set van intrinsieke parameters die uniek zijn voor de camera/lens combinatie. Deze parameters kunnen voorgesteld worden door middel van een cameramatrix  $M$  zoals weergegeven in 3.4 waarbij  $(f_x, f_y)$  de focale lengtes zijn, en  $(c_x, c_y)$  de optische centra.

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

Een eenvoudige methode om een camerakalibratie uit te voeren is een gekend patroon fotograferen uit verschillende hoeken en zo de verschillende parameters te berekenen. Het meestgebruikte patroon voor deze toepassing is een dambordpatroon met afwisselend witte en zwarte vlakken, de

raakpunten van de zwarte vlakken vormen hier een identificeerbaar patroon. In figuur 3.7 is een voorbeeld opgenomen van de patroondetectie.



**Figuur 3.7:** Detectie van het dambordpatroon voor camerakalibratie

### 3.4 Semantische kaart

De semantische kaart is een belangrijk onderdeel van dit werk. De kaart bevat een representatie van alle objecten/features die zichtbaar zijn binnen in de ruimtes. Er is gekozen om de gegevens voor te stellen in het OpenStreetMap (OSM) formaat waarbij alles wordt voorgesteld door middel van nodes in een XML formaat. In figuur 3.8 is een deel uit de kaart gevisualiseerd met behulp van een OSM map editor.

Het OSM formaat definieert alles met behulp van nodes. Elke node kan beschikken over een aantal tags waarin sleutel-waarde gegevens gelinkt kunnen worden aan de nodes. OSM is een formaat dat gebruikt wordt voor andere doeleinden dan deze toepassing, daarom hebben we gekozen om de nodes op maat gemaakte tags toe te kennen die onze eigen parser later kan interpreteren. Aan de kaart zijn een aantal dingen toegevoegd om deze later te kunnen gebruiken. De objecten zijn als tag toegevoegd via een map editor, en de tags zijn ingevuld in een vast patroon. Het resultaat van deze aanpassing in XML is weergegeven in listing 3.2. De belangrijkste gegevens zijn het type en de name tag, deze geven weer dat het gaat over een 'object' met als label 'licht'. Alle gegevens in de node tag zelf zijn geëcreëerd door de map editor bij het aanmaken van de node, waarbij 'lat' en 'lon' de wereldcoördinaten zijn van de node.

**Listing 3.2:** XML voorstelling van 1 object node

```
<node id='137883' action='modify' visible='true' lat='51.068085' lon='4.500029'>
    <tag k='id' v='19' />
    <tag k='name' v='licht' />
    <tag k='type' v='object' />
</node>
```

Een 2de aanpassing aan de kaart is de beschrijving van een te volgen route in het midden van de gangen. Een route bestaat uit een opeenvolging van nodes die aan elkaar grenzen. Een aantal van deze nodes zijn toegevoegd om discrete locaties te verkrijgen die op 1 route liggen. De XML structuur van een locatie node is zichtbaar in listing 3.3. Een locatie node is gekenmerkt door een type='location' en bezit een aantal extra tags.



Figuur 3.8: Grafische voorstelling semantische kaart

De extra tags zijn het belangrijkste onderdeel van de kaart, zij bevatten informatie over de objecten die zichtbaar zijn van op een specifieke locatie. Zo geeft bijvoorbeeld een tag met de sleutel 21 en waarde 10.5 aan dat object 21 zichtbaar is onder een hoek van ongeveer 10.5°.

**Listing 3.3:** XML voorstelling van 1 lokatie node

```
<node id='137973' action='modify' visible='true' lat='51.068085' lon='4.500029'>
    <tag k='id' v='27' />
    <tag k='type' v='location' />
    <tag k='20' v='20' />
    <tag k='21' v='10.5' />
</node>
```

Een route of 'way' in OSM is een gesorteerde lijst van alle node id's die op de route liggen. Dit wordt gebruikt om het pad dat de robot zal volgen aan te geven. In listing 3.4 is in XML formaat een deel van de route beschreven. De aangemaakte locatie nodes zijn via de map editor gekoppeld in één route, deze route zal ingelezen worden door het programma, en is de enige route die de robot kan afleggen.

**Listing 3.4:** XML voorstelling van een route

```
<way id='140546' action='modify' visible='true'>
    <nd ref='4427' />
    <nd ref='137973' />
    <nd ref='137925' />
</way>
```

Het inlezen van het OSM kaartformaat gebeurt via een zelfgemaakte parser die de object, locatie en route nodes inleest in een gelinkte datastructuur zodat het eenvoudig is om later te verwerken. De kaart kan gevisualiseerd worden met behulp van een Python OSM bibliotheek<sup>5</sup>.

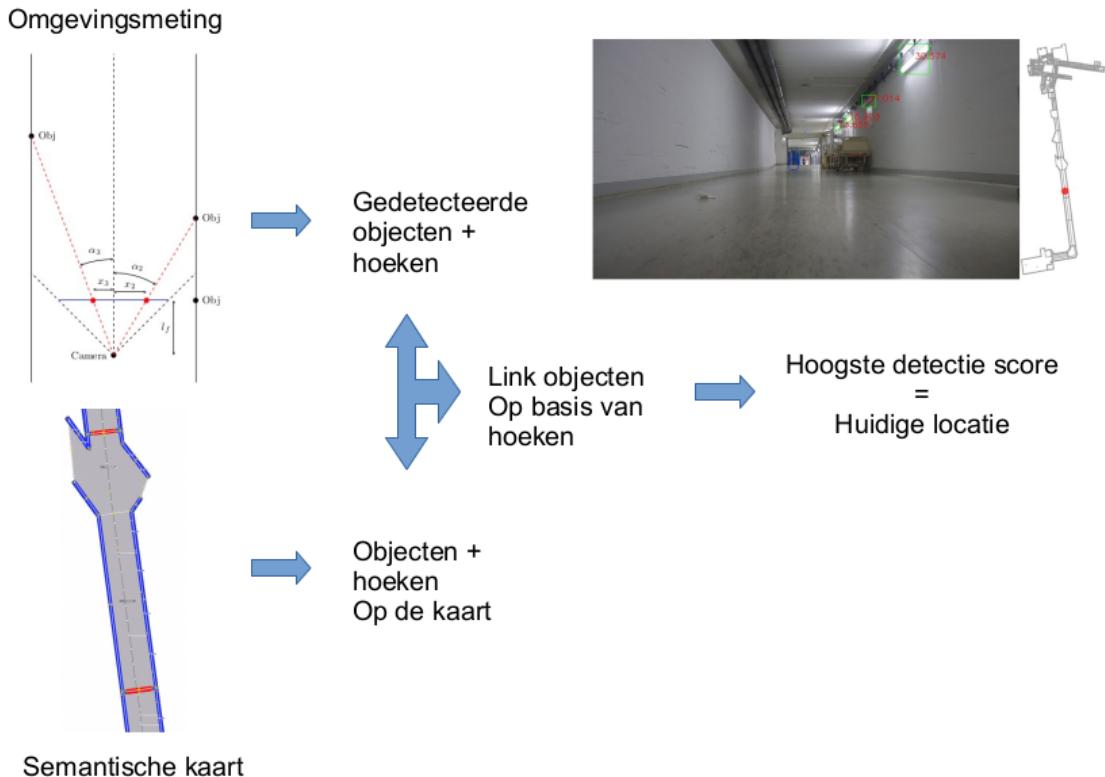
## 3.5 Lokalisatie

Nu we alle stappen afzonderlijk besproken hebben, is het tijd om alles te combineren tot één geheel. Zoals in figuur 3.1 al weergegeven werd, worden de input beelden van de camera frame per frame verwerkt. Eén input afbeelding wordt eerst geanalyseerd door de YOLO object detector waarna het perspectiefpunt gedetecteerd kan worden. Het perspectiefpunt zou in principe ongeveer het midden van het beeld moeten aangeven. Dit is echter niet altijd het geval omdat de robot geroteerd kan zijn ten opzichte van de gang. Daarom wordt er een correctiefactor berekend om het perspectiefpunt ongeveer in het midden van het beeld te leggen.

De 2 detectieresultaten kunnen vervolgens gecombineerd worden om de omgevingsmeting uit te

---

<sup>5</sup><https://github.com/gboeing/osmnx>



**Figuur 3.9:** Overzicht van de lokalisatie

voeren zoals beschreven in hoofdstuk 3.3. Dit geeft als resultaat een lijst van gedetecteerde objecten samen met de hoek waaronder ze zichtbaar zijn.

De omgevingsmeting en de gegevens beschreven op de semantische kaart moeten nu aan elkaar gelinkt worden om de actuele locatie van de robot te bepalen. We gaan er vanuit dat de robot vertrekt op een gekende locatie, deze locatie is één van de discrete locaties aangeduid op de kaart in hoofdstuk 3.4. De eerste vorm van kennis ingebracht in de implementatie is het feit dat de robot niet kan springen, met andere woorden de robot kan enkel op of tussen 2 discrete locatie punten op de kaart zijn. In het programma wordt dus enkel getracht om de huidige locatie te onderscheiden tussen de vorige locatie node, en de volgende locatie node op de gedefinieerde route. Figuur 3.9 geeft het overzicht van hoe de lokalisatie verloopt.

Voor elk van deze 2 mogelijke nodes wordt vervolgens een score berekend, deze score geeft aan hoeveel de theoretische node lijkt op de omgevingsmeting van het huidige frame. Om deze score te berekenen, wordt er eerst geprobeerd om per object-klasse objecten van op de kaart te linken aan objecten uit de omgevingsmeting. Dit gebeurt door de hoeken te vergelijken. De objecten waarvan de detectie en de data hoeken het dichtst bij elkaar liggen worden gelinkt. Uiteraard zijn de metingen niet exact gelijk aan de data, daarom wordt er een afwijking tot 20% toegestaan bij het linken.

De score die de gelijkenis tussen de meting en data weergeeft wordt berekend op basis van deze gelinkte gegevens. Elk object uit de node data dat niet gelinkt kon worden met een detectie wordt

afgestraft, en elke match wordt beloond. Vervolgens wordt per match het verschil in hoek (fout) tussen de detectie en data hoek afgetrokken van de score. De laatste stap is het wegen van de verschillende features. Objecten dichtbij hebben een veel grotere detectiehoek, die nauwkeuriger berekend kan worden. Daarom wordt er op basis van de hoek een wegingsfactor toegevoegd voor dat specifieke object ten opzichte van de score. Objecten veraf die misschien nauwelijks zichtbaar zijn die niet gedetecteerd worden straffen zo bijvoorbeeld minder af dan een object dat dichtbij is en zeker gedeteceerd had moeten worden.

De berekening van de score kan worden samengevat in formule 3.5 waarbij  $a$  de positieve beloningsfactor is,  $b$  de negatieve afstraffactor,  $\alpha_{mi}$  is een hoek uitgelezen van de semantische kaart en  $\alpha_{di}$  een door detectie gemeten hoek.  $n$  staat voor het aantal links tussen detectie en data hoeken en  $m$  is het aantal niet gelinkte objecten. Met andere woorden  $n + m$  is het totaal aantal objecten zichtbaar op de discrete locatie van de kaart. Functie 3.6 genereert de wegingsfactor van de invloed van een object op de score, zoals eerder besproken krijgen objecten met grote hoeken een grotere invloed op de score.

Deze score wordt berekend voor de 2 kandidaat locaties, en de node met de hoogste score wordt beschouwd als de actuele locatie. Vervolgens wordt een nieuw input frame verwerkt, en zal de berekening opnieuw beginnen.

$$score = \sum_{i=1}^n [a - |\alpha_{mi} - \alpha_{di}|] \cdot f_w(\alpha_{mi}) + \sum_{i=1}^m [b \cdot f_w(\alpha_{mi})] \quad (3.5)$$

$$f_w(\alpha) = \begin{cases} 1 & \text{for } 30 \leq |\alpha| \\ 0.75 & \text{for } 15 \leq |\alpha| < 30 \\ 0.5 & \text{for } 8 \leq |\alpha| < 15 \\ 0.3 & \text{for } 0 \leq |\alpha| < 8 \end{cases} \quad (3.6)$$

# **Hoofdstuk 4**

# **Resultaten**

## **4.1 Object detectie**

Zoals besproken in hoofdstuk 3.1 wordt er gebruik gemaakt van YOLOv2 met een aangepaste versie van darknet geïmplementeerd in C. Voor de training hebben we gebruik gemaakt van het standaard trainingsprogramma, voor het gebruik of inferentie van het netwerk maken we gebruik van de Python api via de darknet 'shared library'.

De training en inferentie zijn beide gebeurd op een computer met de specificaties weergegeven in tabel 4.1.

**Tabel 4.1** Computer specificaties

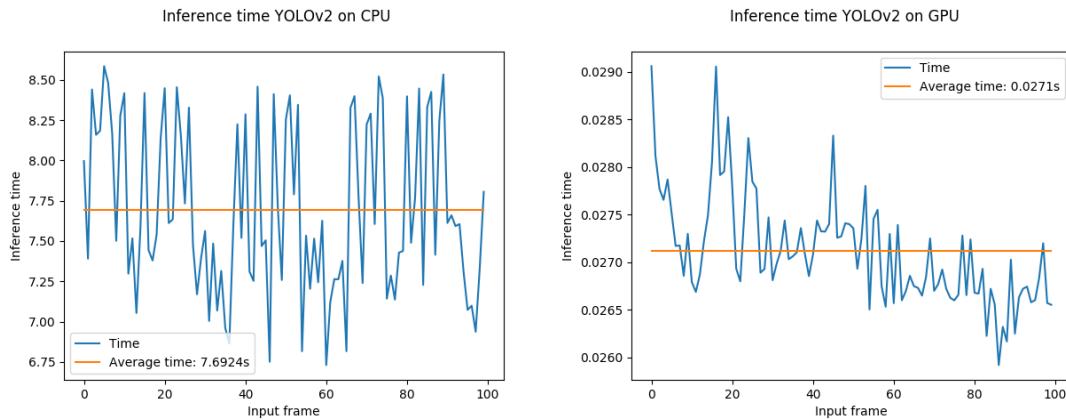
	Model	Kloksnelheid	Geheugen
CPU	Ryzen 5 1600	4.0Ghz	16 GB DDR4
GPU	GTX 1070	1.5Ghz	8 GB GDDR5

### **4.1.1 Training**

We hebben 2 verschillende detectors getraind namelijk een detector op alle 4 de klassen (light, smoke\_detector, exit\_sign en door\_handle), en een detector die enkel lampen kan detecteren. Beide detectors zijn getraind op dezelfde dataset van 899 frames zoals reeds weergegeven in tabel 3.1. Voor de 2de detector hebben we de annotatie files aangepast zodat ze nog enkel lampen bevatten. De detectors hebben beide een training gehad van 36000 iteraties met 64 afbeeldingen per iteratie.

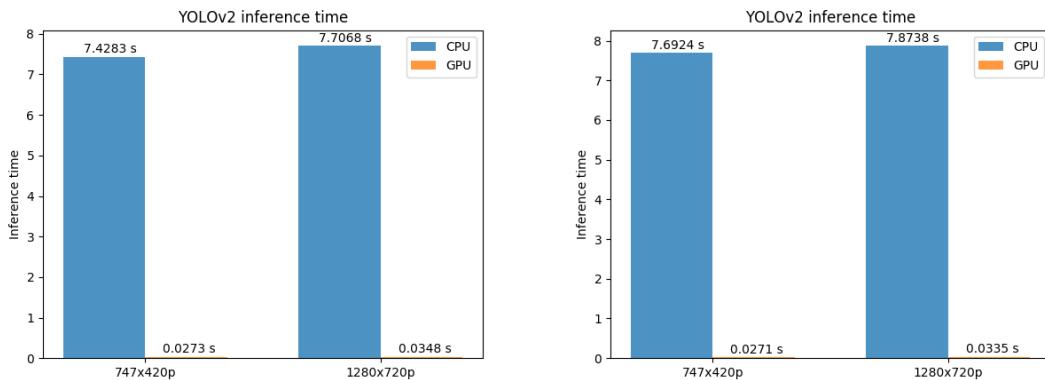
### **4.1.2 Snelheid**

De belangrijkste reden dat we voor de YOLOv2 detector gekozen hebben is de mogelijk om real-time te infereren. Om dit aan te tonen is er een random sample van 100 frames genomen uit de datasets. Alle waarden zijn een gemiddelde van deze 100 afbeeldingen.



**Figuur 4.1:** Inferentie van YOLOv2 detector met 1 klasse op 100 afbeeldingen.

In figuur 4.1 hebben we dezelfde 100 afbeeldingen geïnfereerd met de detector die enkel lampen detecteert. Hierop kunnen we duidelijk afleiden dat de Central Processing Unit (CPU) versie niet bruikbaar is voor real-time gebruik, maar de Graphical Processing Unit (GPU) versie met zijn 0.0271s inferentie tijd of  $36.9\text{fps}$  zeker bruikbaar is. We zullen verder enkel de resultaten verkregen via GPU gebruiken omdat deze substantieel beter zijn.



**Figuur 4.2:** Inferentie van YOLOv2 detector op verschillende inputformaten. **Links:** 4 klassen, **Rechts:** 1 klasse

Een belangrijke factor die invloed heeft op de inferentietijd is de grootte van de afbeeldingen die aan de detector als input gegeven worden. In figuur 4.2 hebben we dezelfde input frames getest met een verschillende grootte, namelijk 1280x720 pixels en 747x420 pixels. Zo kunnen we besluiten dat het verkleinen van de input frames met een factor 1.7 in de breedte en de hoogte een snelheidswinst kan opleveren van ongeveer 20%.

Een 2de conclusie die we kunnen opmaken uit figuur 4.2 is dat het aantal detectieklassen slechts een kleine invloed heeft op de snelheid van de detector.

### 4.1.3 Nauwkeurigheid

Buiten de snelheid van de object detector is ook de nauwkeurigheid van belang. Het testen van de nauwkeurigheid gebeurd door middel van een 2de dataset genaamd de validatieset. Deze validatieset bevat afbeeldingen genomen met dezelfde camera van een ander deel van het gebouw, er zijn dus geen frames die de detector reeds gezien heeft tijdens de training.

De testprocedure begint door de detector te laten lopen op 1 frame, en de annotaties voor hetzelfde frame in te lezen. Voor elke detectie wordt er gekeken of het label overeenkomt met het label in de annotatie. Een 2de metriek waarmee rekening gehouden wordt is de Intersect Over Union(Intersect Over Union (IoU)), zoals geïllustreerd in figuur 4.3 is de IoU het oppervlak overeenkomstig tussen de detectie en de annotatie. Een overlap van 100% is perfect, een overlap van 0% is slecht.

Uit deze 2 metrieken wordt de Mean Average Precision (mAP) berekend, deze waarde geeft de oppervlakte onder de pr-curve. De exacte berekeningen voor deze metrieken zijn beschreven in [6].

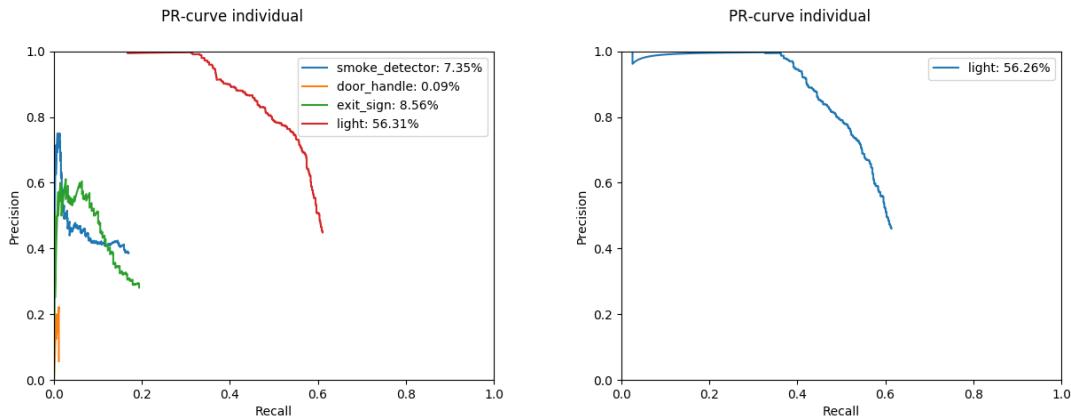
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

**Figuur 4.3:** Grafische voorstelling IoU

De pr-curves en mAP waarden van onze detector zijn weergegeven in figuur 4.4.

Uit de grafieken kunnen we een verband zien tussen de annotaties op het beeldmateriaal beschreven in tabel 3.1 en de nauwkeurigheid van de detector voor bepaalde klassen. De klasse met het grootste aantal voorbeelden namelijk 'light' haalt de beste score. De score voor 'door\_handle' of deurklink is nagenoeg 0, dit is niet verwonderlijk aangezien er slechts een paar voorbeelden waren in de dataset. Dit object geeft dus geen meerwaarde voor de detector of voor het geheel.

Om deze reden is er getest of een detector met één enkele klasse betere resultaten zou opleveren. De logische keuze voor de nieuwe klasse is uiteraard 'light' omdat deze de hoogste precisie haalt, maar ook het belangrijkste object is voor de lokalisatie. In figuur 4.4 zien we echter dat dit weinig verschil maakt, de mAP van de 2 detectors voor het lampobject is vergelijkbaar.

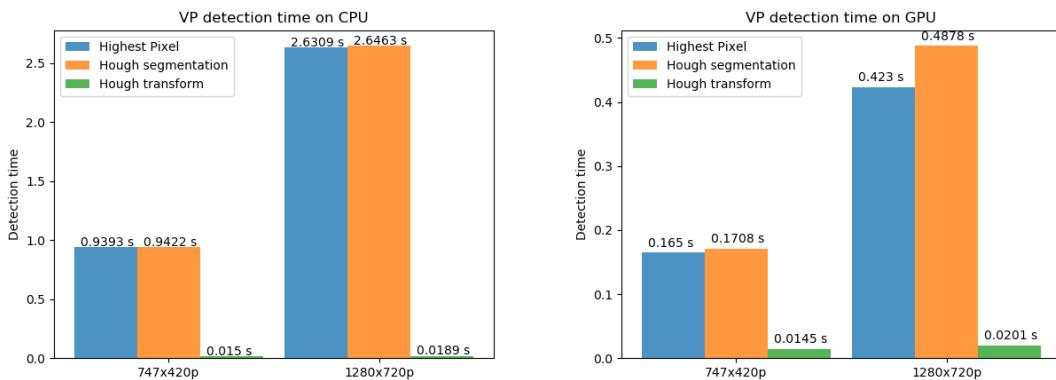


**Figuur 4.4:** Precision-Recall curves van YOLOv2 detector met mAP in de legende. **Links:** detector met 4 klassen. **Rechts:** detector met 1 klasse

## 4.2 Perspectiefpunt detectie

In hoofdstuk 3.2 hebben is besproken hoe we drie verschillende methoden van perspectiefpunt detectie geïmplementeerd hebben. Hier gaan we de 3 technieken vergelijken op nauwkeurigheid en snelheid.

### 4.2.1 Snelheid



**Figuur 4.5:** Detectiesnelheid van 3 perspectiefpunt detectiemethoden op CPU en GPU

In figuur 4.5 zijn detectiesnelheden van de 3 detectietechnieken weergegeven. Voor deze resultaten is het gemiddelde genomen van de detectietijd van 100 willekeurig gekozen input afbeeldingen uit de dataset. Elke afbeelding is 4 keer door een specifieke detector gevalideerd namelijk op CPU en GPU alsook op 2 verschillende resoluties. De verschillende resoluties zijn 1280x720 pixels en 747x420 pixels.

Net zoals bij de objectdetector zien we hier ook een aanzienlijk verschil in detectietijd tussen CPU

en GPU voor 2 van de 3 technieken. Dit is niet verwonderlijk aangezien deze methoden gebruik maken van een segmentatie netwerk. Het verschil in tijd tussen CPU en GPU is zo groot dat we verder dus ook enkel de GPU implementatie gaan gebruiken en bespreken.

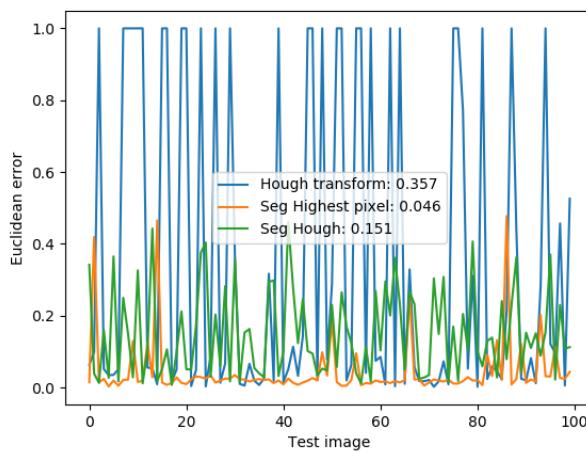
Op de linkse grafiek in figuur 4.5 is te zien dat de perspectieflijn kruising techniek met een grote marge de snelste techniek is, dit komt omdat deze techniek gebaseerd is op klassieke beeldverwerkings technieken en geen gebruik maakt van een neuraal netwerk. Hierbij is dan ook geen verschil tussen het detecteren op CPU en GPU omdat er geen GPU implementatie is.

De hoogste vloerpixel techniek is de 2de snelste, deze techniek is een fractie sneller door zijn eenvoud na het segmenteren van de vloer. Het zoeken van de hoogste pixel neemt amper tijd in beslag en het grootste deel van de tijd komt omwille van de segmentatie.

In figuur 4.5 is ook duidelijk zichtbaar dat de resolutie van de input afbeelding een grote invloed heeft op de detectiesnelheid van het gebruikte segmentatienetwerk. Zo is door de afbeelding bijna te halveren in de breedte en de hoogte een snelheidswinst van ongeveer 60% verkregen.

#### 4.2.2 Nauwkeurigheid

Om de nauwkeurigheid van een perspectiefpunt detector te bepalen is er een script geschreven om voor elk van de 100 willekeurig gekozen input afbeeldingen het perspectiefpunt handmatig aan te duiden. Vervolgens is elke afbeelding getest door de 3 detectoren en het perspectiefpunt bepaald. De coördinaten van het gedetecteerde perspectiefpunt, en het data punt door ons aangeduid, zijn beide in genormaliseerd ten opzichte van de breedte en de hoogte van de input afbeelding. De euclidische afstand tussen de 2 punten is de afwijking of fout van de detector.



**Figuur 4.6:** Afwijking t.o.v. perspectiefpunt voor 3 verschillende detectietechnieken.

In figuur 4.6 is de fout voor elke afbeelding weergegeven alsook de gemiddelde fout in de legende. Hieruit kunnen we afleiden dat de perspectieflijn kruising techniek(3.2.3) ondanks het de snelste techniek is, de laagste nauwkeurigheid haalt. Dit is grotendeels omdat niet voor elke afbeelding de detector erin slaagt een perspectiefpunt te vinden. De reden dat niet in elk frame een punt

gevonden wordt komt door een gebrek aan hough-lines die gevonden worden op basis van de rand-detectie. Door meer met de parameters te spelen zou dit probleem eventueel opgelost kunnen worden om de nauwkeurigheid van deze detector te verhogen.

Ondanks de eenvoudigste techniek te zijn, is de hoogste vloerpixel(3.2.1) methode over heel de lijn het meest nauwkeurig. Een belangrijke reden voor dit resultaat is het excellente resultaat van het gebruikte segmentatienetwerk. Indien de vloersegmentatie van mindere kwaliteit zou zijn, zou deze methode een slechter resultaat opleveren.

De vloerlijn kruising(3.2.2) methode geeft niet het beste resultaat, maar het is gemiddeld gezien zeker aanvaardbaar. Doordat we in hoofdstuk 4.2.1 besloten hebben dat de 2 technieken die gebruik maken van vloersegmentatie ongeveer evenveel rekentijd in beslag nemen, en de hoogste vloerpixel methode de grootste nauwkeurigheid biedt, is deze gekozen voor de uiteindelijke implementatie.

## 4.3 Lokalisatie

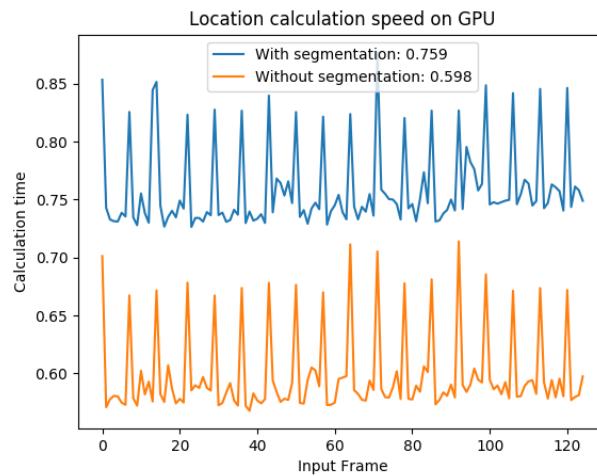
In de lokalisatiestap besproken in hoofdstuk 3.5 komt alles samen. Hier zullen we bekijken hoe goed we een locatie kunnen inschatten op basis van semantische kenmerken in een RGB beeld. Zoals bij de vorige resultaten zijn ook deze opgesplitst in snelheid en nauwkeurigheid.

### 4.3.1 Snelheid

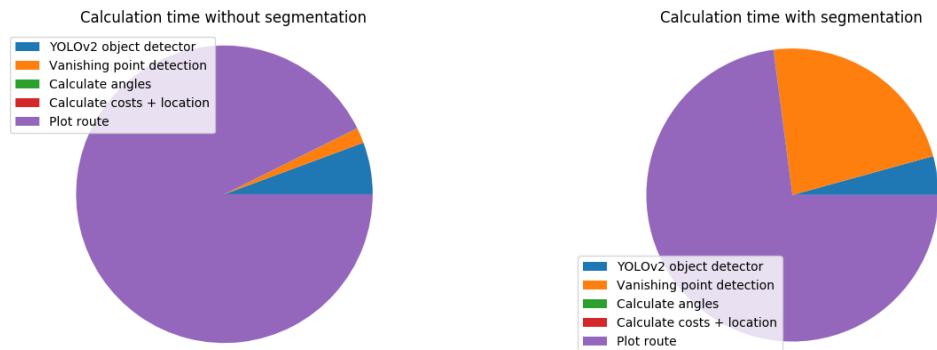
Voor validatie is er een dataset genomen van 125 opeenvolgende frames van een video. De tijden waargenomen voor deze experimenten zijn voor 1 volledige iteratie in het programma, dit houd in:

- YOLOv2 object detectie;
- Perspectiefpunt detectie;
- Lokalisatie berekeningen;
- User interface met frame en locatie op kaart plotten.

In figuur 4.7 is een grafiek te zien van de iteratietijd per frame, in de legende is de gemiddelde tijd weergegeven. Er is uiteraard een redelijk verschil waar te nemen tussen de implementatie die gebruikmaakt van het segmentatienetwerk, en die zonder. Maar de detector zonder segmentatie(perspectiefpunt detectie d.m.v. perspectieflijn kruising 3.2.3) heeft nog steeds een vrij lange rekentijd. Ondanks de theoretische snelheid die we uit vorige hoofdstukken kunnen afleiden ongeveer 0.042 s zou moeten zijn, is de gemiddelde rekentijd meer dan 10 keer hoger. Na profiling van de implementatiecode, zijn we tot de constatatie gekomen dat het grootste deel van de tijd verloren gaat in de bibliotheek die gebruikt wordt om de kaart en de actuele locatie visueel weer te geven. Dit is te zien in figuur 4.8. Hier zien we duidelijk dat de object en perspectiefpunt detector wel een stuk tijd in beslag nemen, maar dat door de visualisatie te laten vallen de snelheid bijna 1/4 van de huidige detectietijd zou kunnen zijn.



**Figuur 4.7:** Berekeningssnelheid van de actuele locatie



**Figuur 4.8:** Gemiddelde profilering van het lokalisatieproces

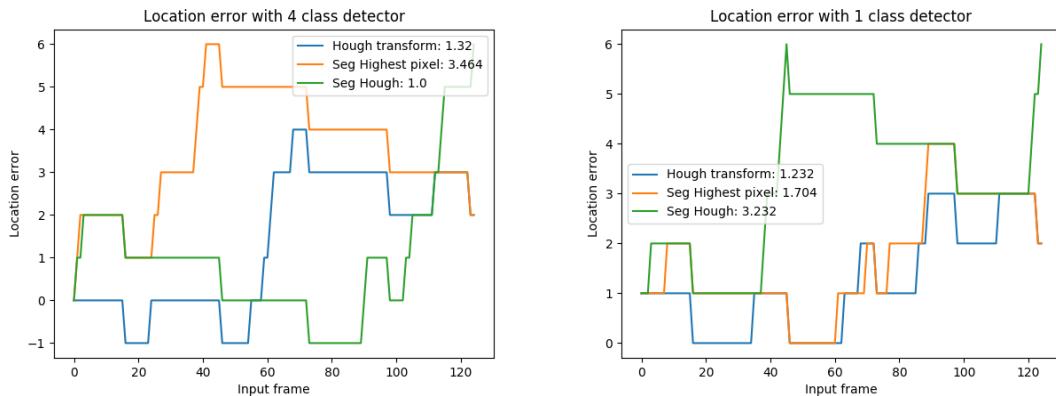
### 4.3.2 Nauwkeurigheid

Voor de validatie is er op een dataset van 125 opeenvolgende frames van een video aangeduid bij welke discrete locatie op de semantische kaart het beeld het beste past. Deze data wordt beschouwd als waarheid. Dit wordt vergeleken met de output locatie per frame van ons programma.

In figuur 4.9 is de fout in locatie weergegeven voor 6 verschillende detector combinaties. Deze combinaties zijn de 2 YOLOv2 detectoren (4 klassen en 1 klasse) samen met de 3 perspectiefpunt detectiemethoden.

De locatiefout is het aantal nodes (beschreven in de route) dat de detector afwijkt van de actuele locatie. Een fout van 0 is dus een juiste predictie, bij een positieve fout loopt de detector vooruit en bij een negatieve fout loopt de detector achter. De legende in figuur 4.9 geeft de gemiddelde fout van de geteste detector weer.

De grafieken geven duidelijk weer dat de lokalisatie niet altijd even goed verloopt. Het algoritme



**Figuur 4.9:** Vergelijking van lokalisatie fouten

heeft de neiging om te vaak vooruit te gaan, behalve bij 1 test is dit niet het geval waardoor de lokalisatie een beetje achterloopt.

De keuze van object detector kan een invloed hebben op de nauwkeurigheid. Dit komt omdat het aantal wel en niet gedetecteerde objecten rechtstreeks gebruikt worden in scoreberekening 3.6. Een object met een grote wegingsfactor dat niet gedetecteerd is, zal zwaar afgestraft worden met een mogelijke lokalisatiefout ten gevolg. Daarom zal het belangrijk zijn om een object detector te gebruiken met een grote nauwkeurigheid.

Ook de keuze van perspectiepunt detector heeft een grote invloed op de lokalisatie nauwkeurigheid. Dit komt omdat de positie van het gedetecteerde punt rechtstreeks gebruikt wordt om de hoeken t.o.v. de gedetecteerde objecten te bepalen. Als de hoeken niet nauwkeurig zijn, is het moeilijk om de metingen te linken aan een positie op de kaart. Als in dit geval, door een slechte meting, de score van de verkeerde locatie hoger wordt, dan wordt er een verkeerde aannname gedaan. Deze ene fout is misschien niet erg, maar wordt wel verder meegezogen in de volgende berekeningen waardoor de fouten zich kunnen opstapelen.

De belangrijkste reden voor de minder goede resultaten van de lokalisatie, is omdat de data in de semantische kaart niet nauwkeurig genoeg is. Bij de start van het implementeren hebben we een semantische kaart verkregen met bijhorend beeldmateriaal van een specifieke omgeving. Op dat moment bevatte de kaart enkel een beschrijving van de muren en deuren. Dit was uiteraard niet genoeg voor de doeleinden die we hier voor ogen hadden, en was het noodzakelijk om de features beschreven in vorige hoofdstukken in de kaart op te nemen. Het was niet mogelijk om van deze objecten de echte afmetingen en locaties te verkrijgen, daarom is er besloten om op de kaart de afstanden en hoeken tot de objecten te schatten. Deze schatting is gebeurd op basis van het verkregen beeldmateriaal.

Doordat onze data die slechts een schatting is, is het zeer moeilijk om exact te zeggen hoe nauwkeurig onze detector is.

## **Hoofdstuk 5**

### **Besluit**

# Bibliografie

- [1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Szeliski. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, Nov 2012.
- [2] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla, and Senior Member. SegNet : A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. pages 1–14.
- [3] Bhakti Baheti, Ujjwal Baid, and Sanjay Talbar. An approach to automatic object tracking system by combination of SIFT and RANSAC with mean shift and KLT. *Conference on Advances in Signal Processing, CASP 2016*, pages 254–259, 2016.
- [4] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986.
- [5] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, April 2018.
- [6] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.
- [7] Chiung Yao Fang, Sei Wang Chen, and Chiou Shann Fuh. Road-sign detection and tracking. *IEEE Transactions on Vehicular Technology*, 52(5):1329–1341, 2003.
- [8] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *In the 12th International Symposium on Experimental Robotics (ISER*, 2010.
- [9] Yinxiao Li and Stanley T. Birchfield. Image-based segmentation of indoor corridor floors for a mobile robot. *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pages 837–843, 2010.
- [10] Adrian Llopart, Ole Ravn, and Nils A. Andersen. Door and cabinet recognition using Convolutional Neural Nets and real-time method fusion for handle detection and grasping. *2017 3rd International Conference on Control, Automation and Robotics, ICCAR 2017*, pages 144–149, 2017.

- [11] D G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, sep 1999.
- [12] Guanghan Ning. Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking. (1):1–4, 2017.
- [13] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.
- [14] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.
- [15] F. Geovani Rodríguez-Telles, L. Abril Torres-Méndez, and Edgar A. Martínez-García. A fast floor segmentation algorithm for visual-based robot navigation. *Proceedings - 2013 International Conference on Computer and Robot Vision, CRV 2013*, pages 167–173, 2013.
- [16] K. Schmid, T. Tomic, F. Ruess, H. Hirschmüller, and M. Suppa. Stereo vision based indoor/outdoor navigation for flying robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3955–3962, Nov 2013.
- [17] Shuran Song, Samuel P. Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [18] R. Swathiha and T. S. Sharmila. Emergency exit sign detection system for visually impaired people. In *2016 International Conference on Inventive Computation Technologies (ICICT)*, volume 1, pages 1–7, Aug 2016.
- [19] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. 1991.
- [20] M Tomono and S Yuta. Mobile robot navigation in indoor environments using object and character recognition. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 313–320 vol.1, apr 2000.
- [21] S. J. Zabihi, S. M. Zabihi, S. S. Beauchemin, and M. A. Bauer. Detection and recognition of traffic signs inside the attentional visual field of drivers. *IEEE Intelligent Vehicles Symposium, Proceedings*, (IV):583–588, 2017.
- [22] Zhong-Ju Zhang. Wall, floor, ceiling, object region identification from single image.
- [23] Huiyu Zhou, Yuan Yuan, and Chunmei Shi. Object tracking using SIFT features and mean shift. *Computer Vision and Image Understanding*, 113(3):345–352, 2009.

## Bijlage A

# CVAT naar YOLO conversie

```
import xml.etree.ElementTree as ET
import argparse
import os

parser = argparse.ArgumentParser()
parser.add_argument("file")
args = parser.parse_args()

classes = []

def getLabels(root):
    labels = root.find('task').find('labels')

    for label in labels:
        classes.append(label.find('name').text)

    with open('cust.names', 'w') as names:
        names.write("\n".join(classes))

if __name__ == '__main__':
    tree = ET.parse(args.file)
    root = tree.getroot()

    print(root.tag)

    if not os.path.isdir('Annotations-yolo'):
        os.mkdir('Annotations-yolo')

    for child in root:
        if child.tag == 'meta':
            getLabels(child)

        if child.tag == 'image':
            image = child.get('name')
            w = float(child.get('width'))
            h = float(child.get('height'))

            boxes = ''
            for box in child.findall('box'):
                xtl = float(box.get('xtl'))
                ytl = float(box.get('ytl'))
                xbr = float(box.get('xbr'))
                ybr = float(box.get('ybr'))

                width = xbr - xtl
                height = ybr - ytl
                x = xtl + (width/2)      #center
                y = ytl + (height/2)

                boxes += '{}-{}-{}-{}{}\n'.format(classes.index(box.get('label')),
                                                x/w, y/h, width/w, height/h)

            with open('Annotations-yolo/{}.txt'.format(os.path.splitext(image)[0]), 'w') as file:
```

```
file.write(boxes)
```

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN  
CAMPUS DE NAYER SINT-KATELIJNE-WAVER  
J. De Nayerlaan 5  
2860 SINT-KATELIJNE-WAVER, België  
tel. + 32 15 31 69 44  
[iw.denayer@kuleuven.be](mailto:iw.denayer@kuleuven.be)  
[www.iw.kuleuven.be](http://www.iw.kuleuven.be)

